OpenCore

Reference Manual (~~0.5.9~~0.6.0)

[2020.06.26]

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. Errata sheet is available in OpenCorePkg repository.

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). The intended audience of the document are programmers and engineers with basic understanding of macOS internals and UEFI functioning. For these reasons this document is available exclusively in English, and all other sources or translations of this document are unofficial and may contain errors.

Third-party articles, utilities, books, and alike may be more useful for a wider audience as they could provide guide-like material. However, they are prone to their authors' preferences, tastes, this document misinterpretation, and essential obsolescence. In case you use these sources, for example, Dortania's OpenCore Desktop Guide and related material, please ensure to follow this document for every made decision and judge its consequences.

Be warned that 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.

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

2.3 Configuration Structure

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

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

Root configuration entries consist of the following:

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

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

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

3.3 Contribution

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

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

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

```
git clone https://github.com/acidanthera/audk UDK
git clone --recursive --depth=1 https://github.com/acidanthera/audk UDK
cd UDK
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/OpenCorePkg/Include/AMI
-I/UefiPackages/OpenCorePkg/Include/Acidanthera
-I/UefiPackages/OpenCorePkg/Include/Apple
-I/UefiPackages/OpenCorePkg/Include/Apple/X64
-I/UefiPackages/OpenCorePkg/Include/Duet
-I/UefiPackages/OpenCorePkg/Include/Generic
-I/UefiPackages/OpenCorePkg/Include/Intel
-I/UefiPackages/OpenCorePkg/Include/Microsoft
-I/UefiPackages/OpenCorePkg/Include/VMware
-I/UefiPackages/OvmfPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
-DNO_MSABI_VA_FUNCS=1
```

Listing 2: ECC Configuration

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. `ProtectUefiServices`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Protect UEFI services from being overridden by the firmware.

    Some modern firmwares including both hardware and virtual machines, like VMware, may update pointers to UEFI services during driver loading and related actions. Consequentially this directly breaks other quirks that affect memory management, like `DevirtualiseMmio`, `ProtectMemoryRegions`, or `RebuildAppleMemoryMap`, and may also break other quirks depending on the effects of these.

    Note: On VMware the need for this quirk may be diagnosed by "Your Mac OS guest might run unreliably with more than one virtual core." message.

12. `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.

13. `ProvideMaxSlide`
    Type: `plist integer`
    Failsafe: `0`
    Description: Provide maximum KASLR slide when higher ones are unavailable.

    This option overrides the maximum slide of 255 by a user specified value between 1 and 254 inclusive when `ProvideCustomSlide` is enabled. It is believed that modern firmwares allocate pool memory from top to bottom, effectively resulting in free memory at the time of slide scanning being later used as temporary memory during kernel loading. In case those memory are unavailable, this option can stop evaluating higher slides.

    Note: The necessity of this quirk is determined by random boot failure when `ProvideCustomSlide` is enabled and the randomized slide fall into the unavailable range. When `AppleDebug` is enabled, usually the debug log may contain messages like `AAPL: [EB|'LD:LKC] } Err(0x9)`. To find the optimal value, manually append `slide=X` to `boot-args` and log the largest one that won't cause boot failure.

14. `RebuildAppleMemoryMap`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Generate Memory Map compatible with macOS.

    Apple kernel has several limitations in parsing UEFI memory map:

    - Memory map size must not exceed 4096 bytes as Apple kernel maps it as a single 4K page. Since some firmwares have very large memory maps (approximately over 100 entries) Apple kernel will crash at boot.
    - Memory attributes table is ignored. `EfiRuntimeServicesCode` memory statically gets `RX` permissions, and all other memory types get `RW` permissions. Since some firmware drivers may write to global variables

8 Misc

8.1 Introduction

This section contains miscellaneous configuration 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 (`OpenDuetPkg` or custom BDS). Without `BootProtect` it also is possible that other operating systems overwrite OpenCore, make sure to enable it if you plan to use them.

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

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

8.2 Properties

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

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

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

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

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

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

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

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

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

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

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~~ colours 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~~ Reset 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. PickerAttributes
   Type: plist integer
   Failsafe: 0
   Description: Sets specific attributes for picker.

   Different pickers may be configured through the attribute mask containing OpenCore-reserved (BIT0~BIT15) and

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

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

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

7. `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. Try to avoid frequent use of this option when dealing with flash drives as large I/O amounts may speedup memory wear and render this flash drive unusable in shorter time.

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

Drivers and tools:

- `BMF` — OpenCanopy, bitmap font
- `BS` — Bootstrap
- `GSTT` — GoptStop
- `HDA` — AudioDxe
- `KKT` — KeyTester
- `MMDD` — MmapDump
- `OCPAVP` — PavpProvision
- `OCRST` — ResetSystem
- `OCUI` — OpenCanopy
- `OC` — OpenCore main
- `VMOPT` — VerifyMemOpt

Libraries:

- `AAPL` — OcDebugLogLib, Apple EfiBoot logging
- `OCABC` — OcAfterBootCompatLib
- `OCAE` — OcAppleEventLib
- `OCAK` — OcAppleKernelLib
- `OCAU` — OcAudioLib
- `OCAV` — OcAppleImageVerificationLib
- `OCA` —- OcAcpiLib
- `OCBP` — OcAppleBootPolicyLib

- OCB — OcBootManagementLib
- OCCL — OcAppleChunkListLib
- OCCPU — OcCpuLib
- OCC — OcConsoleLib
- OCDH — OcDataHubLib
- OCDI — OcAppleDiskImageLib
- OCFSQ — OcFileLib, UnblockFs quirk
- OCFS — OcFileLib
- OCFV — OcFirmwareVolumeLib
- OCHS — OcHashServicesLib
- OCIC — OcImageConversionLib
- OCII — OcInputLib
- OCJS — OcApfsLib
- OCKM — OcAppleKeyMapLib
- OCL — OcDebugLogLib
- OCMCO — OcMachoLib
- OCME — OcHeciLib
- OCMM — OcMemoryLib
- OCPI — OcFileLib, partition info
- OCPNG — OcPngLib
- OCRAM — OcAppleRamDiskLib
- OCRTC — OcRtcLib
- OCSB — OcAppleSecureBootLib
- OCSMB — OcSmbiosLib
- OCSMC — OcSmcLib
- OCST — OcStorageLib
- OCS — OcSerializedLib
- OCTPL — OcTemplateLib
- OCUC — OcUnicodeCollationLib
- OCUT — OcAppleUserInterfaceThemeLib
- OCXML — OcXmlLib

8.5 Security Properties

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

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

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

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

3. `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 while NVRAM reset executed from OpenCore should not erase the boot option created in `Bootstrap`, executing NVRAM reset prior to loading OpenCore will remove it.

6. `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.

    * 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.

  Note: To see verbose output from boot.efi on modern macOS versions enable AppleDebug option. This will save the log to general OpenCore log. For versions before 10.15.4 set bootercfg to log=1. This will print verbose output onscreen.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg-once
  Booter arguments override removed after first launch. Otherwise equivalent to bootercfg.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:efiboot-perf-record
  Enable performance log saving in boot.efi. Performance log is saved to physical memory and is pointed by efiboot-perf-record-data and efiboot-perf-record-size variables. Starting from 10.15.4 it can also be saved to OpenCore log by AppleDebug option.
- 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:run-efi-updater
  Override EFI firmware updating support in macOS (MultiUpdater, ThorUtil, and so on). Setting this to No or alternative boolean-castable value will prevent any firmware updates in macOS starting with 10.10 at least.
- 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.

11.3 Tools

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

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

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

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

Listing 3: Blessing tool

Note 1: You may have to copy `/System/Library/CoreServices/BridgeVersion.bin` to `/Volumes/VOLNAME/DIR`.
Note 2: To be able to use `bless` you may have to disable System Integrity Protection.
Note 3: To be able to boot you may have to disable Secure Boot if present.

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

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

11.4 OpenCanopy

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

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

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

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

- `Cursor` — Mouse cursor (mandatory).
- `Selected` — Selected item (mandatory).
- `Selector` — Selecting item (mandatory).
- `HardDrive` — Generic OS (mandatory).
- `Apple` — Apple OS.
- `AppleRecv` — Apple Recovery OS.
- `AppleTM` — Apple Time Machine.
- `Windows` — Windows.
- `Other` — Custom entry (see `Entries`).

11.7 APFS Properties

1. `EnableJumpstart`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load embedded APFS drivers from APFS containers.

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

2. `GlobalConnect`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform full device connection during APFS loading.

   Instead of partition handle connection normally used for APFS driver loading every handle is connected recursively. This may take more time than usual but can be the only way to access APFS partitions on some firmwares like those found on older HP laptops.

3. `HideVerbose`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Hide verbose output from APFS driver.

   APFS verbose output can be useful for debugging.

4. `JumpstartHotPlug`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load APFS drivers for newly connected devices.

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

5. `MinDate`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal allowed APFS driver date.

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

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

6. `MinVersion`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal allowed APFS driver version.

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

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

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.

11.9 Input Properties

1. `KeyFiltering`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable keyboard input sanity checking.

   Apparently some boards like GA Z77P-D3 may return uninitialised data in `EFI_INPUT_KEY` with all input protocols. This option discards keys that are neither ASCII, nor are defined in the UEFI specification (see tables 107 and 108 in version 2.8).

2. `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.

   Note: Some platforms may require different values, higher or lower. For example, when detecting key misses in OpenCanopy try increasing this value (e.g. to `10`), and when detecting key stall, try decreasing this value. Since every platform is different it may be reasonable to check every value from `1` to `25`.

3. `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.

4. `KeySupport`
   Type: `plist boolean`

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~~ colour 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.

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

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. `DeduplicateBootOrder`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Remove duplicate entries in `BootOrder` 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~~ resetting 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.