# OpenCore

Reference Manual (0.5~~.5~~.6)

[2020.03.01]

# 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 Vanilla 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.

## 3 Setup

### 3.1 Directory Structure

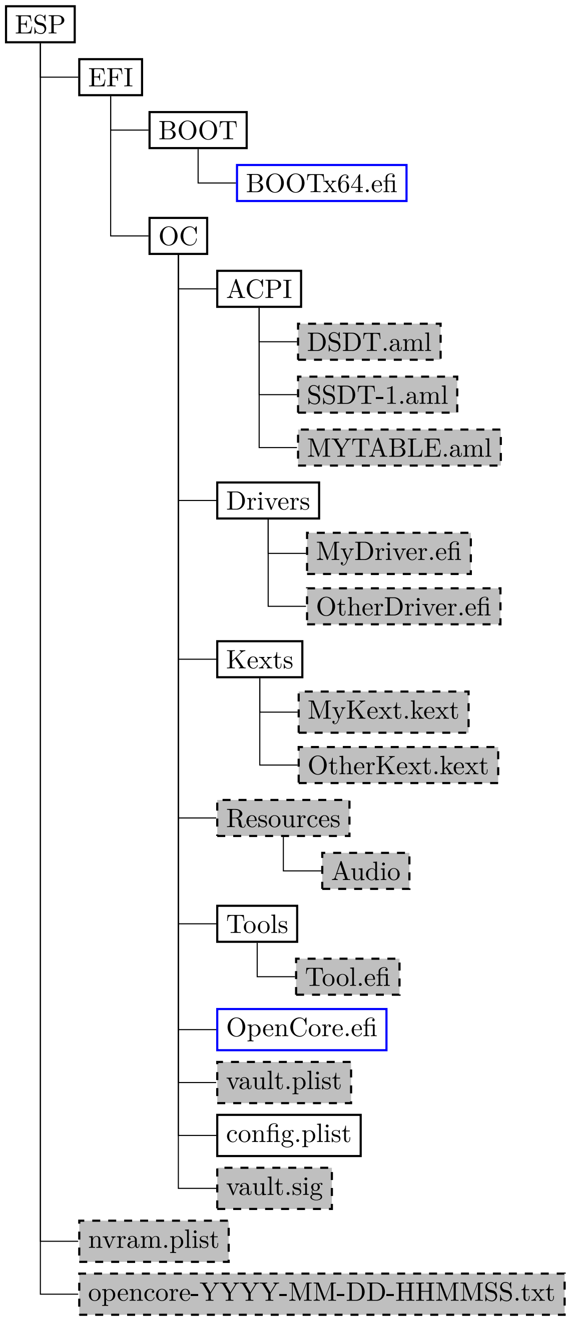

Figure 1. Directory Structure

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

- `BOOTx64.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.

- 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.

## 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.

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 ~~, , and~~ 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/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

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/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
~~-I/UefiPackages/OcSupportPkg/Include~~
-I/UefiPackages/MacInfoPkg/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
```

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.

## 3.4 Coding conventions

Just like any other project we have conventions that we follow during the development. All third-party contributors are highly recommended to read and follow the conventions listed below before submitting their patches. In general it is also recommended to firstly discuss the issue in Acidanthera Bugtracker before sending the patch to ensure no double work and to avoid your patch being rejected.

**Organisation**. The codebase is structured in multiple repositories which contain separate EDK II packages. `AppleSupportPkg` and `OpenCorePkg` are primary packages, and `EfiPkg`, ~~`OcSupportPkg`,~~ `MacInfoPkg.dsc`) are dependent packages.

- Whenever changes are required in multiple repositories, separate pull requests should be sent to each.
- Committing the changes should happen firstly to dependent repositories, secondly to primary repositories to avoid automatic build errors.
- Each unique commit should compile with `XCODE5` and preferably with other toolchains. In the majority of the cases it can be checked by accessing the CI interface. Ensuring that static analysis finds no warnings is preferred.

# 5 Booter

## 5.1 Introduction

This section allows to apply different kinds of UEFI modifications on Apple bootloader (`boot.efi`). The modifications currently provide various patches and environment alterations for different firmwares. Some of these features were originally implemented as a part of AptioMemoryFix.efi, which is no longer maintained. See Tips and Tricks section for migration steps.

If you are using this for the first time on a customised firmware, there is a list of checks to do first. Prior to starting please ensure that you have:

- Most up-to-date UEFI firmware (check your motherboard vendor website).
- `Fast Boot` and `Hardware Fast Boot` disabled in firmware settings if present.
- `Above 4G Decoding` or similar enabled in firmware settings if present. Note, that on some motherboards (notably ASUS WS-X299-PRO) this option causes adverse effects, and must be disabled. While no other motherboards with the same issue are known, consider this option to be first to check if you have erratic boot failures.
- `DisableIoMapper` quirk enabled, or `VT-d` disabled in firmware settings if present, or ACPI DMAR table dropped.
- **No** ‘slide‘ boot argument present in NVRAM or anywhere else. It is not necessary unless you cannot boot at all or see `No slide values are usable! Use custom slide!` message in the log.
- `CFG Lock` (MSR `0xE2` write protection) disabled in firmware settings if present. Cconsider patching it if you have enough skills and no option is available. See ~~nots~~ VerifyMsrE2 notes for more details.
- `CSM` (Compatibility Support Module) disabled in firmware settings if present. You may need to flash GOP ROM on NVIDIA 6xx/AMD 2xx or older. Use GopUpdate (see the second post) or AMD UEFI GOP MAKER in case you are not sure how.
- `EHCI/XHCI Hand-off` enabled in firmware settings `only` if boot stalls unless USB devices are disconnected.
- `VT-x`, `Hyper Threading`, `Execute Disable Bit` enabled in firmware settings if present.
- While it may not be required, sometimes you have to disable `Thunderbolt support`, `Intel SGX`, and `Intel Platform Trust` in firmware settings present.

When debugging sleep issues you may want to (temporarily) disable Power Nap and automatic power off, which appear to sometimes cause wake to black screen or boot loop issues on older platforms. The particular issues may vary, but in general you should check ACPI tables first. Here is an example of a bug found in some Z68 motherboards. To turn Power Nap and the others off run the following commands in Terminal:

```
sudo pmset autopoweroff 0
sudo pmset powernap 0
sudo pmset standby 0
```

*Note*: These settings may reset at hardware change and in certain other circumstances. To view their current state use `pmset -g` command in Terminal.

## 5.2 Properties

1. `MmioWhitelist`
   **Type**: `plist array`
   **Description**: Designed to be filled with `plist dict` values, describing addresses critical for particular firmware functioning when `DevirtualiseMmio` quirk is in use. See MmioWhitelist Properties section below.

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

## 5.3 MmioWhitelist Properties

1. `Address`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Exceptional MMIO address, which memory descriptor should be left virtualised (unchanged) by

5. `DiscardHibernateMap`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Reuse original hibernate memory map.

   This option forces XNU kernel to ignore newly supplied memory map and assume that it did not change after waking from hibernation. This behaviour is required to work by Windows, which mandates to preserve runtime memory size and location after S4 wake.

   *Note*: This may be used to workaround buggy memory maps on older hardware, and is now considered rare legacy. Examples of such hardware are Ivy Bridge laptops with Insyde firmware, like Acer V3-571G. Do not use this unless you fully understand the consequences.

6. `EnableSafeModeSlide`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Patch bootloader to have KASLR enabled in safe mode.

   This option is relevant to the users that have issues booting to safe mode (e.g. by holding `shift` or using `-x` boot argument). By default safe mode forces `0` slide as if the system was launched with `slide=0` boot argument. This quirk tries to patch `boot.efi` to lift that limitation and let some other value (from `1` to `255`) be used. This quirk requires `ProvideCustomSlide` to be enabled.

   *Note*: The necessity of this quirk is determined by safe mode availability. If booting to safe mode fails, this option can be tried to be enabled.

7. `EnableWriteUnprotector`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Permit write access to UEFI runtime services code.

   This option bypasses `RX̂` permissions in code pages of UEFI runtime services by removing write protection (`WP`) bit from `CR0` register during their execution. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `FwRuntimeServices.efi`.

   *Note*: The necessity of this quirk is determined by early boot crashes of the firmware.

8. `ForceExitBootServices`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Retry `ExitBootServices` with new memory map on failure.

   Try to ensure that `ExitBootServices` call succeeds even with outdated MemoryMap key argument by obtaining current memory map and retrying `ExitBootServices` call.

   *Note*: The necessity of this quirk is determined by early boot crashes of the firmware. Do not use this unless you fully understand the consequences.

9. `ProtectCsmRegion`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Protect CSM region areas from relocation.

   Ensure that CSM memory regions are marked as ACPI NVS to prevent boot.efi or XNU from relocating or using them.

   *Note*: The necessity of this quirk is determined by artifacts and sleep wake issues. As `AvoidRuntimeDefrag` resolves a similar problem, no known firmwares should need this quirk. Do not use this unless you fully understand the consequences.

10. `ProtectSecureBoot`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Protect UEFI Secure Boot variables from being written.

Reports security violation during attempts to write to `db`, `dbx`, `PK`, and `KEK` variables from the operating system.

*Note*: This quirk mainly attempts to avoid issues with NVRAM implementations with problematic defragmentation, such as select Insyde or `MacPro5,1`.

11. `ProvideCustomSlide`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Provide custom KASLR slide on low memory.

    This option performs memory map analysis of your firmware and checks whether all slides (from `1` to `255`) can be used. As `boot.efi` generates this value randomly with `rdrand` or pseudo randomly `rdtsc`, there is a chance of boot failure when it chooses a conflicting slide. In case potential conflicts exist, this option forces macOS to use a pseudo random value among the available ones. This also ensures that `slide=` argument is never passed to the operating system for security reasons.

    *Note*: The necessity of this quirk is determined by `OCABC: Only N/256 slide values are usable!` message in the debug log. If the message is present, this option is to be enabled.

12. `SetupVirtualMap`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Setup virtual memory at `SetVirtualAddresses`.

    Select firmwares access memory by virtual addresses after `SetVirtualAddresses` call, which results in early boot crashes. This quirk workarounds the problem by performing early boot identity mapping of assigned virtual addresses to physical memory.

    *Note*: The necessity of this quirk is determined by early boot failures. Currently new firmwares with memory protection support (like OVMF) do not support this quirk due to acidanthera/bugtracker#719.

13. `ShrinkMemoryMap`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Attempt to join similar memory map entries.

    Select firmwares have very large memory maps, which do not fit Apple kernel, permitting up to `64` slots for runtime memory. This quirk attempts to unify contiguous slots of similar types to prevent boot failures.

    *Note*: The necessity of this quirk is determined by early boot failures. It is rare to need this quirk on Haswell or newer. Do not use unless you fully understand the consequences.

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

    This quirk is useful on Mac firmwares, which behave differently in different OS. For example, it is supposed to enable Intel GPU in Windows and Linux in some dual-GPU MacBook models.

# 8 Misc

## 8.1 Introduction

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

## 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\Microsoft\Boot\bootmgfw.efi` for Microsoft 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`, 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.

## 8.3 Boot Properties

1. ~~`BuiltinTextRenderer`~~`HibernateMode`
   **Type**: `plist` ~~`boolean`~~`string`
   **Failsafe**: ~~`false`~~`None`
   **Description**: ~~Enables experimental builtin text renderer.~~ Hibernation detection mode. The following modes are supported:

   ~~This option makes all text going through standard console output render through builtin text renderer bypassing firmware services. While still experimental and feature incomplete, this option effecitly avoids the need for various quirks like `ReplaceTabWithSpace` or `SanitiseClearScreen`. It should also increase the dimensions of the output area.~~

   ~~Since builtin text renderer works in graphics mode, extra care may need to be paid to~~

   - `None` — Avoid hibernation for your own good.
   - ~~`ConsoleBehaviourOs`~~`Auto` ~~, `ConsoleBehaviourUi`, `ConsoleControl`, and `IgnoreTextInGraphics` options. While individual for the target system, it is recommended to use~~ — Use RTC and NVRAM detection.

- ~~`ForceGraphics`~~`RTC` ~~and builtin `ConsoleControl` to avoid compatibility issues.~~
  ~~*Note*: Some Macs, namely~~ — Use RTC detection.
- ~~`MacPro5,1`~~`NVRAM` ~~, may have broken console output with newer GPUs, and thus enabling this option can be required for them.~~ — Use NVRAM detection.

2. ~~`ConsoleMode`~~`HideAuxiliary`
   **Type**: `plist` ~~`string`~~`boolean`
   **Failsafe**: ~~Empty string~~`false`
   **Description**: ~~Sets console output mode as specified with the~~ Hides auxiliary entries from picker menu by default.

   An entry is considered auxiliary when at least one of the following applies:

   - Entry is macOS recovery.
   - Entry is explicitly marked as ~~`WxH`~~`Auxiliary`.
   - Entry is system (e.g. ~~`80x24`~~`Clean NVRAM`)~~formatted string. Set to empty string not to change console mode~~ . ~~Set to `Max` to try to use largest available console mode~~

   To see all entries picker menu needs to be reloaded in extended mode by pressing `Spacebar` key. Hiding auxiliary entries may increase boot performance for multidisk systems.

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

3. `HideSelf`
   **Type**: `plist boolean`
   **Failsafe**: `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.

4. ~~`ConsoleBehaviourOs`~~`PickerAttributes`
   **Type**: `plist` ~~`string`~~`integer`
   **Failsafe**: ~~Empty string~~`0`
   **Description**: ~~Set console control behaviour upon operating system load~~Sets specific attributes for picker.

   ~~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 set different modes for the operating system booter (~~Builtin picker supports colour arguments as a sum of foreground and background colors according to UEFI specification. The value of black background and black foreground (~~`ConsoleBehaviourOs`~~`0`) ~~, 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~~is reserved. List of colour names:

   - ~~Empty string~~ `0x00` — ~~Do not modify console control mode.~~ `EFI_BLACK`
   - ~~`Text`~~`0x01` — ~~Switch to text mode.~~ `EFI_BLUE`
   - ~~`Graphics`~~`0x02` — ~~Switch to graphics mode.~~ `EFI_GREEN`
   - ~~`ForceText`~~`0x03` — ~~Switch to text mode and preserve it (requires `ConsoleControl`~~`EFI_CYAN` ~~).~~
   - ~~`ForceGraphics`~~`0x04` — ~~Switch to graphics mode and preserve it (require `ConsoleControl`~~`EFI_RED` ~~).~~
     ~~Hints:~~
   - ~~Unless empty works, firstly try to set `ConsoleBehaviourOs`~~`0x05` ~~to~~ — ~~`Graphics`~~`EFI_MAGENTA` ~~and `ConsoleBehaviourUi` to `Text`.~~
   - ~~On APTIO IV (Haswell and earlier) it is usually enough to have `ConsoleBehaviourOs`~~`0x06` ~~set to~~ — ~~`Graphics`~~`EFI_BROWN` ~~and `ConsoleBehaviourUi` set to `ForceText` to avoid visual glitches.~~
   - ~~On APTIO V (Broadwell and newer) `ConsoleBehaviourOs`~~`0x07` ~~set to~~ — ~~`ForceGraphics`~~`EFI_LIGHTGRAY` ~~and `ConsoleBehaviourUi` set to `ForceText` usually works best.~~
   - ~~On Apple firmwares `ConsoleBehaviourOs`~~`0x08` ~~set to~~ — ~~`Graphics`~~`EFI_DARKGRAY` ~~and `ConsoleBehaviourUi` set to `Text` is supposed to work best.~~
     ~~*Note*: `IgnoreTextInGraphics` and `SanitiseClearScreen` may need to be enabled for select firmware implementations. Particularly APTIO firmwares.~~
   - ~~`ConsoleBehaviourUi`~~`0x09` ~~**Type**: `plist string`**Failsafe**: Empty string**Description**: Set console control behaviour upon OpenCore user interface load. Refer to~~ — ~~`ConsoleBehaviourOs`~~`EFI_LIGHTBLUE` ~~description~~

~~for details.~~

- ~~HibernateMode~~0x0A ~~**Type**: — plist string~~EFI_LIGHTGREEN ~~**Failsafe**: None**Description**: Hibernation detection mode. The following modes are supported:~~
- ~~None~~0x0B — ~~Avoid hibernation for your own good.~~ EFI_LIGHTCYAN
- ~~Auto~~0x0C — ~~Use RTC and NVRAM detection.~~ EFI_LIGHTRED
- ~~RTC~~0x0D — ~~Use RTC detection.~~ EFI_LIGHTMAGENTA
- ~~NVRAM~~0x0E — ~~Use NVRAM detection.~~ EFI_YELLOW
- 0x0F — EFI_WHITE
- 0x00 — EFI_BACKGROUND_BLACK
- 0x10 — EFI_BACKGROUND_BLUE
- 0x20 — EFI_BACKGROUND_GREEN
- 0x30 — EFI_BACKGROUND_CYAN
- 0x40 — EFI_BACKGROUND_RED
- 0x50 — EFI_BACKGROUND_MAGENTA
- 0x60 — EFI_BACKGROUND_BROWN
- 0x70 — EFI_BACKGROUND_LIGHTGRAY

*Note*: This option may not work well with System text renderer. Setting a background different from black could help testing proper GOP functioning.

5. ~~HideSelf~~PickerAudioAssist
**Type**: plist boolean
**Failsafe**: 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.~~ Enable screen reader by default in boot picker.

For macOS bootloader screen reader preference is set in preferences.efires archive in isVOEnabled.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.

6. PollAppleHotKeys
**Type**: plist boolean
**Failsafe**: false
**Description**: Enable modifier hotkey handling in boot picker.

In addition to action hotkeys, which are partially described in ~~UsePicker~~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.

7. ~~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 FileVault 2 UEFI password interface and boot screen logo. Refer to 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`.~~

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.

10. `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.

11. ~~`UsePicker`~~`PickerMode`
    **Type**: `plist` ~~`boolean`~~`string`
    **Failsafe**: ~~`false`~~`Builtin`
    **Description**: ~~Use OpenCore built-in boot picker~~ 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:

    - ~~`UsePicker`~~`Builtin` ~~set to~~ — boot management is handled by OpenCore, a simple text only user interface is used.
    - ~~`false`~~~~`entirely disables`~~ `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 ~~this case~~ `Apple` mode it may additionally bypass policy enforcement. To implement `External` mode a custom user interface may utilise OpenCorePkg `OcBootManagementLib`~~to implement a user friendly boot picker oneself~~. Reference example of external graphics interface is provided in ExternalUi test driver.

    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`, `AppleUsbKbDxe`, 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 workaround this problem even without loading OpenCore `BootKicker` utility can be blessed.

## 8.4 Debug Properties

1. `DisableWatchDog`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **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`
   **Failsafe**: `0`
   **Description**: Delay in microseconds performed after every printed line visible onscreen (i.e. console).

3. `DisplayLevel`
   **Type**: `plist integer`, 64 bit
   **Failsafe**: `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`
   **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: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
```

5. `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.

6. ~~`RequireSignature`~~`Vault`
   **Type**: `plist` ~~`boolean`~~`string`
   **Failsafe**: ~~`true`~~`Secure`
   **Description**: ~~Require~~ 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~~ 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.

   ~~*Note*: `vault.sig` is used regardless of this option when public key is embedded into `OpenCore.efi`. Setting it to `true` will only ensure configuration sanity, and abort the boot process when public key is not set but was supposed to be used for verification.~~

7. ~~`RequireVault`**Type**: `plist boolean`**Failsafe**: `true`**Description**: Require `vault.plist` file present in `OC` directory.~~

   ~~This 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 script.~~

   ~~Regardless of the underlying filesystem, path name and case must match between `config.plist` and `vault.plist`.~~

   ~~*Note*: `vault.plist` is tried to be read regardless of the value of this option, but setting it to `true` will ensure configuration sanity, and abort the boot process.~~

   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.

8. `ScanPolicy`
**Type**: `plist integer`, 32 bit
**Failsafe**: `0xF0103`
**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.

*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`

## 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: `Shell.efi`.

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask`
  Combined `FirmwareFeaturesMask` and `ExtendedFirmwareFeaturesMask`. Present on newer Macs to avoid extra parsing of SMBIOS tables.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID`
  Hardware `BoardProduct` (e.g. `Mac-35C1E88140C3E6CF`). Not present on real Macs, but used to avoid extra parsing of SMBIOS tables, especially in boot.efi.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB`
  Hardware `BoardSerialNumber`. Override for MLB. Present on newer Macs (2013+ at least).
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM`
  Hardware ROM. Override for ROM. Present on newer Macs (2013+ at least).
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:prev-lang:kbd`
  ASCII string defining default keyboard layout. Format is `lang-COUNTRY:keyboard`, e.g. `ru-RU:252` for Russian locale and ABC keyboard. Also accepts short forms: `ru:252` or `ru:0` (U.S. keyboard, compatible with 10.9). Full decoded keyboard list from `AppleKeyboardLayouts-L.dat` can be found here. Using non-latin keyboard on 10.14 will not enable ABC keyboard, unlike previous and subsequent macOS versions, and is thus not recommended in case you need 10.14.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:security-mode`
  ASCII string defining FireWire security mode. Legacy, can be found in IOFireWireFamily source code in IOFireWireController.cpp. It is recommended not to set this variable, which may speedup system startup. Setting to `full` is equivalent to not setting the variable and `none` disables FireWire security.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:UIScale`
  One-byte data defining boot.efi user interface scaling. Should be **01** for normal screens and **02** for HiDPI screens.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:DefaultBackgroundColor`
  Four-byte `RGBA` data defining boot.efi user interface background colour. Standard colours include **BF BF BF 00** (Light Gray) and **00 00 00 00** (Syrah Black). Other colours may be set at user's preference.

## 9.5 Other Variables

The following variables may be useful for certain configurations or troubleshooting:

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args`
  Kernel arguments, used to pass configuration to Apple kernel and drivers. There are many arguments, which 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`)
  - `batman=VALUE` (`AppleSmartBatteryManager` debug mask)
  - `batman-nosmc=1` (disable `AppleSmartBatteryManager` SMC interface)
  - `cpus=VALUE` (maximum number of CPUs used)
  - `debug=VALUE` (debug mask)
  - `io=VALUE` (`IOKit` debug mask)
  - `keepsyms=1` (show panic log debug symbols)
  - `kextlog=VALUE` (kernel extension loading debug mask)
  - `nv_disable=1` (disables NVIDIA GPU acceleration)
  - `nvda_drv=1` (legacy way to enable NVIDIA web driver, removed in 10.12)
  - `npci=0x2000` (legacy, disables `kIOPCIConfiguratorPFM64`)
  - `lapic_dont_panic=1`
  - `slide=VALUE` (manually set KASLR slide)
  - `smcdebug=VALUE` (`AppleSMC` debug mask)
  - `-amd_no_dgpu_accel` (alternative to WhateverGreen's `-radvesa` for new GPUs)
  - `-nehalem_error_disable`
  - `-no_compat_check` (disable model checking)
  - `-s` (single mode)
  - `-v` (verbose mode)
  - `-x` (safe mode)

  There are multiple external places summarising macOS argument lists: example 1, example 2.
- `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 loggingcontrol:~~ `0x`. At different stages boot.efi will request different debugging (logging) modes (e.g. after `ExitBootServices` it will only print to serial). Several booter arguments control whether these requests will succeed. The list of known requests is covered below:

- `0x00` – `INIT`.
- `0x01` – `VERBOSE` (e.g. `-v`, force console logging).
- `0x02` – `EXIT`.
- `0x03` – `RESET:OK`.
- `0x04` – `RESET:FAIL` (e.g. unknown `board-id`, hibernate mismatch, panic loop, etc.).
- `0x05` – `RESET:RECOVERY`.
- `0x06` – `RECOVERY`.
- `0x07` – `REAN:START`.
- `0x08` – `REAN:END`.
- `0x09` – `DT` (can no longer log to DeviceTree).
- `0x0A` – `EXITBS:START` (forced serial only).
- `0x0B` – `EXITBS:END` (forced serial only).
- `0x0C` – `UNKNOWN`.

In 10.15 debugging support was mostly broken before 10.15.4 due to some kind of refactoring and introduction of a new debug protocol. Some of the arguments and their values below may not be valid for versions prior to 10.15.4. The list of known arguments is covered below:

- `boot-save-log=VALUE` — debug log save mode for normal boot.
  * `0`
  * `1`
  * `2` — (default).
  * `3`
  * `4` — (save to file).
- `wake-save-log=VALUE` — debug log save mode for hibernation wake.
  * `0` — disabled.
  * `1`
  * `2` — (default).
  * `3` — (unavailable).
  * `4` — (save to file, unavailable).
- `breakpoint=VALUE` — enables debug breaks (missing in production `boot.efi`).
  * `0` — disables debug breaks on errors (default).
  * `1` — enables debug breaks on errors.
- `console=VALUE` — enables console logging.
  * `0` — disables console logging.
  * `1` — enables console logging when debug protocol is missing (default).
  * `2` — enables console logging unconditionally (unavailable).
- `embed-log-dt=VALUE` — enables DeviceTree logging.
  * `0` — disables DeviceTree logging (default).
  * `1` — enables DeviceTree logging.
- `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.
- `log-level=VALUE` — log level bitmask.
  * `0x01` — enables trace logging (default).
- `serial=VALUE` — enables serial logging.
  * `0` — disables serial logging (default).
  * `1` — enables serial logging for `EXITBS:END` onwards.
  * `1` — enables serial logging for `EXITBS:START` onwards.
  * `3` — enables serial logging when debug protocol is missing.
  * `4` — enables serial logging unconditionally.
- `timestamps=VALUE` — enables timestamp logging.
  * `0` — disables timestamp logging.
  * `1` — enables timestamp logging (default).
- `log=VALUE` — deprecated starting from 10.15.
  * `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` — deprecated starting from 10.15.
    * 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` — deprecated starting from 10.15. 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.~~

  *Note*: To quickly see verbose output from `boot.efi` on versions before 10.15 set `bootercfg` to `log=1`.
- `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.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:StartupMute`
  Mute startup chime sound in firmware audio support. 8-bit integer. The value of `0x00` means unmuted. Missing variable or any other value means muted. This variable only affects Gibraltar machines (T2).
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:SystemAudioVolume`
  System audio volume level for firmware audio support. 8-bit integer. The bit of `0x80` means muted. Lower bits are used to encode volume range specific to installed audio codec. The value is capped by `MaximumBootBeepVolume` AppleHDA layout value to avoid too loud audio playback in the firmware.

3. `BIOSReleaseDate`
   **Type**: `plist string`
   **Failsafe**: OEM specified
   **SMBIOS**: BIOS Information (Type 0) — BIOS Release Date
   **Description**: Firmware release date. Similar to `BIOSVersion`. May look like `12/08/2017`.

4. `SystemManufacturer`
   **Type**: `plist string`
   **Failsafe**: OEM specified
   **SMBIOS**: System Information (Type 1) — Manufacturer
   **Description**: OEM manufacturer of the particular board. Shall not be specified unless strictly required. Should *not* contain `Apple Inc.`, as this confuses numerous services present in the operating system, such as firmware updates, eficheck, as well as kernel extensions developed in Acidanthera, such as Lilu and its plugins. In addition it will also make some operating systems like Linux unbootable.

5. `SystemProductName`
   **Type**: `plist string`
   **Failsafe**: OEM specified
   **SMBIOS**: System Information (Type 1), Product Name
   **Description**: Preferred Mac model used to mark the device as supported by the operating system. This value must be specified by any configuration for later automatic generation of the related values in this and other SMBIOS tables and related configuration parameters. If `SystemProductName` is not compatible with the target operating system, `-no_compat_check` boot argument may be used as an override.

   *Note*: If `SystemProductName` is unknown, and related fields are unspecified, default values should be assumed as being set to `MacPro6,1` data. The list of known products can be found in `MacInfoPkg`.

6. `SystemVersion`
   **Type**: `plist string`
   **Failsafe**: OEM specified
   **SMBIOS**: System Information (Type 1) — Version
   **Description**: Product iteration version number. May look like `1.1`.

7. `SystemSerialNumber`
   **Type**: `plist string`
   **Failsafe**: OEM specified
   **SMBIOS**: System Information (Type 1) — Serial Number
   **Description**: Product serial number in defined format. Known formats are described in macserial.

8. `SystemUUID`
   **Type**: `plist string`, GUID
   **Failsafe**: OEM specified
   **SMBIOS**: System Information (Type 1) — UUID
   **Description**: A UUID is an identifier that is designed to be unique across both time and space. It requires no central registration process.

9. `SystemSKUNumber`
   **Type**: `plist string`
   **Failsafe**: OEM specified
   **SMBIOS**: System Information (Type 1) — SKU Number
   **Description**: Mac Board ID (`board-id`). May look like `Mac-7BA5B2D9E42DDD94` or `Mac-F221BEC8` in older models. Sometimes it can be just empty.

10. `SystemFamily`
    **Type**: `plist string`
    **Failsafe**: OEM specified
    **SMBIOS**: System Information (Type 1) — Family
    **Description**: Family name. May look like `iMac Pro`.

11. `BoardManufacturer`
    **Type**: `plist string`
    **Failsafe**: OEM specified

# 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 Properties

1. `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.efires` 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.

2. `ConnectDrivers`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Perform UEFI controller connection after driver loading.

   This option is useful for loading ~~filesystem drivers , which usually follow~~ drivers following UEFI driver model ~~, and~~ 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.

3. `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. 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 include:

   - `ApfsDriverLoader` — APFS file system bootstrap driver adding the support of embedded APFS drivers in bootable APFS containers in UEFI firmwares.

- `AudioDxe` — HDA audio support driver in UEFI firmwares for most Intel and some other analog audio controllers. Refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
- `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.
- `FwRuntimeServices` — `OC_FIRMWARE_RUNTIME` protocol implementation that increases the security of ~~OpenCore~~ OpenCore and Lilu by supporting read-only and write-only NVRAM variables. Some ~~quirks, like~~ commonly used quirks, e.g. `RequestBootVarRouting`, require this driver for proper function. Due to the nature of being a runtime driver, i.e. functioning in parallel with the target operating system, it cannot be implemented within OpenCore itself, but is bundled with OpenCore releases.
- `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.
- `AppleUsbKbDxe` — 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.
- `VBoxHfs` — HFS file system driver with bless support. This driver is an alternative to a closed source ~~`HFSPlus`~~`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.

To compile the drivers from UDK (EDK II) use the same command you do 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
```

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

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

6. `Protocols`
   **Type**: `plist dict`
   **Failsafe**: None
   **Description**: Force builtin versions of select protocols described in Protocols Properties section below.

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

7. `Quirks`
   **Type**: `plist dict`
   **Failsafe**: None
   **Description**: Apply individual firmware quirks described in Quirks Properties section below.

## 11.3 Audio Properties

1. `AudioCodec`
   **Type**: `plist integer`
   **Failsafe**: empty string
   **Description**: Codec address on the specified audio controller for audio support.

   Normally this contains first audio codec address on the builtin analog audio controller (`HDEF`). Audio codec addresses, e.g. 2, can be found in the debug log (marked in bold):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   As an alternative this value can be obtained from `IOHDACodecDevice` class in I/O Registry containing it in `IOHDACodecAddress` field.

2. `AudioDevice`
   **Type**: `plist string`
   **Failsafe**: `0`
   **Description**: Device path of the specified audio controller for audio support.

   Normally this contains builtin analog audio controller (`HDEF`) device path, e.g. `PciRoot(0x0)/Pci(0x1b,0x0)`. The list of recognised audio controllers can be found in the debug log (marked in bold):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   As an alternative **`gfxutil -f HDEF`** command can be used in macOS. Specifying empty device path will result in the first available audio controller to be used.

3. `AudioOut`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Index of the output port of the specified codec starting from 0.

   Normally this contains the index of the green out of the builtin analog audio controller (`HDEF`). The number of output nodes (`N`) in the debug log (marked in bold):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   The quickest way to find the right port is to bruteforce the values from `0` to `N - 1`.

4. `AudioSupport`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Activate audio support by connecting to a backend driver.

   Enabling this setting routes audio playback from builtin protocols to a dedicated audio port (`AudioOut`) of the specified codec (`AudioCodec`) located on the audio controller (`AudioDevice`).

5. `MinimumVolume`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Minimal heard volume level from `0` to `100`.

Screen reader will use this volume level, when the calculated volume level is less than `MinimumVolume`. Boot chime sound will not play if the calculated volume level is less than `MinimumVolume`.

6. `PlayChime`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Play chime sound at startup.

   Enabling this setting plays boot chime through builtin audio support. Volume level is determined by `MinimumVolume` and `VolumeAmplifier` settings and `SystemAudioVolume` NVRAM variable.

   *Note*: this setting is separate from `StartupMute` NVRAM variable to avoid conflicts when the firmware is able to play boot chime.

7. `VolumeAmplifier`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Multiplication coefficient for system volume to raw volume linear translation from `0` to `1000`.

   Volume level range read from `SystemAudioVolume` varies depending on the codec. To transform read value in `[0, 127]` range into raw volume range `[0, 100]` the read value is scaled to `VolumeAmplifier` percents:

$$RawVolume = MIN(\frac{SystemAudioVolume * VolumeAmplifier}{100}, 100)$$

   *Note*: the transformation used in macOS is not linear, but it is very close and this nuance is thus ignored.

## 11.4 Input Properties

1. `KeyForgetThreshold`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Remove key unless it was submitted during this timeout in milliseconds.

   `AppleKeyMapAggregator` protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers only report key presses as interrupts and pressing and holding the key on the keyboard results in subsequent submissions of this key with some defined time interval. As a result we use a timeout to remove once pressed keys from the buffer once the timeout expires and no new submission of this key happened.

   This option allows to set this timeout based on your platform. The recommended value that works on the majority of the platforms is `5` milliseconds. For reference, holding one key on VMware will repeat it roughly every `2` milliseconds and the same value for APTIO V is `3-4` milliseconds. Thus it is possible to set a slightly lower value on faster platforms and slightly higher value on slower platforms for more responsive input.

2. `KeyMergeThreshold`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Assume simultaneous combination for keys submitted within this timeout in milliseconds.

   Similarly to `KeyForgetThreshold`, this option works around the sequential nature of key submission. To be able to recognise simultaneously pressed keys in the situation when all keys arrive sequentially, we are required to set a timeout within which we assume the keys were pressed together.

   Holding multiple keys results in reports every `2` and `1` milliseconds for VMware and APTIO V respectively. Pressing keys one after the other results in delays of at least `6` and `10` milliseconds for the same platforms. The recommended value for this option is `2` milliseconds, but it may be decreased for faster platforms and increased for slower.

3. `KeySupport`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable internal keyboard input translation to `AppleKeyMapAggregator` protocol.

This option activates the internal keyboard interceptor driver, based on `AppleGenericInput` aka (`AptioInputFix`), to fill `AppleKeyMapAggregator` database for input functioning. In case a separate driver is used, such as `AppleUsbKbDxe`, this option should never be enabled.

4. `KeySupportMode`
   **Type**: `plist string`
   **Failsafe**: empty string
   **Description**: Set internal keyboard input translation to `AppleKeyMapAggregator` protocol mode.

   - `Auto` — Performs automatic choice as available with the following preference: `AMI`, `V2`, `V1`.
   - `V1` — Uses UEFI standard legacy input protocol `EFI_SIMPLE_TEXT_INPUT_PROTOCOL`.
   - `V2` — Uses UEFI standard modern input protocol `EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL`.
   - `AMI` — Uses APTIO input protocol `AMI_EFIKEYCODE_PROTOCOL`.

5. `KeySwap`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Swap `Command` and `Option` keys during submission.

   This option may be useful for keyboard layouts with `Option` key situated to the right of `Command` key.

6. `PointerSupport`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable internal pointer driver.

   This option implements standard UEFI pointer protocol (`EFI_SIMPLE_POINTER_PROTOCOL`) through select OEM protocols. The option may be useful on Z87 ASUS boards, where `EFI_SIMPLE_POINTER_PROTOCOL` is broken.

7. `PointerSupportMode`
   **Type**: `plist string`
   **Failsafe**: empty string
   **Description**: Set OEM protocol used for internal pointer driver.

   Currently the only supported variant is `ASUS`, using specialised protocol available on select Z87 and Z97 ASUS boards. More details can be found in `LongSoft/UefiTool#116`.

8. `TimerResolution`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Set architecture timer resolution.

   This option allows to update firmware architecture timer period with the specified value in `100` nanosecond units. Setting a lower value generally improves performance and responsiveness of the interface and input handling.

   The recommended value is `50000` (5 milliseconds) or slightly higher. Select ASUS Z87 boards use `60000` for the interface. Apple boards use `100000`. You may leave it as `0` in case there are issues.

## 11.5 Output Properties

1. `TextRenderer`
   **Type**: `plist string`
   **Failsafe**: `BuiltinGraphics`
   **Description**: Chooses renderer for text going through standard console output.

   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`.
- `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`.

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 color before switching to text mode.

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

5. `DirectGopRendering`
   **Type**: `plist boolean`
   **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.

6. `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.

7. `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.

8. `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.

9. `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.

10. `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.

## 11.6 Protocols 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. `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.

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

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

6. `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.

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

8. ~~`ConsoleControl`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Replaces Console Control protocol with a builtin version.~~

   ~~macOS bootloader requires console control protocol for text output, which some firmwares miss. This option is required to be set when the protocol is already available in the firmware, and other console control options are used, such as `IgnoreTextInGraphics`, `SanitiseClearScreen`, and sometimes `ConsoleBehaviourOs` with `ConsoleBehaviourUi`).~~

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

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

11. `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.

12. `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.

13. `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.

14. `UnicodeCollation`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Forcibly reinstalls unicode collation services with builtin version. Should be set to `true` to ensure UEFI Shell compatibility on platforms providing broken unicode collation. In general legacy Insyde and APTIO platforms on Ivy Bridge and earlier are affected.

## 11.7 Quirks Properties

1. ~~`AvoidHighAlloc`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Advises allocators to avoid allocations above first 4 GBs of RAM.~~

   ~~This is a workaround for select board firmwares, namely GA-Z77P-D3 (rev. 1.1), failing to properly access higher memory in UEFI Boot Services. On these boards this quirk is required for booting entries that need to allocate large memory chunks, such as macOS DMG recovery entries. On unaffected boards it may cause boot failures, and thus strongly not recommended. For known issues refer to .~~

2. ~~`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 color before switching to text mode.~~

   ~~*Note*: `ConsoleControl` should be set to `true` for this to work.~~

3. `ExitBootServicesDelay`
**Type**: `plist integer`
**Failsafe**: `0`
**Description**: Adds delay in microseconds after `EXIT_BOOT_SERVICES` event.

   This is a very ugly quirk to circumvent "Still waiting for root device" message on select APTIO IV firmwares, namely ASUS Z87-Pro, when using FileVault 2 in particular. It seems that for some reason they execute code in parallel to `EXIT_BOOT_SERVICES`, which results in SATA controller being inaccessible from macOS. A better approach should be found in some future. Expect 3-5 seconds to be enough in case the quirk is needed.

4. `IgnoreInvalidFlexRatio`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Select firmwares, namely APTIO IV, may contain invalid values in `MSR_FLEX_RATIO` (`0x194`) MSR register. These values may cause macOS boot failure on Intel platforms.

   *Note*: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required.

5. ~~`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*: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required. This option may hide onscreen error messages. `ConsoleControl` may need to be set to `true` for this to work.~~

6. ~~`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*: `ConsoleControl` may need to be set to `true` for this to work.~~

7. ~~`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.~~

8. ~~`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.~~

9. `ReleaseUsbOwnership`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Attempt to detach USB controller ownership from the firmware driver. While most firmwares 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.

10. `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 `FwRuntimeServices.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.

11. `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 `FwRuntimeServices.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.

12. ~~`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*: `ConsoleControl` may need to be set to `true` for this to work. On all known affected systems `ConsoleMode` had to be set to empty string for this to work.~~

13. `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.

# 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. The 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
```

```
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 419430366
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048         1023999   499.0 MiB   2700  Basic data partition
   2         1024000         1226751   99.0 MiB    EF00  EFI system partition
   3         1226752         1259519   16.0 MiB    0C01  Microsoft reserved ...
   4         1259520       419428351   199.4 GiB   0700  Basic data partition

Command (? for help): c
Partition number (1-4): 4
Enter name: BOOTCAMP

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING PARTITIONS!!

Do you want to proceed? (Y/N): Y
OK; writing new GUID partition table (GPT) to \\.\physicaldrive0.
Disk synchronization succeeded! The computer should now use the new partition table.
The operation has completed successfully.
```

Listing 3: Relabeling Windows volume

**How to choose Windows BOOTCAMP with custom NTFS drivers?**

Third-party drivers providing NTFS support, such as NTFS-3G, Paragon NTFS, Tuxera NTFS or Seagate Paragon Driver break certain macOS functionality, including Startup Disk preference pane normally used for operating system selection. While the recommended option remains not to use such drivers as they commonly corrupt the filesystem, and prefer the driver bundled with macOS with optional write support ( command or GUI), there still exist vendor-specific workarounds for their products: Tuxera, Paragon, etc.

## 12.2 Debugging

Similar to other projects working with hardware OpenCore supports auditing and debugging. The use of `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, refer to Debugging the XNU Kernel with IDA Pro for more details.

To obtain the log during boot you can make the use of serial port debugging. Serial port debugging is enabled in `Target`, e.g. `0xB` for onscreen with serial. OpenCore uses `115200` baud rate, `8` data bits, no parity, and `1` stop bit. For macOS your best choice are CP2102-based UART devices. Connect motherboard `TX` to USB UART `RX`, and motherboard `GND` to USB UART `GND`. Use `screen` utility to get the output, or download GUI software, such as CoolTerm.

*Note*: On several motherboards (and possibly USB UART dongles) PIN naming may be incorrect. It is very common to have `GND` swapped with `RX`, thus you have to connect motherboard "`TX`" to USB UART `GND`, and motherboard "`GND`" to USB UART `RX`.

Remember to enable `COM` port in firmware settings, and never use USB cables longer than 1 meter to avoid output corruption. To additionally enable XNU kernel serial output you will need `debug=0x8` boot argument.

## 12.3 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: `Misc` → `Debug` → `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. For early boot troubleshooting, for instance, when OpenCore menu does not appear, using UEFI Shell may help to see early debug messages.

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. **How to choose the default boot entry?**

   OpenCore uses the primary UEFI boot option to select the default entry. This choice can be altered from UEFI Setup, with the macOS Startup Disk preference, or the Windows Boot Camp Control Panel. Since choosing OpenCore's `BOOTx64.EFI` as a primary boot option limits this functionality in addition to several firmwares deleting incompatible boot options, potentially including those created by macOS, you are strongly encouraged to use the `RequestBootVarRouting` quirk, which will preserve your selection made in the operating system within the OpenCore variable space. Note, that `RequestBootVarRouting` requires a separate driver for functioning.

4. **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 macrecovery.py tool from MacInfoPkg.

   For offline installation refer to How to create a bootable installer for macOS article. Apart from App Store and `softwareupdate` utility there also are third-party tools to download an offline image.

5. **Why do online recovery images (`*.dmg`) fail to load?**

   This may be caused by missing HFS+ driver, as all presently known recovery volumes have HFS+ filesystem~~.~~ ~~Another cause may be buggy firmware allocator, which can be worked around with `AvoidHighAlloc` UEFI quirk~~.

6. **Can I use this on Apple hardware or virtual machines?**

   Sure, most relatively modern Mac models including `MacPro5,1` and virtual machines are fully supported. Even though there are little to none specific details relevant to Mac hardware, some ongoing instructions can be found in acidanthera/bugtracker#377.

7. **Why do Find&Replace patches must equal in length?**

   For machine code (x86 code) it is not possible to do differently sized replacements due to relative addressing. For ACPI code this is risky, and is technically equivalent to ACPI table replacement, thus not implemented. More detailed explanation can be found on AppleLife.ru.

8. **How can I migrate from `AptioMemoryFix`?**

   Behaviour similar to that of `AptioMemoryFix` can be obtained by installing `FwRuntimeServices` driver and enabling the quirks listed below. Please note, that most of these are not necessary to be enabled. Refer to their individual descriptions in this document for more details.

   - `ProvideConsoleGop` (UEFI quirk)
   - `AvoidRuntimeDefrag`
   - `DiscardHibernateMap`
   - `EnableSafeModeSlide`
   - `EnableWriteUnprotector`
   - `ForceExitBootServices`
   - `ProtectCsmRegion`