OpenCore

Reference Manual (0.6.~~0~~.1)

[2020.08.29]

2 Configuration

2.1 Configuration Terms

- `OC config` — OpenCore Configuration file in `plist` format named `config.plist`. It has to provide extensible way to configure OpenCore and is structured to be separated into multiple named sections situated in the root `plist dictionary`. These sections are permitted to have `plist array` or `plist dictionary` types and are described in corresponding sections of this document.
- `valid key` — `plist key` object of `OC config` described in this document or its future revisions. Besides explicitly described `valid keys`, keys starting with `#` symbol (e.g. `#Hello`) are also considered `valid keys` and behave as comments, effectively discarding their value, which is still required to be a valid `plist object`. All other `plist keys` are not valid, and their presence yields to `undefined behaviour`.
- `valid value` — valid `plist object` of `OC config` described in this document that matches all the additional requirements in specific `plist object` description if any.
- `invalid value` — valid `plist object` of `OC config` described in this document that is of other `plist type`, does not conform to additional requirements found in specific `plist object` description (e.g. value range), or missing from the corresponding collection. `Invalid value` is read with or without an error message as any possible value of this `plist object` in an undetermined manner (i.e. the values may not be same across the reboots). Whilst reading an `invalid value` is equivalent to reading certain defined `valid value`, applying incompatible value to the host system may yield to `undefined behaviour`.
- `optional value` — `valid value` of `OC config` described in this document that reads in a certain defined manner provided in specific `plist object` description (instead of `invalid value`) when not present in `OC config`. All other cases of `invalid value` do still apply. Unless explicitly marked as `optional value`, any other value is required to be present and reads to `invalid value` if missing.
- `fatal behaviour` — behaviour leading to boot termination. Implementation must stop the boot process from going any further until next host system boot. It is allowed but not required to perform cold reboot or show any warning message.
- `undefined behaviour` — behaviour not prescribed by this document. Implementation is allowed to take any measures including but not limited to `fatal behaviour`, assuming any states or values, or ignoring, unless these measures negatively affect system security in general.

2.2 Configuration Processing

`OC config` is guaranteed to be processed at least once if it was found. Depending on OpenCore bootstrapping mechanism multiple `OC config` files may lead to reading any of them. No `OC Config` may be present on disk, in which case all the values read follow the rules of `invalid value` and `optional value`.

`OC config` has size, nesting, and key amount limitations. `OC config` size does not exceed `16 MBs`. `OC config` has no more than `8` nesting levels. `OC config` has up to `16384` XML nodes (i.e. one `plist dictionary` item is counted as a pair of nodes) within each `plist object`.

Reading malformed `OC config` file leads to `undefined behaviour`. Examples of malformed `OC config` cover at least the following cases:

- files non-conformant to `plist` DTD
- files with unsupported or non-conformant `plist objects` found in this document
- files violating size, nesting, and key amount limitations

It is recommended but not required to abort loading malformed `OC config` and continue as if no `OC config` was present. For forward compatibility it is recommended but not required for the implementation to warn about the use of `invalid values`. Recommended practice of interpreting `invalid values` is to conform to the following convention where applicable:

~~Type~~

Type	~~Value~~Value
plist string	Empty string (<string></string>)
plist data	Empty data (<data></data>)
plist integer	0 (<integer>0</integer>)
plist boolean	False (<false/>)
plist tristate	False (<false/>)

2.3 Configuration Structure

OC config is separated into following sections, which are described in separate sections of this document. By default it is tried to not enable anything and optionally provide kill switches with Enable property for plist dict entries. In general the configuration is written idiomatically to group similar actions in subsections:

- Add provides support for data addition. Existing data will not be overridden, and needs to be handled separately with Delete if necessary.
- Delete provides support for data removal.
- Patch provides support for data modification.
- Quirks provides support for specific hacks.

Root configuration entries consist of the following:

- ACPI
- Booter
- DeviceProperties
- Kernel
- Misc
- NVRAM
- PlatformInfo
- UEFI

It is possible to perform basic validation of the configuration by using ocvalidate utility. Please note, that ocvalidate must match the used OpenCore release and may not be able to detect all configuration flaws present in the file.

Note: Currently most properties try to have defined values even if not specified in the configuration for safety reasons. This behaviour should not be relied upon, and all fields must be properly specified in the configuration.

3.3 Contribution

OpenCore can be compiled as an ordinary EDK II package. Since UDK development was abandoned by TianoCore, OpenCore requires the use of EDK II Stable. Currently supported EDK II release is hosted in acidanthera/audk. The required patches for the package are present in `Patches` directory.

The only officially supported toolchain is `XCODE5`. Other toolchains might work, but are neither supported, nor recommended. Contribution of clean patches is welcome. Please do follow EDK II C Codestyle.

To compile with `XCODE5`, besides Xcode, one should also install NASM and MTOC. The latest Xcode version is recommended for use despite the toolchain name. Example command sequence may look as follows:

```
git clone --recursive --depth=1 https://github.com/acidanthera/audk UDK
git clone --depth=1 https://github.com/acidanthera/audk UDK
cd UDK
git clone https://github.com/acidanthera/OpenCorePkg
git submodule update --init --recommend-shallow
git clone --depth=1 https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

For IDE usage Xcode projects are available in the root of the repositories. Another approach could be Sublime Text with EasyClangComplete plugin. Add `.clang_complete` file with similar content to your UDK root:

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/MdeModulePkg
-I/UefiPackages/MdeModulePkg/Include
-I/UefiPackages/MdeModulePkg/Include/X64
-I/UefiPackages/OpenCorePkg/Include/AMI
-I/UefiPackages/OpenCorePkg/Include/Acidanthera
-I/UefiPackages/OpenCorePkg/Include/Apple
-I/UefiPackages/OpenCorePkg/Include/Apple/X64
-I/UefiPackages/OpenCorePkg/Include/Duet
-I/UefiPackages/OpenCorePkg/Include/Generic
-I/UefiPackages/OpenCorePkg/Include/Intel
-I/UefiPackages/OpenCorePkg/Include/Microsoft
-I/UefiPackages/OpenCorePkg/Include/VMware
-I/UefiPackages/OvmfPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
-DNO_MSABI_VA_FUNCS=1
```

Listing 2: ECC Configuration

Codestyle. The codebase follows EDK II codestyle with few changes and clarifications.

- Write inline documentation for the functions and variables only once: in headers, where a header prototype is available, and inline for `static` variables and functions.
- Use line length of 120 characters or less, preferably 100 characters.
- Use spaces after casts, e.g. `(VOID *)(UINTN) Variable`.
- Use SPDX license headers as shown in acidanthera/bugtracker#483.

3.5 Debugging

The codebase incorporates EDK II debugging and few custom features to improve the experience.

- Use module prefixes, 2-5 letters followed by a colon (`:`), for debug messages. For `OpenCorePkg` use `OC:`, for libraries and drivers use their own unique prefixes.
- Do not use dots (`.`) in the end of debug messages and separate `EFI_STATUS`, printed by `%r`, with a hyphen (e.g. `OCRAM: Allocation of %u bytes failed - %r\n`).
- Use `DEBUG_CODE_BEGIN ()` and `DEBUG_CODE_END ()` constructions to guard debug checks that may potentially reduce the performance of release builds and are otherwise unnecessary.
- Use `DEBUG` macro to print debug messages during normal functioning, and `RUNTIME_DEBUG` for debugging after `EXIT_BOOT_SERVICES`.
- Use `DEBUG_VERBOSE` debug level to leave debug messages for future debugging of the code, which are currently not necessary. By default `DEBUG_VERBOSE` messages are ignored even in `DEBUG` builds.
- Use `DEBUG_INFO` debug level for all non critical messages (including errors) and `DEBUG_BULK_INFO` for extensive messages that should not appear in NVRAM log that is heavily limited in size. These messages are ignored in `RELEASE` builds.
- Use `DEBUG_ERROR` to print critical human visible messages that may potentially halt the boot process, and `DEBUG_WARN` for all other human visible errors, `RELEASE` builds included.

When trying to find the problematic change it is useful to rely on `git-bisect` functionality. There also are some unofficial resources that provide per-commit binary builds of OpenCore, like Dortania.

7 Kernel

7.1 Introduction

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

7.2 Properties

1. `Add`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected kernel drivers from `OC/Kexts` directory.

   Designed to be filled with `plist dict` values, describing each driver. See Add Properties section below. Kernel driver load order follows the item order in the array, thus the dependencies should be written prior to their consumers.

   To track the dependency order one can inspect the `OSBundleLibraries` key in the `Info.plist` of the kext. Any kext mentioned in the `OSBundleLibraries` of the other kext must be precede this kext.

   Note: Kexts may have inner kexts (`Plug-Ins`) in their bundle. Each inner kext must be added separately.

2. `Block`
   Type: `plist array`
   Failsafe: Empty
   Description: Remove selected kernel drivers from prelinked kernel.

   Designed to be filled with `plist dictionary` values, describing each blocked driver. See Block Properties section below.

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

4. `Force`
   Type: `plist array`
   Failsafe: Empty
   Description: Load kernel drivers from system volume if they are not cached.

   Designed to be filled with `plist dict` values, describing each driver. See Force Properties section below. This section resolves the problem of injecting drivers that depend on other drivers, which are not cached otherwise. The issue normally affects older operating systems, where various dependency kexts, like `IOAudioFamily` or `IONetworkingFamily` may not be present in the kernel cache by default. Kernel driver load order follows the item order in the array, thus the dependencies should be written prior to their consumers. `Force` happens before `Add`.

   Note: The signature of the "forced" kernel drivers is not checked anyhow, making the use of this feature extremely dangerous and undesired for secure boot. This feature may not work on encrypted partitions in newer operating systems.

5. `Patch`
   Type: `plist array`
   Failsafe: Empty
   Description: Perform binary patches in kernel and drivers prior to driver addition and removal.

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

6. `Quirks`
   Type: `plist dict`
   Description: Apply individual kernel and driver quirks described in Quirks Properties section below.

7. Scheme
   Type: plist dict
   Description: Define kernelspace operation mode via parameters described in Scheme Properties section below.

7.3 Add Properties

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

2. BundlePath
   Type: plist string
   Failsafe: Empty string
   Description: Kext bundle path (e.g. Lilu.kext or MyKext.kext/Contents/PlugIns/MySubKext.kext).

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

4. Enabled
   Type: plist boolean
   Failsafe: false
   Description: This kernel driver will not be added unless set to true.

5. ExecutablePath
   Type: plist string
   Failsafe: Empty string
   Description: Kext executable path relative to bundle (e.g. Contents/MacOS/Lilu).

6. MaxKernel
   Type: plist string
   Failsafe: Empty string
   Description: Adds kernel driver on specified macOS version or older.

   Kernel version can be obtained with uname -r command, and should look like 3 numbers separated by dots, for example 18.7.0 is the kernel version for 10.14.6. Kernel version interpretation is implemented as follows:

$$\begin{aligned} ParseDarwinVersion(\kappa, \lambda, \mu) = \kappa \cdot 10000 & \quad \text{Where } \kappa \in (0, 99) \text{ is kernel version major} \\ + \lambda \cdot 100 & \quad \text{Where } \lambda \in (0, 99) \text{ is kernel version minor} \\ + \mu & \quad \text{Where } \mu \in (0, 99) \text{ is kernel version patch} \end{aligned}$$

   Kernel version comparison is implemented as follows:

$$\alpha = \begin{cases} ParseDarwinVersion(\texttt{MinKernel}), & \text{If MinKernel is valid} \\ 0 & Otherwise \end{cases}$$

$$\beta = \begin{cases} ParseDarwinVersion(\texttt{MaxKernel}), & \text{If MaxKernel is valid} \\ \infty & Otherwise \end{cases}$$

$$\gamma = \begin{cases} ParseDarwinVersion(FindDarwinVersion()), & \text{If valid "Darwin Kernel Version" is found} \\ \infty & Otherwise \end{cases}$$

$$f(\alpha, \beta, \gamma) = \alpha \leq \gamma \leq \beta$$

   Here $ParseDarwinVersion$ argument is assumed to be 3 integers obtained by splitting Darwin kernel version string from left to right by the . symbol. $FindDarwinVersion$ function looks up Darwin kernel version by locating "Darwin Kernel Version $\kappa.\lambda.\mu$" string in the kernel image.

7. `MinKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Adds kernel driver on specified macOS version or newer.

   Note: Refer to `Add MaxKernel` description for matching logic.

8. `PlistPath`
   Type: `plist string`
   Failsafe: Empty string
   Description: Kext `Info.plist` path relative to bundle (e.g. `Contents/Info.plist`).

7.4 Block Properties

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

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

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This kernel driver will not be blocked unless set to `true`.

4. `Identifier`
   Type: `plist string`
   Failsafe: Empty string
   Description: Kext bundle identifier (e.g. `com.apple.driver.AppleTyMCEDriver`).

5. `MaxKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Blocks kernel driver on specified macOS version or older.

   Note: Refer to `Add MaxKernel` description for matching logic.

6. `MinKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Blocks kernel driver on specified macOS version or newer.

   Note: Refer to `Add MaxKernel` description for matching logic.

7.5 Emulate Properties

1. `Cpuid1Data`
   Type: `plist data`, 16 bytes
   Failsafe: All zero
   Description: Sequence of `EAX`, `EBX`, `ECX`, `EDX` values to replace `CPUID (1)` call in XNU kernel.

   This property serves for two needs:

   - Enabling support of an unsupported CPU model.
   - Enabling XCPM support for an unsupported CPU variant.

   Normally it is only the value of `EAX` that needs to be taken care of, since it represents the full CPUID. The remaining bytes are to be left as zeroes. Byte order is Little Endian, so for example, `C3 06 03 00` stands for CPUID `0x0306C3` (Haswell).

For XCPM support it is recommended to use the following combinations.

- Haswell-E (0x0306F2) to Haswell (0x0306C3):
  Cpuid1Data: C3 06 03 00 00 00 00 00 00 00 00 00 00 00 00 00
  Cpuid1Mask: FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
- Broadwell-E (0x0406F1) to Broadwell (0x0306D4):
  Cpuid1Data: D4 06 03 00 00 00 00 00 00 00 00 00 00 00 00 00
  Cpuid1Mask: FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00

Keep in mind, that the following configurations are unsupported (at least out of the box):

- Consumer Ivy Bridge (0x0306A9) as Apple disabled XCPM for Ivy Bridge and recommends legacy power management for these CPUs. You will need to manually patch _xcpm_bootstrap to force XCPM on these CPUs instead of using this option.
- Low-end CPUs (e.g. Haswell+ Pentium) as they are not supported properly by macOS. Legacy hacks for older models can be found in the Special NOTES section of acidanthera/bugtracker#365.

2. Cpuid1Mask
   Type: plist data, 16 bytes
   Failsafe: All zero
   Description: Bit mask of active bits in Cpuid1Data.

   When each Cpuid1Mask bit is set to 0, the original CPU bit is used, otherwise set bits take the value of Cpuid1Data.

7.6 Force Properties

1. Arch
   Type: plist string
   Failsafe: Any
   Description: Kext architecture (Any, i386, x86_64).
2. BundlePath
   Type: plist string
   Failsafe: Empty string
   Description: Kext bundle path (e.g. System\Library \Extensions \IONetworkingFamily.kext).
3. Comment
   Type: plist string
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.
4. Enabled
   Type: plist boolean
   Failsafe: false
   Description: This kernel driver will not be added when not present unless set to true.
5. ExecutablePath
   Type: plist string
   Failsafe: Empty string
   Description: Kext executable path relative to bundle (e.g. Contents/MacOS/IONetworkingFamily).
6. Identifier
   Type: plist string
   Failsafe: Empty string
   Description: Kext identifier to perform presence checking before adding (e.g. com.apple.iokit.IONetworkingFamily). Only drivers which identifiers are not be found in the cache will be added.
7. MaxKernel
   Type: plist string
   Failsafe: Empty string
   Description: Adds kernel driver on specified macOS version or older.

   Note: Refer to Add Add MaxKernel description for matching logic.

8. `MinKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Adds kernel driver on specified macOS version or newer.

   Note: Refer to `Add Add MaxKernel` description for matching logic.

9. `PlistPath`
   Type: `plist string`
   Failsafe: Empty string
   Description: Kext `Info.plist` path relative to bundle (e.g. `Contents/Info.plist`).

7.7 Patch Properties

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

2. `Base`
   Type: `plist string`
   Failsafe: Empty string
   Description: Selects symbol-matched base for patch lookup (or immediate replacement) by obtaining the address of provided symbol name. Can be set to empty string to be ignored.

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

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

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

6. `Find`
   Type: `plist data`
   Failsafe: Empty data
   Description: Data to find. Can be set to empty for immediate replacement at `Base`. Must equal to `Replace` in size otherwise.

7. `Identifier`
   Type: `plist string`
   Failsafe: Empty string
   Description: Kext bundle identifier (e.g. `com.apple.driver.AppleHDA`) or `kernel` for kernel patch.

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

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

10. `MaxKernel`
    Type: `plist string`
    Failsafe: Empty string
    Description: Patches data on specified macOS version or older.

    Note: Refer to `Add MaxKernel` description for matching logic.

11. `MinKernel`
    Type: `plist string`
    Failsafe: Empty string
    Description: Patches data on specified macOS version or newer.

    Note: Refer to `Add MaxKernel` description for matching logic.

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

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

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

7.8 Quirks Properties

1. `AppleCpuPmCfgLock`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.6 (64-bit)
   Description: Disables `PKG_CST_CONFIG_CONTROL` (`0xE2`) MSR modification in AppleIntelCPUPowerManagement.kext, commonly causing early kernel panic, when it is locked from writing.

   Certain firmwares lock `PKG_CST_CONFIG_CONTROL` MSR register. To check its state one can use bundled `VerifyMsrE2` tool. Select firmwares have this register locked on some cores only.

   As modern firmwares provide `CFG Lock` setting, which allows configuring `PKG_CST_CONFIG_CONTROL` MSR register lock, this option should be avoided whenever possible. For several APTIO firmwares not displaying `CFG Lock` setting in the GUI it is possible to access the option directly:

   (a) Download UEFITool and IFR-Extractor.
   (b) Open your firmware image in UEFITool and find `CFG Lock` unicode string. If it is not present, your firmware may not have this option and you should stop.
   (c) Extract the `Setup.bin` PE32 Image Section (the one UEFITool found) through `Extract Body` menu option.
   (d) Run IFR-Extractor on the extracted file (e.g. `./ifrextract Setup.bin Setup.txt`).
   (e) Find `CFG Lock, VarStoreInfo (VarOffset/VarName):` in `Setup.txt` and remember the offset right after it (e.g. `0x123`).
   (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 your actual offset, and reboot.

   ~~**WARNING**~~**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.

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. See `AppleCpuPmCfgLock` description for more details.

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

   This is normally used in conjunction with `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. In general only certain Xeon models benefit from the patch.

5. `CustomSMBIOSGuid`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.6 (64-bit)
   Description: Performs GUID patching for `UpdateSMBIOSMode Custom` mode. Usually relevant for Dell laptops.

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

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

7. `DisableRtcChecksum`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.6 (64-bit)
   Description: Disables primary checksum (`0x58-0x59`) writing in AppleRTC.

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

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

8. `DummyPowerManagement`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.6 (64-bit)
   Description: Disables `AppleIntelCpuPowerManagement`.

   Note: This option is a preferred alternative to `NullCpuPowerManagement.kext` for CPUs without native power management driver in macOS.

9. `ExternalDiskIcons`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.6 (64-bit)
   Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

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

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

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

11. `LapicKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.6 (64-bit)
    Description: Disables kernel panic on LAPIC interrupts.

12. `PanicNoKextDump`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.13 (not required for older)
    Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

13. `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.

14. `ThirdPartyDrives`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.6 (64-bit, not required for older)
    Description: Apply vendor patches to IOAHCIBlockStorage.kext to enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

    Note: This option may be avoided on user preference. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called `trimforce`. Starting from 10.15 this utility creates `EnableTRIM` variable in `APPLE_BOOT_VARIABLE_GUID` namespace with `01 00 00 00` value.

15. `XhciPortLimit`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.11 (not required for older)
    Description: Patch various kexts (AppleUSBXHCI.kext, AppleUSBXHCIPCI.kext, IOUSBHostFamily.kext) to remove USB port count limit of 15 ports.

    Note: This option should be avoided whenever possible. USB port limit is imposed by the amount of used bits in locationID format and there is no possible way to workaround this without heavy OS modification. The only valid solution is to limit the amount of used ports to 15 (discarding some). More details can be found on AppleLife.ru.

7.9 Scheme Properties

These properties are particularly relevant for older macOS operating systems. For more details on how to install and troubleshoot such macOS installation refer to Legacy Apple OS.

1. `FuzzyMatch`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Use `kernelcache` with different checksums when available.

   On macOS 10.6 and earlier `kernelcache` filename has a checksum, which essentially is `adler32` from SMBIOS product name and EfiBoot device path. On certain firmwares EfiBoot device path differs between UEFI and macOS due to ACPI or hardware specifics, rendering `kernelcache` checksum as always different.

   This setting allows matching the latest `kernelcache` with a suitable architecture when the `kernelcache` without suffix is unavailable, improving macOS 10.6 boot performance on several platforms.

2. `KernelArch`
   Type: `plist string`
   Failsafe: `Auto`
   Description: Prefer specified kernel architecture (`Auto`, `i386`, `i386-user32`, `x86_64`) when available.

   On macOS 10.7 and earlier XNU kernel can boot with architectures different from the usual `x86_64`. This setting will use the specified architecture to boot macOS when it is supported by the macOS and the configuration:

   - `Auto` — Choose the preferred architecture automatically.
   - `i386` — Use `i386` (32-bit) kernel when available.
   - `i386-user32` — Use `i386` (32-bit) kernel when available and force the use of 32-bit userspace on 64-bit capable processors. On macOS 64-bit capable processors are assumed to support `SSSE3`. This is not the case for older 64-bit capable Pentium processors, which cause some applications to crash on macOS 10.6. The behaviour corresponds to `-legacy` kernel boot argument.
   - `x86_64` — Use `x86_64` (64-bit) kernel when available.

   Below is the algorithm determining the kernel architecture.

   (a) `arch` argument in image arguments (e.g. when launched via UEFI Shell) or in `boot-args` variable overrides any compatibility checks and forces the specified architecture, completing this algorithm.
   (b) OpenCore build architecture restricts capabilities to `i386` and `i386-user32` mode for the 32-bit firmware variant.
   (c) Determined EfiBoot version restricts architecture choice:
      - 10.4-10.5 — `i386` or `i386-user32`
      - 10.6-10.7 — `i386`, `i386-user32`, or `x86_64`
      - 10.8 or newer — `x86_64`
   (d) If `KernelArch` is set to `Auto` and `SSSE3` is not supported by the CPU, capabilities are restricted to `i386-user32` if supported by EfiBoot.
   (e) Board identifier (from SMBIOS) based on EfiBoot version disables `x86_64` support on an unsupported model if any `i386` variant is supported. `Auto` is not consulted here as the list is not overridable in EfiBoot.
   (f) `KernelArch` restricts the support to the explicitly specified architecture (when not set to `Auto`) if the architecture remains present in the capabilities.
   (g) The best supported architecture is chosen in this order: `x86_64`, `i386`, `i386-user32`.

   Unlike macOS 10.7, where select boards identifiers are treated as the `i386` only machines, and macOS 10.5 or earlier, where `x86_64` is not supported by the macOS kernel, macOS 10.6 is very special. The architecture choice on macOS 10.6 depends on many factors including not only the board identifier, but also macOS product type (client vs server), macOS point release, and RAM amount. The detection of them all is complicated and not practical, because several point releases had genuine bugs and failed to properly perform the server detection in the first place. For this reason OpenCore on macOS 10.6 will fallback to `x86_64` architecture whenever it is supported by the board at all, just like on macOS 10.7. As a reference here is the 64-bit Mac model compatibility corresponding to actual EfiBoot behaviour on macOS 10.6.8 and 10.7.5.

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

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

   Different variants of macOS support different kernel caching variants designed to improve boot performance. This setting allows to prevent using faster kernel caching variants if slower variants are available for debugging and stability reasons. I.e. by specifying `Mkext` one will disable `Prelinked` for e.g. 10.6 but not 10.7.

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

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

- 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 is meant to work reliably only when `RequestBootVarRouting` option is enabled or the firmware does not control UEFI boot options (`OpenDuetPkg` or custom BDS). Without `BootProtect` it also is possible that other operating systems overwrite OpenCore, make sure to enable it if you plan to use them.

Note 2: UEFI variable boot options' boot arguments will be removed if present as they may contain arguments compromising the operating system, which is undesired once secure boot is enabled.

Note 3: Some operating systems, namely Windows, will create their boot option and mark it as top most upon first boot or after NVRAM Reset. When this happens default boot entry choice will update till next manual reconfiguration.

8.2 Properties

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

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

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

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

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

   Designed to be filled with `plist dict` values, describing each load entry. See Entry Properties section below.

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

6. `Tools`
   Type: `plist array`
   Description: Add tool entries to boot picker.

   Designed to be filled with `plist dict` values, describing each load entry. See Entry Properties section below.

   Note: Select tools, for example, UEFI Shell, are very dangerous and **MUST NOT** appear in production configurations, especially in vaulted ones and protected with secure boot, as they may be used to easily bypass secure boot chain. For tool examples check the UEFI section of this document.

8.3 Boot Properties

1. `ConsoleAttributes`
   Type: `plist integer`
   Failsafe: `0`
   Description: Sets specific attributes for console.

   Text renderer supports colour arguments as a sum of foreground and background colours according to UEFI specification. The value of black background and black foreground (`0`) is reserved. List of colour names:

   - `0x00` — `EFI_BLACK`

8. `TakeoffDelay`
   Type: `plist integer`, 32 bit
   Failsafe: `0`
   Description: Delay in microseconds performed before handling picker startup and `action hotkeys`.

   Introducing a delay may give extra time to hold the right `action hotkey` sequence to e.g. boot to recovery mode. On some platforms setting this option to at least `5000-10000` microseconds may be necessary to access `action hotkeys` at all due to the nature of the keyboard driver.

9. `Timeout`
   Type: `plist integer`, 32 bit
   Failsafe: `0`
   Description: Timeout in seconds in boot picker before automatic booting of the default boot entry. Use 0 to disable timer.

10. `PickerMode`
    Type: `plist string`
    Failsafe: `Builtin`
    Description: Choose boot picker used for boot management.

    Picker describes underlying boot management with an optional user interface responsible for handling boot options. The following values are supported:

    - `Builtin` — boot management is handled by OpenCore, a simple text only user interface is used.
    - `External` — an external boot management protocol is used if available. Otherwise `Builtin` mode is used.
    - `Apple` — Apple boot management is used if available. Otherwise `Builtin` mode is used.

    Upon success `External` mode will entirely disable all boot management in OpenCore except policy enforcement. In `Apple` mode it may additionally bypass policy enforcement. See OpenCanopy plugin for an example of a custom user interface.

    OpenCore built-in boot picker contains a set of actions chosen during the boot process. The list of supported actions is similar to Apple BDS and in general can be accessed by holding `action hotkeys` during boot process. Currently the following actions are considered:

    - `Default` — this is the default option, and it lets OpenCore built-in boot picker to loads the default boot option as specified in Startup Disk preference pane.
    - `ShowPicker` — this option forces picker to show. Normally it can be achieved by holding `OPT` key during boot. Setting `ShowPicker` to `true` will make `ShowPicker` the default option.
    - `ResetNvram` — this option performs select UEFI variable erase and is normally achieved by holding `CMD+OPT+P+R` key combination during boot. Another way to erase UEFI variables is to choose `Reset NVRAM` in the picker. This option requires `AllowNvramReset` to be set to `true`.
    - `BootApple` — this options performs booting to the first found Apple operating system unless the default chosen operating system is already made by Apple. Hold `X` key to choose this option.
    - `BootAppleRecovery` — this option performs booting to Apple operating system recovery. Either the one related to the default chosen operating system, or first found in case default chosen operating system is not made by Apple or has no recovery. Hold `CMD+R` key combination to choose this option.

    Note 1: Activated `KeySupport`, `OpenUsbKbDxe`, or similar driver is required for key handling to work. On many firmwares it is not possible to get all the keys function.

    Note 2: In addition to `OPT` OpenCore supports `Escape` key to display picker when `ShowPicker` is disabled. This key exists for `Apple` picker mode and for firmwares with PS/2 keyboards that fail to report held `OPT` key and require continual presses of `Escape` key to enter the boot menu.

    Note 3: On Macs with problematic GOP it may be difficult to access Apple BootPicker. ~~To~~ `BootKicker` utility can be blessed to workaround this problem even without loading OpenCore. On some Macs `BootKicker` ~~utility can be blessed~~will not run from OpenCore.

8.4 Debug Properties

1. `AppleDebug`
   Type: `plist boolean`

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

   This option will create a `SysReport` directory on ESP partition unless it is already present. The directory will contain ACPI and SMBIOS dumps.

   Note: For security reasons `SysReport` option is **not** available in `RELEASE` builds. Use a `DEBUG` build if you need this option.

8. `Target`
   Type: `plist integer`
   Failsafe: `0`
   Description: A bitmask (sum) of enabled logging targets. By default all the logging output is hidden, so this option is required to be set when debugging is necessary.

   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.

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

   Data Hub log will not log kernel and kext patches. To obtain Data Hub log use the following command in macOS:

```
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. For safety reasons log size is limited to 32 kilobytes. Some firmwares may truncate it much earlier or drop completely if they have no memory. Using non-volatile flag will write the log to NVRAM flash after every printed line. To obtain UEFI variable log 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*~~**Warning**: Some firmwares are reported to have broken NVRAM garbage collection. This means that they may not be able to always free space after variable deletion. Do not use non-volatile NVRAM logging without extra need on such devices.

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

   File logging will create a file named `opencore-YYYY-MM-DD-HHMMSS.txt` at 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 firmwares are not reliable, and may corrupt data when writing files through UEFI. Log is attempted to be written in the safest manner, and thus is very slow. Ensure that `DisableWatchDog` is set to `true` when you use a slow drive. Try to avoid frequent use of this option when dealing with flash drives as large I/O amounts may speedup memory wear and render this flash drive unusable in shorter time.

   When interpreting the log, note that the lines are prefixed with a tag describing the relevant location (module) of the log line allowing one to better attribute the line to the functionality. The list of currently used tags is provided below.

   Drivers and tools:

   - `BMF` — OpenCanopy, bitmap font
   - `BS` — Bootstrap

- GSTT — GoptStop
- HDA — AudioDxe
- KKT — KeyTester
- MMDD — MmapDump
- OCPAVP — PavpProvision
- OCRST — ResetSystem
- OCUI — OpenCanopy
- OC — OpenCore main
- VMOPT — VerifyMemOpt

Libraries:

- AAPL — OcDebugLogLib, Apple EfiBoot logging
- OCABC — OcAfterBootCompatLib
- OCAE — OcAppleEventLib
- OCAK — OcAppleKernelLib
- OCAU — OcAudioLib
- OCAV — OcAppleImageVerificationLib
- OCA —- OcAcpiLib
- OCBP — OcAppleBootPolicyLib
- OCB — OcBootManagementLib
- OCCL — OcAppleChunkListLib
- OCCPU — OcCpuLib
- OCC — OcConsoleLib
- OCDH — OcDataHubLib
- OCDI — OcAppleDiskImageLib
- OCFSQ — OcFileLib, UnblockFs quirk
- OCFS — OcFileLib
- OCFV — OcFirmwareVolumeLib
- OCHS — OcHashServicesLib
- OCI4 — OcAppleImg4Lib
- OCIC — OcImageConversionLib
- OCII — OcInputLib
- OCJS — OcApfsLib
- OCKM — OcAppleKeyMapLib
- OCL — OcDebugLogLib
- OCMCO — OcMachoLib
- OCME — OcHeciLib
- OCMM — OcMemoryLib
- 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
- OCXML — OcXmlLib

8.5 Security Properties

1. AllowNvramReset
   Type: plist boolean
   Failsafe: false
   Description: Allow CMD+OPT+P+R handling and enable showing NVRAM Reset entry in boot picker.

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

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

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

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

   Setting this value to any non-zero 64-bit integer will allow using personalised Apple Secure Boot identifiers. If you want to use this setting, make sure to generate a random 64-bit number with a cryptographically secure random number generator. With this value set and `SecureBootModel` valid and not `Disabled` it is possible to achieve `Full Security` of Apple Secure Boot.

   Note 1: You will have to reinstall the operating system or use macOS DMG recovery to `bless -personalize` your installation after setting this value to non-zero. Installing the operating system with `ApECID` value set to non-zero is only possible through macOS recovery or personalized builds created with `asr`.

   Note 2: Currently the use of this option is unrealiable (apparently to a bug in macOS installer), and thus its use is not recommended.

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

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

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

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

   Valid values:

   - `None` — do nothing.
   - `Bootstrap` — create or update top-priority `\EFI\OC\Bootstrap\Bootstrap.efi` boot option (`Boot9696`) in UEFI variable storage at bootloader startup. For this option to work `RequestBootVarRouting` is required to be enabled.

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

   Note 1: Some firmewares may have broken NVRAM, no boot option support, or various other incompatibilities of any kind. While unlikely, the use of this option may even cause boot failure. Use at your own risk on boards known to be compatible.

   Note 2: Be warned that while NVRAM reset executed from OpenCore should not erase the boot option created in `Bootstrap`, executing NVRAM reset prior to loading OpenCore will remove it.

6. `DmgLoading`
   Type: `plist string`

Failsafe: `Signed`
Description: Define Disk Image (DMG) loading policy used for macOS Recovery.

Valid values:

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

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

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

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

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

   To use booter path for mounting 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 OpenCore version use the following command in macOS:

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

   To obtain OEM information use the following commands in macOS:

   ```
   nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:oem-product # SMBIOS Type1 ProductName
   nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:oem-vendor  # SMBIOS Type2 Manufacturer
   nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:oem-board   # SMBIOS Type2 ProductName
   ```

8. `HaltLevel`
   Type: `plist integer`, 64 bit
   Failsafe: `0x80000000` (`DEBUG_ERROR`)
   Description: EDK II debug level bitmask (sum) causing CPU to halt (stop execution) after obtaining a message of `HaltLevel`. Possible values match `DisplayLevel` values.

9. `Vault`
   Type: `plist string`
   Failsafe: `Secure`
   Description: Enables vaulting mechanism in OpenCore.

   Valid values:

   - `Optional` — require nothing, no vault is enforced, insecure.
   - `Basic` — require `vault.plist` file present in `OC` directory. This provides basic filesystem integrity verification and may protect from unintentional filesystem corruption.
   - `Secure` — require `vault.sig` signature file for `vault.plist` in `OC` directory. This includes `Basic` integrity checking but also attempts to build a trusted bootchain.

`vault.plist` file should contain SHA-256 hashes for all files used by OpenCore. Presence of this file is highly recommended to ensure that unintentional file modifications (including filesystem corruption) do not happen unnoticed. To create this file automatically use `create_vault.sh` script. Regardless of the underlying filesystem, path name and case must match between `config.plist` and `vault.plist`.

`vault.sig` file should contain a raw 256 byte RSA-2048 signature from SHA-256 hash of `vault.plist`. The signature is verified against the public key embedded into `OpenCore.efi`. To embed the public key you should do either of the following:

- Provide public key during the `OpenCore.efi` compilation in `OpenCoreVault.c` file.
- Binary patch `OpenCore.efi` replacing zeroes with the public key between `=BEGIN OC VAULT=` and `==END OC VAULT==` ASCII markers.

RSA public key 520 byte format description can be found in Chromium OS documentation. To convert public key from X.509 certificate or from PEM file use RsaTool.

The complete set of commands to:

- Create `vault.plist`.
- Create a new RSA key (always do this to avoid loading old configuration).
- Embed RSA key into `OpenCore.efi`.
- Create `vault.sig`.

Can look as follows:

```
cd /Volumes/EFI/EFI/OC
/path/to/create_vault.sh .
/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, but you have to use an external method to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this you are recommended to at least enable UEFI SecureBoot with a custom certificate, and sign `OpenCore.efi` and `BOOTx64.efi` with your custom key. More details on customising secure boot on modern firmwares can be found in Taming UEFI SecureBoot paper (in Russian).

Note 2: `vault.plist` and `vault.sig` are used regardless of this option when `vault.plist` is present or public key is embedded into `OpenCore.efi`. Setting this option will only ensure configuration sanity, and abort the boot process otherwise.

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

    This value allows to prevent scanning (and booting) from untrusted source based on a bitmask (sum) of select 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 the additional measures are to be applied.

    Third party drivers may introduce additional security (and performance) measures following the provided scan policy. Scan policy is exposed in `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, and to avoid mounting of undesired file systems it is best not to load its driver. 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. This 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. 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_EXT, allows scanning of EXT (Linux Root) file system.
- 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.
- 0x00200000 (bit 21) — OC_SCAN_ALLOW_DEVICE_USB, allow scanning USB devices.
- 0x00400000 (bit 22) — OC_SCAN_ALLOW_DEVICE_FIREWIRE, allow scanning FireWire devices.
- 0x00800000 (bit 23) — OC_SCAN_ALLOW_DEVICE_SDCARD, allow scanning card reader devices.
- 0x01000000 (bit 24) — OC_SCAN_ALLOW_DEVICE_PCI, allow scanning devices directly connected to PCI bus (e.g. VIRTIO).

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

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

11. SecureBootModel
Type: plist string
Failsafe: Default
Description: Apple Secure Boot hardware model.

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

- Default — Recent available model, currently set to j137.
- Disabled — No model, Secure Boot will be disabled.
- j137 — iMacPro1,1 (December 2017) minimum macOS 10.13.2 (17C2111)
- j680 — MacBookPro15,1 (July 2018) minimum macOS 10.13.6 (17G2112)
- j132 — MacBookPro15,2 (July 2018) minimum macOS 10.13.6 (17G2112)
- j174 — Macmini8,1 (October 2018) minimum macOS 10.14 (18A2063)
- j140k — MacBookAir8,1 (October 2018) minimum macOS 10.14.1 (18B2084)
- j780 — MacBookPro15,3 (May 2019) minimum macOS 10.14.5 (18F132)
- j213 — MacBookPro15,4 (July 2019) minimum macOS 10.14.5 (18F2058)
- j140a — MacBookAir8,2 (July 2019) minimum macOS 10.14.5 (18F2058)
- j152f — MacBookPro16,1 (November 2019) minimum macOS 10.15.1 (19B2093)
- j160 — MacPro7,1 (December 2019) minimum macOS 10.15.1 (19B88)
- j230k — MacBookAir9,1 (March 2020) minimum macOS 10.15.3 (19D2064)
- j214k — MacBookPro16,2 (May 2020) minimum macOS 10.15.4 (19E2269)
- j223 — MacBookPro16,3 (May 2020) minimum macOS 10.15.4 (19E2265)
- j215 — MacBookPro16,4 (June 2020) minimum macOS 10.15.5 (19F96)
- j185 — iMac20,1 (August 2020) minimum macOS 10.15.6 (19G2005)
- j185f — iMac20,2 (August 2020) minimum macOS 10.15.6 (19G2005)

PlatformInfo and SecureBootModel are independent, allowing to enabling Apple Secure Boot with any SMBIOS. Setting SecureBootModel to any valid value but Disabled is equivalent to Medium Security of Apple Secure Boot. To achieve Full Security one will need to also specify ApECID value.

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

(a) Just like on T2 Macs you will not be able to install any unsigned kernel drivers and several signed kernel drivers including NVIDIA Web Drivers.
(b) The list of cached drivers may be different, resulting in the need to change the list of `Added` or `Forced` kernel drivers. For example, `IO80211Family` cannot be injected in this case.
(c) If your platform requires certain settings, but they were not enabled, because the obvious issues did not trigger before, you may get boot failure. Be extra careful with `IgnoreInvalidFlexRatio` or `HashServices`.
(d) Operating systems released before Apple Secure Boot landed (e.g. macOS 10.12 or earlier) will still boot until UEFI Secure Boot is enabled. This is so, because from Apple Secure Boot point they are treated as incompatible and are assumed to be handled by the firmware just like Microsoft Windows is.
(e) On older CPUs (e.g. before Sandy Bridge) enabling Apple Secure Boot might cause slightly slower loading by up to 1 second.
(f) Since `Default` value will increase with time to support the latest major release operating system, it is not recommended to use `ApECID` and `Default` value together.

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

8.6 Entry Properties

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

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

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

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

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

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

   - `Entries` specify external boot options, and therefore take device paths in `Path` key. These values are not checked, thus be extremely careful. Example: `PciRoot(0x0)/Pci(0x1,0x1)/.../\EFI\COOL.EFI`
   - `Tools` specify internal boot options, which are part of bootloader vault, and therefore take file paths relative to `OC/Tools` directory. Example: `OpenShell.efi`.

9 NVRAM

9.1 Introduction

Has `plist dict` type and allows to set volatile UEFI variables commonly referred as NVRAM variables. Refer to `man nvram` for more details. macOS extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication, and thus supplying several NVRAM is required for proper macOS functioning.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' NVRAM variable belongs to. macOS uses 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`)
- `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 PlatformNVRAM or Generic subsections of PlatformInfo section. Please ensure that variables of this section never collide with them, as behaviour is undefined otherwise.

For proper macOS functioning it is often required to use `OC_FIRMWARE_RUNTIME` protocol implementation currently offered as a part of `OpenRuntime` driver. While it brings any benefits, there are certain limitations which arise depending on the use.

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 have to be aware of `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 metadata` format. GUIDs must be provided in canonic string format in upper or lower case (e.g. `8BE4DF61-93CA-11D2-AA0D-00E098032B8C`).

   Created variables get `EFI_VARIABLE_BOOTSERVICE_ACCESS` and `EFI_VARIABLE_RUNTIME_ACCESS` attributes set. Variables will only be set if not present or deleted. I.e. to overwrite an existing variable value add the variable name to the `Delete` section. This approach enables to provide default values till the operating system takes the lead.

   Note: If `plist key` does not conform to GUID format, behaviour is undefined.

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 of NVRAM variable file named `nvram.plist` from EFI volume root.

   This file must have 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 `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`. The use of third-party scripts may require `ExposeSensitiveData` set to `0x3` to provide `boot-path` variable with OpenCore EFI partition UUID.

~~WARNING~~Warning: This feature is very dangerous as it passes unprotected data to your firmware variable services. Use it only when no hardware NVRAM implementation is provided by the firmware or it 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 select NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

   You can use `*` value to accept all variables for select GUID.

   WARNING: Choose variables very carefully, as nvram.plist is not vaulted. For instance, do not put `boot-args` or `csr-active-config`, as this can bypass SIP.

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

   Note: This value is recommended to be enabled on most firmwares, but is left configurable for firmwares that may have issues with NVRAM variable storage garbage collection or alike.

To read NVRAM variable value from macOS one could use `nvram` by concatenating variable GUID and name separated by `:` 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*~~Warning: These variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Using `PlatformInfo` is the recommend 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:

- `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
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask`
  Combined `FirmwareFeaturesMask` and `ExtendedFirmwareFeaturesMask`. Present on newer Macs to avoid extra parsing of SMBIOS tables.

10 PlatformInfo

Platform information is comprised of several identification fields generated or filled manually to be compatible with macOS services. The base part of the configuration may be obtained from `AppleModels`, which itself generates a set of interfaces based on a database in YAML format. These fields are written to three select destinations:

- SMBIOS
- Data Hub
- NVRAM

Most of the fields specify the overrides in SMBIOS, and their field names conform to EDK2 SmBios.h header file. However, several important fields reside in Data Hub and NVRAM. Some of the values can be found in more than one field and/or destination, so there are two ways to control their update process: manual, where one specifies all the values (the default), and semi-automatic, where (`Automatic`) only select values are specified, and later used for system configuration.

To inspect SMBIOS contents dmidecode utility can be used. Version with macOS specific enhancements can be downloaded from Acidanthera/dmidecode.

10.1 Properties

1. `Automatic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Generate PlatformInfo based on `Generic` section instead of using values from `DataHub`, `NVRAM`, and `SMBIOS` sections.

   Enabling this option is useful when `Generic` section is flexible enough:

   - When enabled `SMBIOS`, `DataHub`, and `PlatformNVRAM` data is unused.
   - When disabled `Generic` section is unused.

   Warning: It is strongly discouraged set this option to `false` when intending to update platform information. The only reason to do that is when doing minor correction of the SMBIOS present and alike. In all other cases not using `Automatic` may lead to hard to debug errors.

2. `UpdateDataHub`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update Data Hub fields. These fields are read from `Generic` or `DataHub` sections depending on `Automatic` value.

3. `UpdateNVRAM`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update NVRAM fields related to platform information.

   These fields are read from `Generic` or `PlatformNVRAM` sections depending on `Automatic` value. All the other fields are to be specified with `NVRAM` section.

   If `UpdateNVRAM` is set to `false` the aforementioned variables can be updated with `NVRAM` section. If `UpdateNVRAM` is set to `true` the behaviour is undefined when any of the fields are present in `NVRAM` section.

4. `UpdateSMBIOS`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update SMBIOS fields. These fields are read from `Generic` or `SMBIOS` sections depending on `Automatic` value.

5. `UpdateSMBIOSMode`
   Type: `plist string`
   Failsafe: `Create`
   Description: Update SMBIOS fields approach:

AudioDxe*	HDA audio support driver in UEFI firmwares for most Intel and some other analog audio controllers. Staging driver, refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
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 firmwares. For Sandy Bridge and earlier CPUs ExFatDxeLegacy driver should be used due to the lack of RDRAND instruction support.
HfsPlus	Proprietary HFS file system driver with bless support commonly found in Apple firmwares. For Sandy Bridge and earlier CPUs 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 firmwares starting with Ivy Bridge generation. Some applications with the GUI like UEFI Shell may need this driver to work properly.
EnhancedFatDxe	FAT filesystem driver from FatPkg. This driver is embedded in all UEFI firmwares, and cannot be used from OpenCore. It is known that multiple firmwares have a bug in their FAT support implementation, which leads to corrupted filesystems on write attempt. Embedding this driver within the firmware may be required in case writing to EFI partition is needed during the boot process.
NvmExpressDxe*	NVMe support driver from MdeModulePkg. This driver is included in most firmwares starting with 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.
OpenUsbKbDxe*	USB keyboard driver adding the support of 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.
PartitionDxe	Proprietary partition management driver with Apple Partitioning Scheme support commonly found in Apple firmwares. This driver can be used to support loading older DMG recoveries such as macOS 10.9 using Apple Partitioning Scheme. For Sandy Bridge and earlier CPUs PartitionDxeLegacy driver should be used due to the lack of RDRAND instruction support.
Ps2KeyboardDxe*	PS/2 keyboard driver from MdeModulePkg. OpenDuetPkg and some firmwares may not include this driver, but it is necessary for PS/2 keyboard to work. Note, unlike OpenUsbKbDxe this driver has no AppleKeyMapAggregator support and thus requires KeySupport to be enabled.
Ps2MouseDxe*	PS/2 mouse driver from MdeModulePkg. Some very old laptop firmwares may not include this driver, but it is necessary for touchpad to work in UEFI graphical interfaces, such as OpenCanopy.
UsbMouseDxe*	USB mouse driver from MdeModulePkg. Some virtual machine firmwares like OVMF may not include this driver, but it is necessary for mouse to work in UEFI graphical interfaces, such as OpenCanopy.
VBoxHfs	HFS file system driver with bless support. This driver is an alternative to a closed source HfsPlus driver commonly found in Apple firmwares. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit.
XhciDxe*	XHCI USB controller support driver from MdeModulePkg. This driver is included in most firmwares starting with Sandy Bridge generation. For earlier firmwares or legacy systems it may be used to support external USB 3.0 PCI cards.

Driver marked with * are bundled with OpenCore. To compile the drivers from UDK (EDK II) use the same command you normally use for OpenCore compilation, but choose a corresponding package:

```
git clone https://github.com/acidanthera/audk UDK
cd UDK
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p FatPkg/FatPkg.dsc
build -a X64 -b RELEASE -t XCODE5 -p MdeModulePkg/MdeModulePkg.dsc
```

- `Other` — Custom entry (see `Entries`).
- `ResetNVRAM` — Reset NVRAM system action or tool.
- `Shell` — Entry with UEFI Shell name (e.g. `OpenShell`).
- `Tool` — Any other tool.

Predefined labels are put to `\EFI\OC\Resources\Label` directory. Each label has `.lbl` or `.l2x` suffix to represent the scaling level. Full list of labels is provided below. All labels are mandatory.

- `EFIBoot` — Generic OS.
- `Apple` — Apple OS.
- `AppleRecv` — Apple Recovery OS.
- `AppleTM` — Apple Time Machine.
- `Windows` — Windows.
- `Other` — Custom entry (see `Entries`).
- `ResetNVRAM` — Reset NVRAM system action or tool.
- `Shell` — Entry with UEFI Shell name (e.g. `OpenShell`).
- `Tool` — Any other tool.

Label and icon generation can be performed with bundled utilities: `disklabel` and `icnspack`. Please refer to sample data for the details about the dimensions. Font is Helvetica 12 pt times scale factor.

Font format corresponds to AngelCode binary BMF. While there are many utilities to generate font files, currently it is recommended to use dpFontBaker to generate bitmap font (using CoreText produces best results) and fonverter to export it to binary format.

~~*WARNING*: OpenCanopy is currently considered experimental and is not recommended for everyday use. Refer to for more details regarding the current limitations.~~

11.5 OpenRuntime

`OpenRuntime` is an OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol. This protocol implements multiple features required for OpenCore that are otherwise not possible to implement in OpenCore itself as they are needed to 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, like 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 Properties

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

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

   Audio support provides a way for upstream protocols to interact with the selected hardware and audio resources. All audio resources should reside in `\EFI\OC\Resources\Audio` directory. Currently the only supported audio file format is 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.

Currently two renderers are supported: `Builtin` and `System`. `System` renderer uses firmware services for text rendering. `Builtin` bypassing firmware services and performs text rendering on its own. Different renderers support a different set of options. It is recommended to use `Builtin` renderer, as it supports HiDPI mode and uses full screen resolution.

UEFI firmwares generally support `ConsoleControl` with two rendering modes: `Graphics` and `Text`. Some firmwares do not support `ConsoleControl` and rendering modes. OpenCore and macOS expect text to only be shown in `Graphics` mode and graphics to be drawn in any mode. Since this is not required by UEFI specification, exact behaviour varies.

Valid values are combinations of text renderer and rendering mode:

- `BuiltinGraphics` — Switch to `Graphics` mode and use `Builtin` renderer with custom `ConsoleControl`.
- `BuiltinText` — Switch to `Text` mode and use `Builtin` renderer with custom `ConsoleControl`.
- `SystemGraphics` — Switch to `Graphics` mode and use `System` renderer with custom `ConsoleControl`.
- `SystemText` — Switch to `Text` mode and use `System` renderer with custom `ConsoleControl`.
- `SystemGeneric` — Use `System` renderer with system `ConsoleControl` assuming it behaves correctly.

The use of `BuiltinGraphics` is generally straightforward. For most platforms it is necessary to enable `ProvideConsoleGop`, set `Resolution` to `Max`. `BuiltinText` variant is an alternative `BuiltinGraphics` for some very old and buggy laptop firmwares, which can only draw in `Text` mode.

The use of `System` protocols is more complicated. In general the preferred setting is `SystemGraphics` or `SystemText`. Enabling `ProvideConsoleGop`, setting `Resolution` to `Max`, enabling `ReplaceTabWithSpace` is useful on almost all platforms. `SanitiseClearScreen`, `IgnoreTextInGraphics`, and `ClearScreenOnModeSwitch` are more specific, and their use depends on the firmware.

Note: Some Macs, namely `MacPro5,1`, may have broken console output with newer GPUs, and thus only `BuiltinGraphics` may work for them.

2. `ConsoleMode`
   Type: `plist string`
   Failsafe: Empty string
   Description: Sets console output mode as specified with the `WxH` (e.g. `80x24`) formatted string.

   Set to empty string not to change console mode. Set to `Max` to try to use largest available console mode. Currently `Builtin` text renderer supports only one console mode, so this option is ignored.

   Note: This field is best to be left empty on most firmwares.

3. `Resolution`
   Type: `plist string`
   Failsafe: Empty string
   Description: Sets console output screen resolution.

   - Set to `WxH@Bpp` (e.g. `1920x1080@32`) or `WxH` (e.g. `1920x1080`) formatted string to request custom resolution from GOP if available.
   - Set to empty string not to change screen resolution.
   - Set to `Max` to try to use largest available screen resolution.

   On HiDPI screens `APPLE_VENDOR_VARIABLE_GUID UIScale` NVRAM variable may need to be set to `02` to enable HiDPI scaling in `Builtin` text renderer, FileVault 2 UEFI password interface, and boot screen logo. Refer to Recommended Variables section for more details.

   Note: This will fail when console handle has no GOP protocol. When the firmware does not provide it, it can be added with `ProvideConsoleGop` set to `true`.

4. `ClearScreenOnModeSwitch`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Some firmwares clear only part of screen when switching from graphics to text mode, leaving a fragment of previously drawn image visible. This option fills the entire graphics screen with black colour before switching to text mode.

   Note: This option only applies to `System` renderer.

Some firmwares do not implement legacy UGA protocol, but it may be required for screen output by older EFI applications like EfiBoot from 10.4.

11.11 ProtocolOverrides Properties

1. `AppleAudio`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reinstalls Apple audio protocols with builtin versions.

   Apple audio protocols allow macOS bootloader and OpenCore to play sounds and signals for screen reading or audible error reporting. Supported protocols are beep generation and VoiceOver. VoiceOver protocol is specific to Gibraltar machines (T2) and is not supported before macOS High Sierra (10.13). Instead older macOS versions use AppleHDA protocol, which is currently not implemented.

   Only one set of audio protocols can be available at a time, so in order to get audio playback in OpenCore user interface on Mac system implementing some of these protocols this setting should be enabled.

   Note: Backend audio driver needs to be configured in `UEFI Audio` section for these protocols to be able to stream audio.

2. `AppleBootPolicy`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reinstalls Apple Boot Policy protocol with a builtin version. This may be used to ensure APFS compatibility on VMs or legacy Macs.

   Note: Some Macs, namely `MacPro5,1`, do have APFS compatibility, but their Apple Boot Policy protocol contains recovery detection issues, thus using this option is advised on them as well.

3. `AppleDebugLog`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reinstalls Apple Debug Log protocol with a builtin version.

4. `AppleEvent`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reinstalls Apple Event protocol with a builtin version. This may be used to ensure File Vault 2 compatibility on VMs or legacy Macs.

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

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

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

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

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

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

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

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

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

12. `AppleUserInterfaceTheme`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Reinstalls Apple User Interface Theme protocol with a builtin version.

13. `DataHub`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Reinstalls Data Hub protocol with a builtin version. This will delete all previous properties if the protocol was already installed.

14. `DeviceProperties`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Reinstalls Device Property protocol with a builtin version. This will delete all previous properties if it was already installed. This may be used to ensure full compatibility on VMs or legacy Macs.

15. `FirmwareVolume`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Forcibly wraps Firmware Volume protocols or installs new to support custom cursor images for File Vault 2. Should be set to `true` to ensure File Vault 2 compatibility on everything but VMs and legacy Macs.

    Note: Several virtual machines including VMware may have corrupted cursor image in HiDPI mode and thus may also require this setting to be enabled.

16. `HashServices`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Forcibly reinstalls Hash Services protocols with builtin versions. Should be set to `true` to ensure File Vault 2 compatibility on platforms providing broken SHA-1 hashing. Can be diagnosed by invalid cursor size with `UIScale` set to `02`, in general platforms prior to APTIO V (Haswell and older) are affected.

17. `OSInfo`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Forcibly reinstalls OS Info protocol with builtin versions. This protocol is generally used to receive notifications from macOS bootloader, by the firmware or by other applications.

18. `UnicodeCollation`
    Type: `plist boolean`

12 Troubleshooting

12.1 Legacy Apple OS

TBA

12.2 UEFI Secure Boot

TBA

12.3 Windows support

Can I install Windows?

While no official Windows support is provided, 64-bit UEFI Windows installations (Windows 8 and above) prepared with Boot Camp are supposed to work. Third-party UEFI installations as well as systems partially supporting UEFI boot, like Windows 7, might work with some extra precautions. Things to keep in mind:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- All the modifications applied (to ACPI, NVRAM, SMBIOS, etc.) are supposed to be operating system agnostic, i.e. apply equally regardless of the OS booted. This enables Boot Camp software experience on Windows.
- macOS requires the first partition to be EFI System Partition, and does not support the default Windows layout. While OpenCore does have a workaround for this, it is highly recommend not to rely on it and install properly.
- Windows may need to be reactivated. To avoid it consider setting SystemUUID to the original firmware UUID. Be warned, on old firmwares it may be invalid, i.e. not random. In case you still have issues, consider using HWID or KMS38 license or making the use `Custom UpdateSMBIOSMode`. Other nuances of Windows activation are out of the scope of this document and can be found online.

What additional software do I need?

To enable operating system switching and install relevant drivers in the majority of cases you will need Windows support software from Boot Camp. For simplicity of the download process or when configuring an already installed Windows version a third-party utility, Brigadier, can be used successfully. Note, that you may have to download and install 7-Zip prior to using Brigadier.

Remember to always use the latest version of Windows support software from Boot Camp, as versions prior to 6.1 do not support APFS, and thus will not function correctly. To download newest software pass most recent Mac model to Brigadier, for example `./brigadier.exe -m iMac19,1`. To install Boot Camp on an unsupported Mac model afterwards run PowerShell as Administrator and enter `msiexec /i BootCamp.msi`. In case you already have a previous version of Boot Camp installed you will have to remove it first by running `msiexec /x BootCamp.msi` command. `BootCamp.msi` file is located in `BootCamp/Drivers/Apple` directory and can be reached through Windows Explorer.

While Windows support software from Boot Camp solves most of compatibility problems, sometimes you may have to address some of them manually:

- To invert mouse wheel scroll direction `FlipFlopWheel` must be set to `1` as explained on SuperUser.
- `RealTimeIsUniversal` must be set to `1` to avoid time desync between Windows and macOS as explained on SuperUser (this one is usually not needed).
- To access Apple filesystems like HFS and APFS separate software may need to be installed. Some of the known utilities are: Apple HFS+ driver (hack for Windows 10), HFSExplorer, MacDrive, Paragon APFS, Paragon HFS+, TransMac, etc. Remember to never ever attempt to modify Apple file systems from Windows as this often leads to irrecoverable data loss.

Why do I see `Basic data partition` in Boot Camp Startup Disk control panel?

Boot Camp control panel uses GPT partition table to obtain each boot option name. After installing Windows separately you will have to relabel the partition manually. This can be done with many utilities including open-source gdisk utility. Reference example: