



# OpenCore

Reference Manual (0.5.~~7~~.8)

[2020.05.03]

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Generic Terms . . . . .	2
3.2	Installation and Upgrade . . . . .	6
3.3	Contribution . . . . .	6
8.5	Security Properties . . . . .	32
<b>9</b>	<b>NVRAM</b>	<b>36</b>
9.1	Introduction . . . . .	36
9.2	Properties . . . . .	36
10.2	Generic Properties . . . . .	42
<b>11</b>	<b>UEFI</b>	<b>49</b>
11.1	Introduction . . . . .	49
11.2	Drivers . . . . .	49
11.3	Tools . . . . .	51
11.4	OpenCanopy . . . . .	51
11.5	OpenRuntime . . . . .	52
11.6	Properties . . . . .	52
11.7	<a href="#">APFS Properties</a> . . . . .	54
11.8	Audio Properties . . . . .	54
11.11	<del>Protocol</del> <a href="#">ProtocolOverrides</a> Properties . . . . .	59
11.13	<a href="#">ReservedMemory Properties</a> . . . . .	62
<b>12</b>	<b>Troubleshooting</b>	<b>64</b>
12.1	Windows support . . . . .	64

# 1 Introduction

This document provides information on OpenCore user configuration file format used to setup the correct functioning of macOS operating system. It is to be read as the official clarification of expected OpenCore behaviour. All deviations, if found in published OpenCore releases, shall be considered documentation or implementation bugs, and are requested to be reported through Acidanthera Bugtracker. All other sources or translations of this document are unofficial and may contain errors.

This document is structured as a specification, and is not meant to provide a step by step algorithm for configuring end-user board support package (BSP). Any third-party articles, tools, books, etc., providing such material are prone to their authors' preferences, tastes, this document misinterpretation, and essential obsolescence. In case you still use these sources, for example, [OpenCore Desktop Guide](#) ([parent link](#)), please ensure following this document for every made decision and judging its consequences. Regardless of the sources used you are required to fully understand every dedicated OpenCore configuration option and concept prior to reporting any issues in Acidanthera Bugtracker.

## 1.1 Generic Terms

- **plist** — Subset of ASCII Property List format written in XML, also know as XML plist format version 1. Uniform Type Identifier (UTI): `com.apple.property-list`. Plists consist of **plist objects**, which are combined to form a hierarchical structure. Due to plist format not being well-defined, all the definitions of this document may only be applied after plist is considered valid by running `plutil -lint`. External references: <https://www.apple.com/DTDs/PropertyList-1.0.dtd>, `man plutil`.
- **plist type** — plist collections (**plist array**, **plist dictionary**, **plist key**) and primitives (**plist string**, **plist data**, **plist date**, **plist boolean**, **plist integer**, **plist real**).
- **plist object** — definite realisation of **plist type**, which may be interpreted as value.
- **plist array** — array-like collection, conforms to `array`. Consists of zero or more **plist objects**.
- **plist dictionary** — map-like (associative array) collection, conforms to `dict`. Consists of zero or more **plist keys**.
- **plist key** — contains one **plist object** going by the name of **plist key**, conforms to `key`. Consists of printable 7-bit ASCII characters.
- **plist string** — printable 7-bit ASCII string, conforms to `string`.
- **plist data** — base64-encoded blob, conforms to `data`.
- **plist date** — ISO-8601 date, conforms to `date`, unsupported.
- **plist boolean** — logical state object, which is either true (1) or false (0), conforms to `true` and `false`.
- **plist integer** — possibly signed integer number in base 10, conforms to `integer`. Fits in 64-bit unsigned integer in two's complement representation, unless a smaller signed or unsigned integral type is explicitly mentioned in specific **plist object** description.
- **plist real** — floating point number, conforms to `real`, unsupported.
- **plist metadata** — value cast to data by the implementation. Permits passing **plist string**, in which case the result is represented by a null-terminated sequence of bytes (aka C string), **plist integer**, in which case the result is represented by *32-bit* little endian sequence of bytes in two's complement representation, **plist boolean**, in which case the value is one byte: 01 for `true` and 00 for `false`, and **plist data** itself. All other types or larger integers invoke undefined behaviour.

- **Resources**  
Directory used for storing media resources, such as audio files for screen reader support. See [UEFI Audio Properties](#) section for more details.
- **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-YYYY-MM-DD-HHMMSS.txt**  
OpenCore log file.

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

## 3.2 Installation and Upgrade

To install OpenCore reflect the Configuration Structure described in the previous section on a EFI volume of a GPT partition. While corresponding sections of this document do provide some information 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. Information about kernel extensions may be found in a separate Kext List document available in OpenCore repository. Vaulting information is provided in Security Properties section of this document.

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

For BIOS booting a third-party UEFI environment provider will have to be used. **DuetPkg** is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system you can install **DuetPkg** with a dedicated tool [↗BootInstall \(bundled with OpenCore\)](#).

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.

## 3.3 Contribution

OpenCore can be compiled as an ordinary 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 and MacInfoPkg.

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/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/OpenCorePkg
source edksetup.sh
make -C BaseTools
```

*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.

7. [DisableRtcChecksum](#)

**Type:** `plist boolean`

**Failsafe:** `false`

**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`

**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`

**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`

**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`

**Description:** Disables kernel panic on LAPIC interrupts.

12. `PanicNoKextDump`

**Type:** `plist boolean`

**Failsafe:** `false`

**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`

**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`

**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.

- `.VolumeIcon.icns` file at volume root for other filesystems.
- `<TOOL_NAME>.icns` file for Tools.

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

- `0x0002` — `OC_ATTR_USE_DISK_LABEL_FILE`, provides custom rendered titles for boot entries:
  - `.disk_label` (`.disk_label_2x`) file near bootloader for all filesystems.
  - `<TOOL_NAME.1b1` (`<TOOL_NAME.12x`) file near tool for Tools.

Prerendered labels can be generated via `disklabel` utility or `bless` command. When disabled or missing text labels (`.contentDetails` or `.disk_label.contentDetails`) are to be rendered instead.

- `0x0004` — `OC_ATTR_USE_GENERIC_LABEL_IMAGE`, provides predefined label images for boot entries without custom entries. May give less detail for the actual boot entry.
- `0x0008` — `OC_ATTR_USE_ALTERNATE_ICONS`, changes used icon set to an alternate one if it is supported. For example, this could make a use of old-style icons with a custom background colour.

## 6. PickerAudioAssist

**Type:** plist boolean

**Failsafe:** false

**Description:** Enable screen reader by default in boot picker.

For macOS bootloader screen reader preference is set in `preferences.efi` archive in `isV0Enabled.int32` file and is controlled by the operating system. For OpenCore screen reader support this option is an independent equivalent. Toggling screen reader support in both OpenCore boot picker and macOS bootloader FileVault 2 login window can also be done with `Command + F5` key combination.

*Note:* screen reader requires working audio support, see [UEFI Audio Properties](#) section for more details.

## 7. PollAppleHotKeys

**Type:** plist boolean

**Failsafe:** false

**Description:** Enable modifier hotkey handling in boot picker.

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

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

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

## 8. ShowPicker

**Type:** plist boolean

**Failsafe:** false

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

## 9. 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.

## 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.
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. `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 fdsetup 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.

4. `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 bootstrapping OpenCore.

*Note 1:* Some firmwares 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 NVRAM reset will also erase the boot option created in Bootstrap mode.

5. `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
```

---

## 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.
2. Assigned NVRAM variables are not always allowed to exceed 512 bytes.  
This is true for `Boot`-prefixed variables when `RequestBootVarFallback` is used, and for overwriting volatile variables with non-volatile on UEFI 2.8 non-conformant firmwares.

### 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 ~~and not blocked. To overwrite a variable add it to or blocked. I.e. to overwrite an existing variable value add the variable name to the~~ `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`

**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 `Block` (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



- **Overwrite** — Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
- **Custom** — Write ~~first SMBIOS table~~ (SMBIOS tables (~~gEfiSmbiosTableGuid~~gEfiSmbios(3)TableGuid) to ~~gOeCustomSmbiosTableGuid~~gOeCustomSmbios(3)TableGuid 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.

*Note: A side effect of using Custom approach is making SMBIOS updates exclusive to macOS, avoiding a collision with existing Windows activation and custom OEM software but potentially breaking Apple-specific tools.*

#### 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.

## 10.2 Generic Properties

#### 1. SpoofVendor

**Type:** plist boolean

**Failsafe:** 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. AdviseWindows

**Type:** plist boolean

**Failsafe:** false

**Description:** Forces Windows support in **FirmwareFeatures**.

Added bits to **FirmwareFeatures**:

- FW\_FEATURE\_SUPPORTS\_CSM\_LEGACY\_MODE (0x1) - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being not the first partition on the disk.
- FW\_FEATURE\_SUPPORTS\_UEFI\_WINDOWS\_BOOT (0x20000000) - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being the first partition on the disk.

#### 3. SystemProductName

**Type:** plist string

**Failsafe:** MacPro6,1

**Description:** Refer to SMBIOS SystemProductName.

#### 4. SystemSerialNumber

**Type:** plist string

**Failsafe:** OPENCORE\_SN1

**Description:** Refer to SMBIOS SystemSerialNumber.

## **11 UEFI**

### **11.1 Introduction**

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

### **11.2 Drivers**

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

## 11.3 Tools

Standalone tools may help to debug firmware and hardware. Some of the known tools are listed below. While some tools can be launched from within OpenCore many should be run separately either directly or from `Shell`.

To boot into OpenShell or any other tool directly save `OpenShell.efi` under the name of `EFI\BOOT\BOOTX64.EFI` on a FAT32 partition. In general it is unimportant whether the partition scheme is GPT or MBR.

While the previous approach works both on Macs and other computers, an alternative Mac-only approach to bless the tool on an HFS+ or APFS volume:

---

```
sudo bless --verbose --file /Volumes/VOLNAME/DIR/OpenShell.efi \
--folder /Volumes/VOLNAME/DIR/ --setBoot
```

---

Listing 3: Blessing tool

*Note 1:* You may have to copy `/System/Library/CoreServices/BridgeVersion.bin` to `/Volumes/VOLNAME/DIR`.

*Note 2:* To be able to use `bless` you may have to disable System Integrity Protection.

*Note 3:* To be able to boot you may have to disable Secure Boot if present.

Some of the known tools are listed below (builtin tools are marked with \*):

<code>BootKicker*</code>	Enter Apple BootPicker menu (exclusive for Macs with compatible GPUs).
<code>ChipTune*</code>	Test BeepGen protocol and generate audio signals of different style and length.
<code>CleanNvram*</code>	Reset NVRAM alternative bundled as a standalone tool.
<code>GopStop*</code>	Test GraphicsOutput protocol with a simple scenario.
<code>HdaCodecDump*</code>	Parse and dump High Definition Audio codec information (requires <code>AudioDxe</code> ).
<code>KeyTester*</code>	Test keyboard input in <code>SimpleText</code> mode.
<code>MemTest86</code>	Memory testing utility.
<code>OpenControl*</code>	Unlock and lock back NVRAM protection for other tools to be able to get full NVRAM access when launching from OpenCore.
<code>OpenShell*</code>	OpenCore-configured UEFI <code>Shell</code> for compatibility with a broad range of firmwares.
<code>PavpProvision</code>	Perform EPID provisioning (requires certificate data configuration).
<code>ResetSystem*</code>	<a href="#"><u>Utility to perform system reset. Takes reset type as an argument: ColdReset, WarmReset, Shutdown. Defaults to ColdReset.</u></a>
<code>RtcRw*</code>	<a href="#"><u>Utility to read and write RTC (CMOS) memory.</u></a>
<code>VerifyMsrE2*</code>	Check CFG Lock (MSR 0xE2 write protection) consistency across all cores.

## 11.4 OpenCanopy

OpenCanopy is a graphical OpenCore user interface that runs in `External PickerMode` and relies on `OpenCorePkg` `OcBootManagementLib` similar to the builtin text interface.

OpenCanopy requires graphical resources located in `Resources` directory to run. Sample resources (fonts and images) can be found in `OcBinaryData` repository.

OpenCanopy provides full support for `PickerAttributes` and offers a configurable builtin icon set. The default chosen icon set depends on the `DefaultBackgroundColor` variable value. For `Light Gray Old` icon set will be used, for other colours — the one without a prefix.

Predefined icons are put to `\EFI\OC\Resources\Image` directory. Full list of supported icons (in `.icns` format) is provided below. Missing optional icons will use the closest available icon. External entries will use `Ext`-prefixed icon if available (e.g. `OldExtHardDrive.icns`).

- `Cursor` — Mouse cursor (mandatory).
- `Selected` — Selected item (mandatory).
- `Selector` — Selecting item (mandatory).
- `HardDrive` — Generic OS (mandatory).
- `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.

Predefined labels are put to `\EFI\OC\Resources\Label` directory. Each label has `.1b1` or `.12x` suffix to represent the scaling level. Full list of labels is provided below. All labels are mandatory.

- **EFIBoot** — Generic OS.
- **Apple** — Apple OS.
- **AppleRecovery** — 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.

**WARNING:** OpenCanopy is currently considered experimental and is not recommended for everyday use. Refer to [acidanthera/bugtracker#759](#) 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**).
- NVRAM proxying, allowing to manipulate multiple variables on variable updates (e.g. **RequestBootVarFallback**).
- 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.

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

- Audio type can be `OCEFIAudio` for OpenCore audio files or `AXEFIAudio` for macOS bootloader audio files.
- Audio localisation is a two letter language code (e.g. `en`) with an exception for Chinese, Spanish, and Portuguese. Refer to `APPLE_VOICE_OVER_LANGUAGE_CODE` definition for the list of all supported localisations.
- Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to `APPLE_VOICE_OVER_AUDIO_FILE` definition. For OpenCore audio paths refer to `OC_VOICE_OVER_AUDIO_FILE` definition. The only exception is OpenCore boot chime file, which is `OCEFIAudio_VoiceOver_Boot.wav`.

Audio localisation is determined separately for macOS bootloader and OpenCore. For macOS bootloader it is set in `preferences.efi` archive in `systemLanguage.utf8` file and is controlled by the operating system. For OpenCore the value of `prev-lang:kbd` variable is used. When native audio localisation of a particular file is missing, English language (`en`) localisation is used. Sample audio files can be found in OcBinaryData repository.

### 3. ConnectDrivers

**Type:** plist boolean

**Failsafe:** false

**Description:** Perform UEFI controller connection after driver loading.

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

*Note:* Some firmwares, made by Apple in particular, only connect the boot drive to speedup the boot process. Enable this option to be able to see all the boot options when having multiple drives.

### 4. Drivers

**Type:** plist array

**Failsafe:** None

**Description:** Load selected drivers from `OC/Drivers` directory.

Designed to be filled with string filenames meant to be loaded as UEFI drivers.

### 5. Input

**Type:** plist dict

**Failsafe:** None

**Description:** Apply individual settings designed for input (keyboard and mouse) in Input Properties section below.

### 6. Output

**Type:** plist dict

**Failsafe:** None

**Description:** Apply individual settings designed for output (text and graphics) in Output Properties section below.

### 7. ~~Protocols~~ProtocolOverrides

**Type:** plist dict

**Failsafe:** None

**Description:** Force builtin versions of select protocols described in ProtocolOverrides Properties section below.

*Note:* all protocol instances are installed prior to driver loading.

### 8. Quirks

**Type:** plist dict

**Failsafe:** None

**Description:** Apply individual firmware quirks described in Quirks Properties section below.

### 9. ReservedMemory

**Type:** plist array

**Description:** Designed to be filled with plist dict values, describing memory areas exquisite to particular firmware and hardware functioning, which should not be used by the operating system. An example of such memory region could be second 256 MB corrupted by Intel HD 3000 or an area with faulty RAM. See Reserved-Memory Properties section below.

## 11.7 APFS Properties

### 1. EnableJumpstart

**Type:** plist boolean

**Failsafe:** false

**Description:** Load embedded APFS drivers from APFS containers.

APFS EFI driver is bundled in all bootable APFS containers. This option performs loading of signed APFS drivers with respect to **ScanPolicy**. See more details in “EFI Jumpstart” section of Apple File System Reference.

### 2. HideVerbose

**Type:** plist boolean

**Failsafe:** false

**Description:** Hide verbose output from APFS driver.

APFS verbose output can be useful for debugging.

### 3. JumpstartHotPlug

**Type:** plist boolean

**Failsafe:** false

**Description:** Load APFS drivers for newly connected devices.

Performs APFS driver loading not only at OpenCore startup but also during boot picker. This permits APFS USB hot plug. Disable if not required.

### 4. MinDate

**Type:** plist integer

**Failsafe:** 0

**Description:** Minimal allowed APFS driver date.

APFS driver date connects APFS driver with the calendar release date. Older versions of APFS drivers may contain unpatched vulnerabilities, which can be used to inflict harm on your computer. This option permits restricting APFS drivers to only recent releases.

- 0 — require the default supported release date of APFS in OpenCore. The default release date will increase with time and thus this setting is recommended. Currently set to 2018/06/21.
- -1 — permit any release date to load (strongly discouraged).
- Other — use custom minimal APFS release date, e.g. 20200401 for 2020/04/01. APFS release dates can be found in OpenCore boot log and **OcApfsLib**.

### 5. MinVersion

**Type:** plist integer

**Failsafe:** 0

**Description:** Minimal allowed APFS driver version.

APFS driver version connects APFS driver with the macOS release. APFS drivers from older macOS releases will become unsupported and thus may contain unpatched vulnerabilities, which can be used to inflict harm on your computer. This option permits restricting APFS drivers to only modern macOS versions.

- 0 — require the default supported version of APFS in OpenCore. The default version will increase with time and thus this setting is recommended. Currently set to the latest point release from High Sierra from App Store (748077008000000).
- -1 — permit any version to load (strongly discouraged).
- Other — use custom minimal APFS version, e.g. 1412101001000000 from macOS Catalina 10.15.4. APFS versions can be found in OpenCore boot log and **OcApfsLib**.

## 11.8 Audio Properties

### 1. AudioCodec

**Type:** plist integer

**Failsafe:** 0

**Description:** Codec address on the specified audio controller for audio support.

**Failsafe:** false

**Description:** Use builtin graphics output protocol renderer for console.

On some firmwares this may provide better performance or even fix rendering issues, like on **MacPro5,1**. However, it is recommended not to use this option unless there is an obvious benefit as it may even result in slower scrolling.

#### 7. IgnoreTextInGraphics

**Type:** plist boolean

**Failsafe:** false

**Description:** Select firmwares output text onscreen in both graphics and text mode. This is normally unexpected, because random text may appear over graphical images and cause UI corruption. Setting this option to **true** will discard all text output when console control is in mode different from **Text**.

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

#### 8. ReplaceTabWithSpace

**Type:** plist boolean

**Failsafe:** false

**Description:** Some firmwares do not print tab characters or even everything that follows them, causing difficulties or inability to use the UEFI Shell builtin text editor to edit property lists and other documents. This option makes the console output spaces instead of tabs.

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

#### 9. ProvideConsoleGop

**Type:** plist boolean

**Failsafe:** false

**Description:** Ensure GOP (Graphics Output Protocol) on console handle.

macOS bootloader requires GOP to be present on console handle, yet the exact location of GOP is not covered by the UEFI specification. This option will ensure GOP is installed on console handle if it is present.

*Note:* This option will also replace broken GOP protocol on console handle, which may be the case on **MacPro5,1** with newer GPUs.

#### 10. ReconnectOnResChange

**Type:** plist boolean

**Failsafe:** false

**Description:** Reconnect console controllers after changing screen resolution.

On some firmwares when screen resolution is changed via GOP, it is required to reconnect the controllers, which produce the console protocols (simple text out). Otherwise they will not produce text based on the new resolution.

*Note:* On several boards this logic may result in black screen when launching OpenCore from Shell and thus it is optional. In versions prior to 0.5.2 this option was mandatory and not configurable. Please do not use this unless required.

#### 11. SanitiseClearScreen

**Type:** plist boolean

**Failsafe:** false

**Description:** Some firmwares reset screen resolution to a failsafe value (like 1024x768) on the attempts to clear screen contents when large display (e.g. 2K or 4K) is used. This option attempts to apply a workaround.

*Note:* This option only applies to **System** renderer. On all known affected systems **ConsoleMode** had to be set to empty string for this to work.

### 11.11 ~~Protocols~~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. AppleImageConversion

**Type:** plist boolean

**Failsafe:** false

**Description:** Reinstalls Apple Image Conversion protocol with a builtin version.

6. AppleKeyMap

**Type:** plist boolean

**Failsafe:** false

**Description:** Reinstalls Apple Key Map protocols with builtin versions.

7. [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.](#)

8. 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.

9. AppleUserInterfaceTheme

**Type:** plist boolean

**Failsafe:** false

**Description:** Reinstalls Apple User Interface Theme protocol with a builtin version.

10. DataHub

**Type:** plist boolean

**Failsafe:** false



manage to properly do that, or at least have an option for, select firmwares do not. As a result, operating system may freeze upon boot. Not recommended unless required.

#### 4. RequestBootVarFallback

**Type:** plist boolean

**Failsafe:** false

**Description:** Request fallback of some Boot prefixed variables from OC\_VENDOR\_VARIABLE\_GUID to EFI\_GLOBAL\_VARIABLE\_GUID.

This quirk requires RequestBootVarRouting to be enabled and therefore OC\_FIRMWARE\_RUNTIME protocol implemented in OpenRuntime.efi.

By redirecting Boot prefixed variables to a separate GUID namespace we achieve multiple goals:

- Operating systems are jailed and only controlled by OpenCore boot environment to enhance security.
- Operating systems do not mess with OpenCore boot priority, and guarantee fluent updates and hibernation wakes for cases that require reboots with OpenCore in the middle.
- Potentially incompatible boot entries, such as macOS entries, are not deleted or anyhow corrupted.

However, some firmwares do their own boot option scanning upon startup by checking file presence on the available disks. Quite often this scanning includes non-standard locations, such as Windows Bootloader paths. Normally it is not an issue, but some firmwares, ASUS firmwares on APTIO V in particular, have bugs. For them scanning is implemented improperly, and firmware preferences may get accidentally corrupted due to BootOrder entry duplication (each option will be added twice) making it impossible to boot without cleaning NVRAM.

To trigger the bug one should have some valid boot options (e.g. OpenCore) and then install Windows with RequestBootVarRouting enabled. As Windows bootloader option will not be created by Windows installer, the firmware will attempt to create it itself, and then corrupt its boot option list.

This quirk forwards all UEFI specification valid boot options, that are not related to macOS, to the firmware into BootF### and BootOrder variables upon write. As the entries are added to the end of BootOrder, this does not break boot priority, but ensures that the firmware does not try to append a new option on its own after Windows installation for instance.

#### 5. RequestBootVarRouting

**Type:** plist boolean

**Failsafe:** false

**Description:** Request redirect of all Boot prefixed variables from EFI\_GLOBAL\_VARIABLE\_GUID to OC\_VENDOR\_VARIABLE\_GUID.

This quirk requires OC\_FIRMWARE\_RUNTIME protocol implemented in OpenRuntime.efi. The quirk lets default boot entry preservation at times when firmwares delete incompatible boot entries. Simply said, you are required to enable this quirk to be able to reliably use Startup Disk preference pane in a firmware that is not compatible with macOS boot entries by design.

#### 6. UnblockFsConnect

**Type:** plist boolean

**Failsafe:** false

**Description:** Some firmwares block partition handles by opening them in By Driver mode, which results in File System protocols being unable to install.

*Note:* The quirk is mostly relevant for select HP laptops with no drives listed.

## 11.13 ReservedMemory Properties

#### 1. Address

**Type:** plist integer

**Failsafe:** 0

**Description:** Start address of the reserved memory region, which should be allocated as reserved effectively marking the memory of this type inaccessible to the operating system.

The addresses written here must be part of the memory map, have EfiConventionalMemory type, and page-aligned (4 KBs).

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. Size  
Type: plist integer  
Failsafe: 0  
Description: Size of the reserved memory region, must be page-aligned (4 KBs).
4. Enabled  
Type: plist boolean  
Failsafe: false  
Description: This region will not be reserved unless set to **true**.

## 12 Troubleshooting

### 12.1 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.
- To install Windows, macOS, and OpenCore on the same drive you can specify Windows bootloader path (`\EFI\Microsoft\Boot\bootmgfw.efi`) in `BlessOverride` section.
- 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 tools 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 tools including open-source `gdisk` utility. Reference example:

---

```
PS C:\gdisk> .\gdisk64.exe \\.\\physicaldrive0
GPT fdisk (gdisk) version 1.0.4

Command (? for help): p
Disk \\.\\physicaldrive0: 419430400 sectors, 200.0 GiB
Sector size (logical): 512 bytes
Disk identifier (GUID): DEC57EB1-B3B5-49B2-95F5-3B8C4D3E4E12
```