OpenCore

Reference Manual (0.5.~~8~~.9)

[2020.05.13]

1 Introduction

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

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

1.1 Generic Terms

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

3 Setup

3.1 Directory Structure

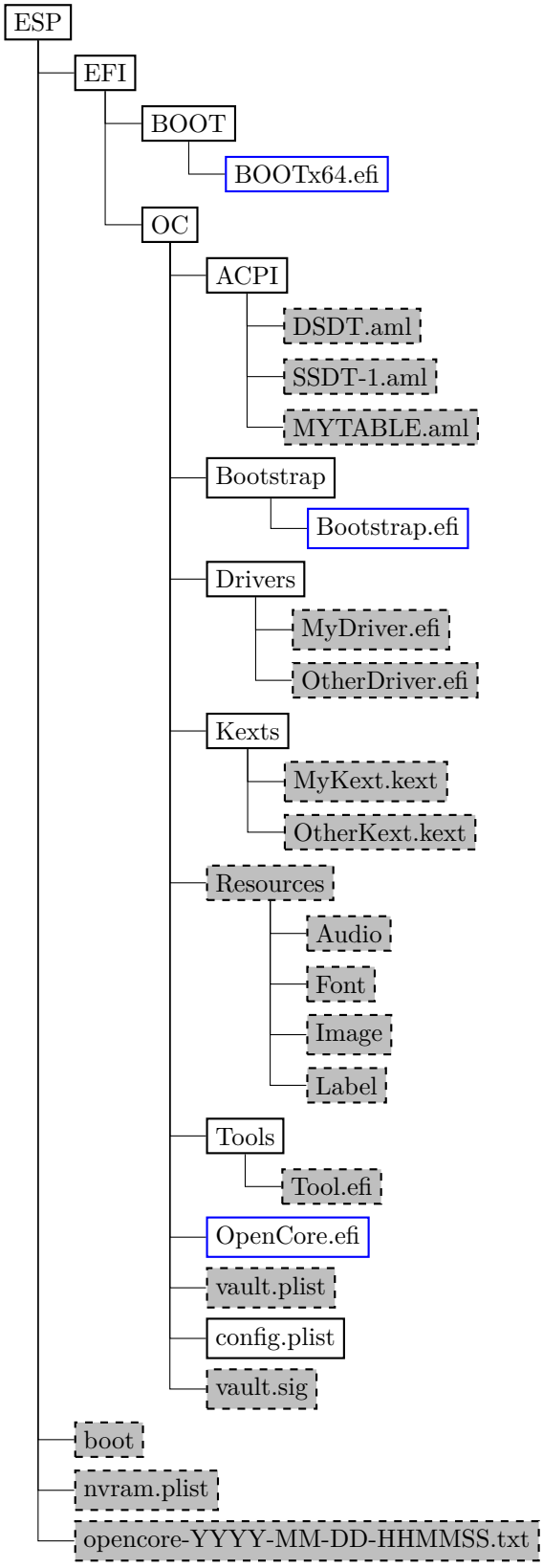

Figure 1. Directory Structure

When directory boot is used the directory structure used should follow the description on Directory Structure figure.

Available entries include:

- `BOOTx64.efi` ~~Initial booter~~and `Bootstrap.efi`
  Initial bootstrap loaders, which loads `OpenCore.efi` unless it was already started as a driver. `BOOTx64.efi` is loaded by the firmware by default according to UEFI specification, and `Bootstrap.efi` can be registered as a custom option to let OpenCore coexist with operating systems using `BOOTx64.efi` as their own loaders (e.g. Windows), see `BootProtect` for more details.
- `boot`
  Duet bootstrap loader, which initialises UEFI environment on legacy BIOS firmwares and loads `OpenCore.efi` similarly to other bootstrap loaders. Modern Duet bootstrap loader will default to `OpenCore.efi` on the same partition when present.
- `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. This directory also contains image files for graphical user interface. See OpenCanopy section for more details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  Main booter driver responsible for operating system loading.
- `vault.plist`
  Hashes for all files potentially loadable by `OC Config`.
- `config.plist`
  `OC Config`.
- `vault.sig`
  Signature for `vault.plist`.
- `nvram.plist`
  OpenCore variable import file.
- `opencore-YYYY-MM-DD-HHMMSS.txt`
  OpenCore log file.

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

3.2 Installation and Upgrade

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

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

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

For upgrade purposes refer to `Differences.pdf` document, providing the information about the changes affecting the configuration compared to the previous release, and `Changelog.md` document, containing the list of modifications across all published updates.

3.3 Contribution

OpenCore can be compiled as an ordinary EDK II package. 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 required patches for the package are present in `Patches` directory.

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

Required external package dependencies include EfiPkg and MacInfoPkg.

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

```
git clone https://github.com/acidanthera/audk UDK
cd UDK
git clone https://github.com/acidanthera/DuetPkg
git clone https://github.com/acidanthera/EfiPkg
git clone https://github.com/acidanthera/MacInfoPkg
git clone https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
make -C BaseTools
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/DuetPkg/Include
-I/UefiPackages/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
-I/UefiPackages/OvmfPkg/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
-DNO_MSABI_VA_FUNCS=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`, `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.
- External pull requests and tagged commits must be validated. That said, commits in master may build but may not necessarily work.
- Internal branches should be named as follows: `author-name-date`, e.g. `vit9696-ballooning-20191026`.
- Commit messages should be prefixed with the primary module (e.g. library or code module) the changes were made in. For example, `OcGuardLib: Add OC_ALIGNED macro`. For non-library changes `Docs` or `Build` prefixes are used.

Design. The codebase is written in a subset of freestanding C11 (C17) supported by most modern toolchains used by EDK II. Applying common software development practices or requesting clarification is recommended if any particular case is not discussed below.

- Never rely on undefined behaviour and try to avoid implementation defined behaviour unless explicitly covered below (feel free to create an issue when a relevant case is not present).
- Use `OcGuardLib` to ensure safe integral arithmetics avoiding overflows. Unsigned wraparound should be relied on with care and reduced to the necessary amount.
- Check pointers for correct alignment with `OcGuardLib` and do not rely on the architecture being able to dereference unaligned pointers.
- Use flexible array members instead of zero-length or one-length arrays where necessary.
- Use static assertions (`STATIC_ASSERT`) for type and value assumptions, and runtime assertions (`ASSERT`) for precondition and invariant sanity checking. Do not use runtime assertions to check for errors as they should never alter control flow and potentially be excluded.
- Assume `UINT32`/`INT32` to be `int`-sized and use `%u`, `%d`, and `%x` to print them.
- Assume `UINTN`/`INTN` to be of unspecified size, and cast them to `UINT64`/`INT64` for printing with `%Lu`, `%Ld` and so on as normal.
- Do not rely on integer promotions for numeric literals. Use explicit casts when the type is implementation-dependent or suffixes when type size is known. Assume `U` for `UINT32` and `ULL` for `UINT64`.
- Do ensure unsigned arithmetics especially in bitwise maths, shifts in particular.
- `sizeof` operator should take variables instead of types where possible to be error prone. Use `ARRAY_SIZE` to obtain array size in elements. Use `L_STR_LEN` and `L_STR_SIZE` macros from `OcStringLib` to obtain string literal sizes to ensure compiler optimisation.
- Do not use `goto` keyword. Prefer early `return`, `break`, or `continue` after failing to pass error checking instead of nesting conditionals.
- Use `EFIAPI`, force UEFI calling convention, only in protocols, external callbacks between modules, and functions with variadic arguments.
- Provide inline documentation to every added function, at least describing its inputs, outputs, precondition, postcondition, and giving a brief description.

8 Misc

8.1 Introduction

This section contains miscellaneous configuration ~~entries for OpenCore behaviour that does not go to any other sections~~ affecting OpenCore operating system loading behaviour as well as other entries, which do not go to any other section.

OpenCore tries to follow "bless" model also known as "Apple Boot Policy". The primary specialty of "bless" model is to allow embedding boot options within the file system (and be accessible through a specialised driver) as well as supporting a broader range of predefined boot paths compared to the removable media list found in the UEFI specification.

Each partition will only be used for booting when it corresponds to "Scan policy": a set of restrictions to only use partitions with specific file systems and from specific device types. Scan policy behaviour is discussed in `ScanPolicy` property description.

Scan process starts with obtaining all the partitions filtered with "Scan policy". Each partition may produce multiple primary and alternate options. Primary options describe operating systems installed on this media. Alternate options describe recovery options for the operating systems on the media. It is possible for alternate options to exist without primary options and vice versa. Be warned that the options may not necessarily describe the operating systems on the same partition. Each primary and alternate option can be an auxiliary option or not, refer to `HideAuxiliary` for more details. Algorithm to determine boot options behaves as follows:

1. Obtain all available partition handles filtered by "Scan policy" (and driver availability).
2. Obtain all available boot options from `BootOrder` UEFI variable.
3. For each found boot option:
   - Retrieve device path of the boot option.
   - Perform fixups (e.g. NVMe subtype correction) and expansion (e.g. for Boot Camp) of the device path.
   - Obtain device handle by locating device path of the resulting device path (ignore it on failure).
   - Find device handle in the list of partition handles (ignore it if missing).
   - For disk device paths (not specifying a bootloader) execute "bless" (may return > 1 entry).
   - For file device paths check presence on the file system directly.
   - Exclude options with blacklisted filenames (refer to `BlacklistAppleUpdate` option).
   - On OpenCore boot partition exclude all OpenCore bootstrap files by header checks.
   - Mark device handle as *used* in the list of partition handles if any.
   - Register the resulting entries as primary options and determine their types.
     The option will become auxiliary for some types (e.g. Apple HFS recovery).
4. For each partition handle:
   - If partition handle is marked as *unused* execute "bless" primary option list retrieval.
     In case `BlessOverride` list is set, not only standard "bless" paths will be found but also custom ones.
   - Exclude options with blacklisted filenames (refer to
   - BlacklistAppleUpdate option).
   - On OpenCore boot partition exclude all OpenCore bootstrap files by header checks.
   - Register the resulting entries as primary options and determine their types if found.
     The option will become auxiliary for some types (e.g. Apple HFS recovery).
   - If partition already has primary options of "Apple Recovery" type proceed to next handle.
   - Lookup alternate entries by "bless" recovery option list retrieval and predefined paths.
   - Register the resulting entries as alternate auxiliary options and determine their types if found.
5. Custom entries and tools are added as primary options without any checks with respect to `Auxiliary`.
6. System entries (e.g. `Reset NVRAM`) are added as primary auxiliary options.

The display order of the boot options in the picker and the boot process are determined separately from the scanning algorithm. The display order as follows:

- Alternate options follow corresponding primary options, i.e. Apple recovery will be following the relevant macOS option whenever possible.
- Options will be listed in file system handle firmware order to maintain an established order across the reboots regardless of the chosen operating system for loading.
- Custom entries, tools, and system entries will be added after all other options.
- Auxiliary options will only show upon entering "Advanced Mode" in the picker (usually by pressing "Space").

The boot process is as follows:

- Try looking up first valid primary option through `BootNext` UEFI variable.
- On failure looking up first valid primary option through `BootOrder` UEFI variable.
- Mark the option as the default option to boot.
- Boot option through the picker or without it depending on the `ShowPicker` option.
- Show picker on failure otherwise.

Note 1: This process is meant to work reliably only when `RequestBootVarRouting` option is enabled or the firmware does not control UEFI boot options (`DuetPkg` or custom BDS). Without `BootProtect` it also is possible that other operating systems overwrite OpenCore, make sure to enable it if you plan to use them.

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

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

8.2 Properties

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

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

   Designed to be filled with `plist string` entries containing absolute UEFI paths to customised bootloaders, for example, `\EFI\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. `ConsoleAttributes`
   Type: `plist integer`
   Failsafe: `0`
   Description: Sets specific attributes for console.

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

- 0x00 — EFI_BLACK
- 0x01 — EFI_BLUE
- 0x02 — EFI_GREEN
- 0x03 — EFI_CYAN
- 0x04 — EFI_RED
- 0x05 — EFI_MAGENTA
- 0x06 — EFI_BROWN
- 0x07 — EFI_LIGHTGRAY
- 0x08 — EFI_DARKGRAY
- 0x09 — EFI_LIGHTBLUE
- 0x0A — EFI_LIGHTGREEN
- 0x0B — EFI_LIGHTCYAN
- 0x0C — EFI_LIGHTRED
- 0x0D — EFI_LIGHTMAGENTA
- 0x0E — 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.

2. `HibernateMode`
   Type: `plist string`
   Failsafe: `None`
   Description: Hibernation detection mode. The following modes are supported:

   - `None` — Avoid hibernation for your own good.
   - `Auto` — Use RTC and NVRAM detection.
   - `RTC` — Use RTC detection.
   - `NVRAM` — Use NVRAM detection.

3. `HideAuxiliary`
   Type: `plist boolean`
   Failsafe: `false`
   Description: 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 macOS Time Machine.
   - Entry is explicitly marked as `Auxiliary`.
   - Entry is system (e.g. `Clean NVRAM`).

   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.

4. ~~`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.~~

5. `PickerAttributes`
   Type: `plist integer`

Failsafe: `false`
Description: Enable `boot.efi` debug log saving to OpenCore log.

Note: This option only applies to 10.15.4 and newer.

2. `ApplePanic`
Type: `plist boolean`
Failsafe: `false`
Description: Save macOS kernel panic to OpenCore root partition.

The file is saved as `panic-YYYY-MM-DD-HHMMSS.txt`. It is strongly recommended to have `keepsyms=1` boot argument to see debug symbols in the panic log. In case it was not present `kpdescribe.sh` utility (bundled with OpenCore) may be used to partially recover the stacktrace.

Development and debug kernels produce more helpful kernel panics. Consider downloading and installing `KernelDebugKit` from developer.apple.com when debugging a problem. To activate a development kernel you will need to add a `kcsuffix=development` boot argument. Use `uname -a` command to ensure that your current loaded kernel is a development (or a debug) kernel.

In case OpenCore kernel panic saving mechanism was not used, kernel panics may still be found in `/Library/Logs/Diagnosti` directory. Starting with macOS Catalina kernel panics are stored in JSON format, so they need to be preprocessed before passing to `kpdescribe.sh`:

```
cat Kernel.panic | grep macOSProcessedStackshotData |
  python -c 'import json,sys;print(json.load(sys.stdin)["macOSPanicString"])'
```

3. `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.

4. `DisplayDelay`
Type: `plist integer`
Failsafe: `0`
Description: Delay in microseconds performed after every printed line visible onscreen (i.e. console).

5. `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`.

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

The following logging targets are supported:

- `0x01` (bit `0`) — Enable logging, otherwise all log is discarded.
- `0x02` (bit `1`) — Enable basic console (onscreen) logging.
- `0x04` (bit `2`) — Enable logging to Data Hub.
- `0x08` (bit `3`) — Enable serial port logging.
- `0x10` (bit `4`) — Enable UEFI variable logging.
- `0x20` (bit `5`) — Enable non-volatile UEFI variable logging.

- `0x40` (bit `6`) — Enable logging to file.

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

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

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

UEFI variable log does not include some messages and has no performance data. For safety reasons log size is limited to 32 kilobytes. Some firmwares may truncate it much earlier or drop completely if they have no memory. Using non-volatile flag will write the log to NVRAM flash after every printed line. To obtain UEFI variable log use the following command in macOS:

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

Warning: Some firmwares are reported to have broken NVRAM garbage collection. This means that they may not be able to always free space after variable deletion. Do not use non-volatile NVRAM logging without extra need on such devices.

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

File logging will create a file named `opencore-YYYY-MM-DD-HHMMSS.txt` at EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmwares are not reliable, and may corrupt data when writing files through UEFI. Log is attempted to be written in the safest manner, and thus is very slow. Ensure that `DisableWatchDog` is set to `true` when you use a slow drive.

8.5 Security Properties

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

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

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

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

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

4. `BlacklistAppleUpdate`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Ignore boot options trying to update Apple peripheral firmware (e.g. `MultiUpdater.efi`).

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

Valid values:

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

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

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

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

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

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

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

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

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

   ```
   u=$(nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path | sed 's/.*GPT,\([^,]*\),.*/\1/'); \
     if [ "$u" != "" ]; then sudo diskutil mount $u ; fi
   ```

   To obtain OpenCore version use the following command in macOS:

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

   To obtain OEM information use the following commands in macOS:

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

7. `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.

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

   Valid values:

   - `Optional` — require nothing, no vault is enforced, insecure.
   - `Basic` — require `vault.plist` file present in `OC` directory. This provides basic filesystem integrity verification and may protect from unintentional filesystem corruption.

9 NVRAM

9.1 Introduction

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

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

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14` (`APPLE_VENDOR_VARIABLE_GUID`)
- `7C436110-AB2A-4BBB-A880-FE41995C9F82` (`APPLE_BOOT_VARIABLE_GUID`)
- `8BE4DF61-93CA-11D2-AA0D-00E098032B8C` (`EFI_GLOBAL_VARIABLE_GUID`)
- `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` (`OC_VENDOR_VARIABLE_GUID`)

Note: Some of the variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Please ensure that variables of this section never collide with them, as behaviour is undefined otherwise.

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

1. Not all tools may be aware of protected namespaces.
   When `RequestBootVarRouting` is used `Boot`-prefixed variable access is restricted and protected in a separate namespace. To access the original variables tools have to be aware of `OC_FIRMWARE_RUNTIME` logic.
2. ~~Assigned NVRAM variables are not always allowed to exceed 512 bytes.This is true for `Boot`-prefixed variables when `RequestBootVarFallback` is used, and for overwriting volatile variables with non-volatile on UEFI 2.8 non-conformant firmwares.~~

9.2 Properties

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

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

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

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

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

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

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

   Variable loading happens prior to `Block` (and `Add`) phases. Unless `LegacyOverwrite` is enabled, it will not overwrite any existing variable. Variables allowed to be set must be specified in `LegacySchema`. Third-party scripts may be used to create `nvram.plist` file. An example of such script can be found in `Utilities`. The use of

- 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~~BGRA 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

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

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

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

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

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

WARNING: OpenCanopy is currently considered experimental and is not recommended for everyday use. Refer to acidanthera/bugtracker#759 for more details regarding the current limitations.

11.5 OpenRuntime

`OpenRuntime` is an OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol. This protocol implements multiple features required for OpenCore that are otherwise not possible to implement in OpenCore itself as they are needed to work in runtime, i.e. during operating system functioning. Feature highlights:

- NVRAM namespaces, allowing to isolate operating systems from accessing select variables (e.g. `RequestBootVarRouting` or `ProtectSecureBoot`).
- ~~NVRAM proxying, allowing to manipulate multiple variables on variable updates (e.g. `RequestBootVarFallback`).~~
- Read-only and write-only NVRAM variables, enhancing the security of OpenCore, Lilu, and Lilu plugins, like VirtualSMC, which implements `AuthRestart` support.
- NVRAM isolation, allowing to protect all variables from being written from an untrusted operating system (e.g. `DisableVariableWrite`).
- UEFI Runtime Services memory protection management to workaround read-only mapping (e.g. `EnableWriteUnprotector`).

11.6 Properties

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

2. `Audio`
   Type: `plist dict`

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. ~~`DirectGopCacheMode`~~**Type**~~: `plist string`~~**Failsafe**~~: Empty string~~**Description**~~: Cache mode for builtin graphics output protocol framebuffer.~~

~~Tuning cache mode may provide better rendering performance on some firmwares. Providing empty string leaves cache control settings to the firmware. Valid non-empty values are: `Uncacheable`, `WriteCombining`, and `WriteThrough`.~~

~~*Note*: This option is not supported on most hardware (see for more details).~~

6. `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.

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

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

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

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

9. `ProvideConsoleGop`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Ensure GOP (Graphics Output Protocol) on console handle.

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

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

10. `ReconnectOnResChange`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Reconnect console controllers after changing screen resolution.

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

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

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

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

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

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

11. `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.

12. `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.

13. `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.

14. `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.

15. `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.

11.12 Quirks Properties

1. ~~`ExitBootServicesDelay`~~`DeduplicateBootOrder`
   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.~~

2. ~~`IgnoreInvalidFlexRatio`~~**Type**~~:~~ `plist boolean`
   Failsafe: `false`
   Description: ~~Select firmwares, namely APTIO IV, may contain invalid values in~~ Remove duplicate entries in ~~`MSR_FLEX_RATIO`~~`BootOrder` ~~(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.

3. `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.

4. `RequestBootVarFallback`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Request fallback of some `Boot` prefixed variables from `OC_VENDOR_VARIABLE_GUID` to variable in `EFI_GLOBAL_VARIABLE_GUID`.

   This quirk requires `RequestBootVarRouting` to be enabled and therefore `OC_FIRMWARE_RUNTIME` protocol implemented in `OpenRuntime.efi`.

   By redirecting `Boot` prefixed variables to a separate GUID namespace with the help of `RequestBootVarRouting` quirk 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 removes all duplicates in `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 forinstance. variable attempting to resolve the consequences of the bugs upon OpenCore loading. It is recommended to use this key along with `BootProtect` option.

5. `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.

6. `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.

7. `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.

- 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 (bundled with OpenCore) 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.

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 OpenRuntime 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
   - ProtectMemoryRegions