

OpenCore

Reference Manual (0.0~~2~~.3)

[2019.06.12]

Type	Value
<code>plist integer</code>	0 (<integer>0</integer>)
<code>plist boolean</code>	False (<false/>)
<code>plist tristate</code>	False (<false/>)

3.3 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.
- `Block` provides support for data removal or ignorance.
- `Patch` provides support for data modification.
- `Quirks` provides support for specific hacks.

Root configuration entries consist of the following:

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

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.

3.4 Directory Structure

When directory boot is used the directory structure used should follow the description on Directory Structure figure. Available entries include:

- `B00Tx64.efi`
Initial booter, which loads `OpenCore.efi` unless it was already started as a driver.
- `ACPI`
Directory used for storing supplemental ACPI information for `ACPI` section.
- `Drivers`
Directory used for storing supplemental UEFI drivers for `UEFI` section.
- `Kexts`
Directory used for storing supplemental kernel information for `Kernel` section.
- [Tools](#)
[Directory used for storing supplemental tools.](#)
- `OpenCore.efi`
Main booter driver responsible for operating system loading.
- `vault.plist`
Hashes for all files potentially loadable by OC Config.
- `config.plist`
OC Config.
- `vault.sig`
Signature for `vault.plist`.
- [nvram.plist](#)
[OpenCore variable import file.](#)
- [opencore.log](#)
[OpenCore log file.](#)

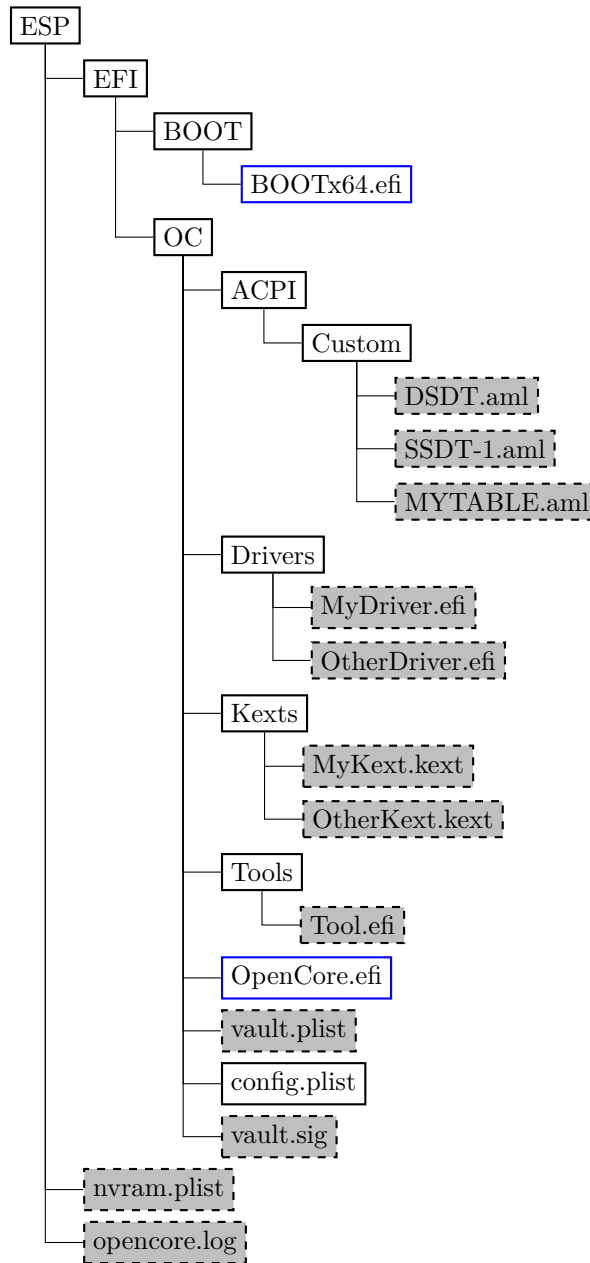


Figure 1. Directory Structure

3.5 Installation and Upgrade

To install OpenCore reflect the Configuration Structure described in the previous section on a EFI volume of a GPT partition. While corresponding sections of this document do provide some information in regards to external resources like ACPI tables, UEFI drivers, or kernel extensions (kexts), completeness of the matter is out of the scope of this document. Extra information about particular kernel extensions may be found in Lilu’s Known Plugins table. Vaulting information is provided in Security Properties section of this document.

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

For BIOS booting a third-party UEFI environment provider will have to be used. DuetPkg is one of the known UEFI environment providers for legacy systems. ~~While it is known to be possible to~~ run OpenCore on such a legacy system ~~; configuration and use of you can install~~ DuetPkg ~~is currently out of the scope of this document~~ with a dedicated tool: BootInstall.

For upgrade purposes refer to `Differences.pdf` document, providing the information about the changes affecting

the configuration compared to the previous release, and `Changelog.md` document, containing the list of modifications across all published updates.

3.6 Contribution

OpenCore can be compiled as an ordinary ~~EDK II package with~~ EDK II. Since UDK development was abandoned by TianoCore, OpenCore requires the use of EDK II Stable. Currently supported EDK II release (potentially with patches enhancing the experience) is hosted in acidanthera/audk.

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.

Required external package dependencies include EfiPkg, MacInfoPkg, and OcSupportPkg.

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 https://github.com/tianocore/edk2 -b UDK2018 UDK
git clone https://github.com/acidanthera/audk UDK
cd UDK
git clone https://github.com/acidanthera/EfiPkg
git clone https://github.com/acidanthera/MacInfoPkg
git clone https://github.com/acidanthera/OcSupportPkg
git clone 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

NOOPT or DEBUG build modes instead of RELEASE can produce a lot more debug output. With NOOPT source level debugging with GDB or IDA Pro is also available. For GDB check OcSupport Debug page. For IDA Pro you will need IDA Pro 7.3 or newer.

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/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AptioFixPkg/Include
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
-I/UefiPackages/OcSupportPkg/Include
-I/UefiPackages/MacInfoPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IIInclude
-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
```

Listing 2: ECC Configuration

Warning: Tool developers modifying `config.plist` or any other OpenCore files must ensure that their tool checks for `opencore-version` NVRAM variable (see Debug Properties section below) and warn the user if the version listed is unsupported or prerelease. OpenCore configuration may change across the releases and the tool shall ensure that it carefully follows this document. Failure to do so may result in this tool to be considered as malware and blocked with all possible means.

Default value: false

Description: Provide reset register and flag in FADT table to enable reboot and shutdown on legacy hardware. Not recommended unless required.

2. ~~IgnoreForWindows~~**Type: plist booleanDefault value: falseDescription:** Disable all sorts of ACPI modifications when booting Windows operating system.—

~~This flag implements a quick workaround for those, who made their ACPI tables incompatible with Windows, but need it right now. Not recommended, as ACPI tables must be compatible with any operating system regardless of the changes.—~~

~~Note: This option may be removed in the future.—~~

3. NormalizeHeaders

Type: plist boolean

Default value: false

Description: Cleanup ACPI header fields to workaround macOS ACPI implementation bug causing boot crashes. Reference: Debugging AppleACPIPlatform on 10.13 by Alex James aka theracermaster. The issue is fixed in macOS Mojave (10.14).

4. RebaseRegions

Type: plist boolean

Default value: false

Description: Attempt to heuristically relocate ACPI memory regions. Not recommended.

ACPI tables are often generated dynamically by underlying firmware implementation. Among the position-independent code, ACPI tables may contain physical addresses of MMIO areas used for device configuration, usually grouped in regions (e.g. `OperationRegion`). Changing firmware settings or hardware configuration, upgrading or patching the firmware inevitably leads to changes in dynamically generated ACPI code, which sometimes lead to the shift of the addresses in aforementioned `OperationRegion` constructions.

For this reason it is very dangerous to apply any kind of modifications to ACPI tables. The most reasonable approach is to make as few as possible changes to ACPI and try to not replace any tables, especially DSDT. When this is not possible, then at least attempt to ensure that custom DSDT is based on the most recent DSDT or remove writes and reads for the affected areas.

When nothing else helps this option could be tried to avoid stalls at `PCI Configuration Begin` phase of macOS booting by attempting to fix the ACPI addresses. It does not do magic, and only works with most common cases. Do not use unless absolutely required.

5. ResetHwSig

Type: plist boolean

Default value: false

Description: Reset FACS table `HardwareSignature` value to 0.

This works around firmwares that fail to maintain hardware signature across the reboots and cause issues with waking from hibernation.

6. ResetLogoStatus

Type: plist boolean

Default value: false

Description: Reset BGRT table `Displayed` status field to false.

This works around firmwares that provide BGRT table but fail to handle screen updates afterwards.

6 Kernel

6.1 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.

6.2 Properties

1. Add

Type: plist array

Default value: 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.

2. Block

Type: plist array

Default value: 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. Patch

Type: plist array

Default value: 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.

5. Quirks

Type: plist dict

Description: Apply individual kernel and driver quirks described in Quirks Properties section below.

6.3 Add Properties

1. BundlePath

Type: plist string

Default value: Empty string

Description: Kext bundle path (e.g. `Lilu.kext` or `MyKext.kext/Contents/PlugIns/MySubKext.kext`).

2. Comment

Type: plist string

Default value: 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

Default value: false

Description: This kernel driver will not be added unless set to `true`.

4. ExecutablePath

Type: plist string

Default value: Empty string

Description: Kext executable path relative to bundle (e.g. Contents/MacOS/Lilu).

5. MatchKernel

Type: plist string

Default value: Empty string

Description: Blocks kernel driver on selected macOS version only. The selection happens based on prefix match with the kernel version, i.e. 16.7.0 will match macOS 10.12.6 and 16. will match any macOS 10.12.x version.

6. PlistPath

Type: plist string

Default value: Empty string

Description: Kext Info.plist path relative to bundle (e.g. Contents/Info.plist).

6.4 Block Properties

1. Comment

Type: plist string

Default value: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. Enabled

Type: plist boolean

Default value: false

Description: This kernel driver will not be blocked unless set to true.

3. Identifier

Type: plist string

Default value: Empty string

Description: Kext bundle identifier (e.g. com.apple.driver.AppleTyMCEDriver).

4. MatchKernel

Type: plist string

Default value: Empty string

Description: Blocks kernel driver on selected macOS version only. The selection happens based on prefix match with the kernel version, i.e. 16.7.0 will match macOS 10.12.6 and 16. will match any macOS 10.12.x version.

6.5 Emulate Properties

1. Cpuid1Data

Type: plist data, 32 bytes

Default value: All zero

Description: Sequence of EAX, EBX, ECX, EDX values in Little Endian order to replace CPUID (1) call in XNU kernel.

2. Cpuid1Mask

Type: plist data, 32 bytes

Default value: All zero

Description: Bit mask of active bits in Cpuid1Data. When each Cpuid1Mask is set to 0, the original CPU bit is used, otherwise .

6.6 Patch Properties

1. Base

Type: plist string

Default value: 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.

2. Comment

Type: plist string

Description: Disables `PKG_CST_CONFIG_CONTROL` (0xE2) MSR modification in `AppleIntelCPUPowerManagement.kext`, commonly causing early kernel panic, when it is locked from writing.

Note: This option should be avoided whenever possible. Modern firmwares provide `CFG Lock` setting, disabling which is much cleaner. More details about the issue can be found in `VerifyMsrE2` notes.

2. `AppleXcpmCfgLock`

Type: `plist boolean`

Default value: `false`

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. Modern firmwares provide `CFG Lock` setting, disabling which is much cleaner. More details about the issue can be found in `VerifyMsrE2` notes.

3. `AppleXcpmExtraMsrs`

Type: `plist boolean`

Default value: `false`

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-X, 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. `CustomSMBIOSGuid`

Type: `plist boolean`

Default value: `false`

Description: Performs GUID patching for `UpdateSMBIOSMode Custom` mode. Usually relevant for Dell laptops.

5. `DisableIoMapper`

Type: `plist boolean`

Default value: `false`

Description: Disables `IoMapper` support in XNU (VT-d), which may conflict with the firmware implementation.

Note: This option is a preferred alternative to dropping `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.

6. `ExternalDiskIcons`

Type: `plist boolean`

Default value: `false`

Description: Apply icon type patches to `IOAHCIPort.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.

7. `LapicKernelPanic`

Type: `plist boolean`

Default value: `false`

Description: Disables kernel panic on AP core lapic interrupt. For BSP core lapic interrupt `lapic_dont_panic=1` kernel boot argument is to be used when `debug` kernel boot argument is present.

8. `PanicNoKextDump`

Type: `plist boolean`

Default value: `false`

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

9. `ThirdPartyTrim`

Type: `plist boolean`

Default value: `false`

Description: Patch `IOAHCIFamily.kext` to force TRIM command support on AHCI SSDs.

7 Misc

7.1 Introduction

This section contains miscellaneous configuration entries for OpenCore behaviour that does not go to any other sections

7.2 Properties

1. **Boot**
Type: plist dict
Description: Apply boot configuration described in Boot Properties section below.
2. **Debug**
Type: plist dict
Description: Apply debug configuration described in Debug Properties section below.
3. **Security**
Type: plist dict
Description: Apply security configuration described in Security Properties section below.

4. **Tools**
Type: plist array
Description: Add new entries to boot picker.

Designed to be filled with **plist dict** values, describing each block entry. See Tools Properties section below.

Note: Select tools, for example, UEFI Shell or NVRAM cleaning 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.

7.3 Boot Properties

1. **ConsoleMode**
Type: plist string
Default value: 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.
2. **ConsoleBehaviourOs**
Type: plist string
Default value: Empty string
Description: Set console control behaviour upon operating system load.

Console control is a legacy protocol used for switching between text and graphics screen output. Some firmwares do not provide it, yet select operating systems require its presence, which is what **ConsoleControl** UEFI protocol is for.

When console control is available, OpenCore can be made console control aware, and and set different modes for the operating system booter (**ConsoleBehaviourOs**), which normally runs in graphics mode, and its own user interface (**ConsoleBehaviourUi**), which normally runs in text mode. Possible behaviours, set as values of these options, include:

- Empty string — Do not modify console control mode.
- **Text** — Switch to text mode.
- **Graphics** — Switch to graphics mode.
- **ForceText** — Switch to text mode and preserve it (requires **ConsoleControl**).
- **ForceGraphics** — Switch to graphics mode and preserve it (require **ConsoleControl**).

Hints:

- Unless empty works, firstly try to set **ConsoleBehaviourOs** to **Graphics** and **ConsoleBehaviourUi** to **Text**.
- On APTIO IV (Haswell and earlier) it is usually enough to have **ConsoleBehaviourOs** set to **Graphics** and **ConsoleBehaviourUi** set to **ForceText** to avoid visual glitches.

- On APTIO V (Broadwell and newer) ConsoleBehaviourOs set to ForceGraphics and ConsoleBehaviourUi set to ~~Text~~[ForceText](#) usually works best.
- [On Apple firmwares ConsoleBehaviourOs set to Graphics and ConsoleBehaviourUi set to Text is supposed to work best.](#)

Note: IgnoreTextInGraphics may need to be enabled for select firmware implementations.

3. ConsoleBehaviourUi

Type: plist string

Default value: Empty string

Description: Set console control behaviour upon OpenCore user interface load. Refer to ConsoleBehaviourOs description for details.

4. [HibernateMode](#)

Type: plist string

Default value: None

Description: [Hibernation detection mode. The following modes are supported:](#)

- [None — Avoid hibernation for your own good.](#)
- [Auto — Use RTC and NVRAM detection.](#)
- [RTC — Use RTC detection.](#)
- [NVRAM — Use NVRAM detection.](#)

5. HideSelf

Type: plist boolean

Default value: false

Description: Hides own boot entry from boot picker. This may potentially hide other entries, for instance, when another UEFI OS is installed on the same volume and driver boot is used.

6. Resolution

Type: plist string

Default value: Empty string

Description: Sets console output screen resolution.

- Set to WxH@Bpp (e.g. 1920x1080@32) 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 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 UEFI quirk set to true.

7. ShowPicker

Type: plist boolean

Default value: false

Description: Show simple boot picker to allow boot entry selection.

8. Timeout

Type: plist integer, 32 bit

Default value: 0

Description: Timeout in seconds in boot picker before automatic booting of the default boot entry.

9. [UsePicker](#)

Type: plist boolean

Default value: false

Description: [Use OpenCore built-in boot picker for boot management.](#)

[UsePicker set to false entirely disables all boot management in OpenCore except policy enforcement. In this case a custom user interface may utilise OcSupportPkg OcBootManagementLib to implement a user friendly boot picker oneself.](#)

Note: By default OpenCore built-in boot picker loads the default discovered option, this can be changed by setting `ShowPicker` to `true`.

7.4 Debug Properties

1. DisableWatchDog

Type: plist boolean

Default value: NO

Description: Select firmwares may not succeed in quickly booting the operating system, especially in debug mode, which results in watch dog timer aborting the process. This option turns off watch dog timer.

2. DisplayDelay

Type: plist integer

Default value: 0

Description: Delay in microseconds performed after every printed line visible onscreen (i.e. console).

3. DisplayLevel

Type: plist integer, 64 bit

Default value: 0

Description: EDK II debug level bitmask (sum) showed onscreen. Unless **Target** enables console (onscreen) printing, onscreen debug output will not be visible. The following levels are supported (discover more in DebugLib.h):

- 0x00000002 (bit 1) — `DEBUG_WARN` in `DEBUG`, `NOOPT`, `RELEASE`.
- 0x00000040 (bit 6) — `DEBUG_INFO` in `DEBUG`, `NOOPT`.
- 0x00400000 (bit 22) — `DEBUG_VERBOSE` in custom builds.
- 0x80000000 (bit 31) — `DEBUG_ERROR` in `DEBUG`, `NOOPT`, `RELEASE`.

4. Target

Type: plist integer

Default value: 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:

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

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.

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, USB, 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

7.6 Tools Properties

1. Comment
Type: `plist_string`
Default value: `Empty string`
Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.
2. Enabled
Type: `plist_boolean`
Default value: `false`
Description: This tool will not be listed unless set to `true`.
3. Name
Type: `plist_string`
Default value: `Empty string`
Description: Human readable tool name displayed in boot picker.
4. Path
Type: `plist_string`
Default value: `Empty string`
Description: File path to select UEFI tool relative to `OC/Tools` directory.

8 NVRAM

8.1 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.

8.2 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 and not blocked. To overwrite a variable add it to `Block` 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. Block

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`

Default value: `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 `Block` (and `Add`) phases, and 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. Example can be found in `Tools`. The use of third-party scripts may require `ExposeSensitiveData` set to `0x3` to provide `boot-path` variable with OpenCore EFI partition UUID.

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. 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.

may be found by looking for the use of `PE_parse_boot_argn` function in the kernel or driver code. Some of the known boot arguments include:

- `acpi_layer=0xFFFFFFFF`
- `acpi_level=0xFFFF5F` (implies `ACPI_ALL_COMPONENTS`)
- `cpus=VALUE`
- `debug=VALUE`
- `io=VALUE`
- `keepsyms=1`
- `kextlog=VALUE`
- `nvda_drv=1`
- `lapic_dont_panic=1`
- `slide=VALUE`
- `-nehalem_error_disable`
- `-no_compat_check`
- `-s`
- `-v`
- `-x`
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg`
Booter arguments, similar to `boot-args` but for `boot.efi`. Accepts a set of arguments, which are hexadecimal 64-bit values with or without `0x` prefix primarily for logging control:
 - `log=VALUE`
 - * 1 — `AppleLoggingConOutOrErrSet/AppleLoggingConOutOrErrPrint` (classical `ConOut/StdErr`)
 - * 2 — `AppleLoggingStdErrSet/AppleLoggingStdErrPrint` (`StdErr` or serial?)
 - * 4 — `AppleLoggingFileSet/AppleLoggingFilePrint` (`BOOTER.LOG/BOOTER.OLD` file on EFI partition)
 - `debug=VALUE`
 - * 1 — enables print something to `BOOTER.LOG` (stripped code implies there may be a crash)
 - * 2 — enables perf logging to `/efi/debug-log` in the device three
 - * 4 — enables timestamp printing for styled `printf` calls
 - `level=VALUE` — Verbosity level of `DEBUG` output. Everything but `0x80000000` is stripped from the binary, and this is the default value.
 - `kc-read-size=VALUE` — Chunk size used for buffered I/O from network or disk for `prelinkedkernel` reading and related. Set to 1MB (`0x100000`) by default, can be tuned for faster booting.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg-once`
Booter arguments override removed after first launch. Otherwise equivalent to `bootercfg`.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:fmm-computer-name`
Current saved host name. ASCII string.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:nvda_drv`
NVIDIA Web Driver control variable. Takes ASCII digit 1 or 0 to enable or disable installed driver.

- **Overwrite** — Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
- **Custom** — Write first SMBIOS table (gEfiSmbiosTableGuid) to gOcCustomSmbiosTableGuid to workaround firmwares overwriting SMBIOS contents at ExitBootServices. Otherwise equivalent to **Create**. Requires patching AppleSmbios.kext and AppleACPIPlatform.kext to read from another GUID: "EB9D2D31" - ➤ "EB9D2D35" (in ASCII), done automatically by CustomSMBIOSGuid quirk.

6. Generic

Type: plist dictionary

Optional: When Automatic is false

Description: Update all fields. This section is read only when Automatic is active.

7. DataHub

Type: plist dictionary

Optional: When Automatic is true

Description: Update Data Hub fields. This section is read only when Automatic is not active.

8. PlatformNVRAM

Type: plist dictionary

Optional: When Automatic is true

Description: Update platform NVRAM fields. This section is read only when Automatic is not active.

9. SMBIOS

Type: plist dictionary

Optional: When Automatic is true

Description: Update SMBIOS fields. This section is read only when Automatic is not active.

9.2 Generic Properties

1. SpoofVendor

Type: plist boolean

Default value: false

Description: Sets SMBIOS vendor fields to Acidanthera.

It is dangerous to use Apple in SMBIOS vendor fields for reasons given in **SystemManufacturer** description. However, certain firmwares may not provide valid values otherwise, which could break some software.

2. SystemProductName

Type: plist string

Default value: MacPro6,1

Description: Refer to SMBIOS SystemProductName.

3. SystemSerialNumber

Type: plist string

Default value: OPENCORE_SN1

Description: Refer to SMBIOS SystemSerialNumber.

4. SystemUUID

Type: plist string, GUID

Default value: OEM specified

Description: Refer to SMBIOS SystemUUID.

5. MLB

Type: plist string

Default value: OPENCORE_MLB_SN11

Description: Refer to SMBIOS BoardSerialNumber.

6. ROM

Type: plist data, 6 bytes

Default value: all zero

Description: Refer to 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM.

11 Troubleshooting

11.1 Tips and Tricks

1. How to debug boot failure?

Normally it is enough to obtain the actual error message. For this ensure that:

- You have a `DEBUG` or `NOOPT` version of OpenCore.
- Logging is enabled (1) and shown onscreen (2): `Misc → Debug → Target = 3`.
- Logged messages from at least `DEBUG_ERROR` (0x80000000), `DEBUG_WARN` (0x00000002), and `DEBUG_INFO` (0x00000040) levels are visible onscreen: `Misc → Debug → DisplayLevel = 0x80000042`.
- Critical error messages, like `DEBUG_ERROR`, stop booting: `Misc → Security → HaltLevel = 0x80000000`.
- Watch Dog is disabled to prevent automatic reboot: `Uefi → Quirks → DisableWatchDog = true`.
- Boot Picker (entry selector) is enabled: `Misc → Boot → ShowPicker = true`.

If there is no obvious error, check the available hacks in `Quirks` sections one by one.

2. How to customise boot entries?

OpenCore follows standard Apple Bless model and extracts the entry name from `.contentDetails` and `.disk_label.contentDetails` files in the booter directory if present. These files contain an ASCII string with an entry title, which may then be customised by the user.

3. What is the simplest way to install macOS?

Copy online recovery image (`*.dmg` and `*.chunklist` files) to `com.apple.recovery.boot` directory on a FAT32 partition with OpenCore. Load OpenCore Boot Picker and choose the entry, it will have a `(dmg)` suffix. Custom name may be created by providing `.contentDetails` file.

To download recovery online you may use Recovery tool from `OcSupportPkg`.

4. 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 in general. Third-party UEFI installations as well as systems partially supporting UEFI boot, like Windows 7, might work with some extra precautions. Be warned that macOS requires first partition to be EFI System Partition, and does not support the default Windows layout. Other than that, 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.

Loading Windows from OpenCore may lead to the need of reactivation. To avoid it consider leaving SystemUUID field empty, so that the original firmware UUID is used. 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, the nuances of Windows activation are out of the scope of this document and can be found online.