OpenCore

Reference Manual (0.5.8.9)

[2020.05.31]

Contents

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~~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, ~~etc. , providing such material~~ 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 ~~still~~ use these sources, for example, ~~()~~Dortania's OpenCore Desktop Guide and related material, please ensure ~~following~~ to follow this document for every made decision and ~~judging~~ judge its consequences. ~~Regardless~~

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

Root configuration entries consist of the following:

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

It is possible to perform basic validation of the configuration by using `ConfigValidity` utility. Please note, that `ConfigValidity` 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 Setup

3.1 Directory Structure

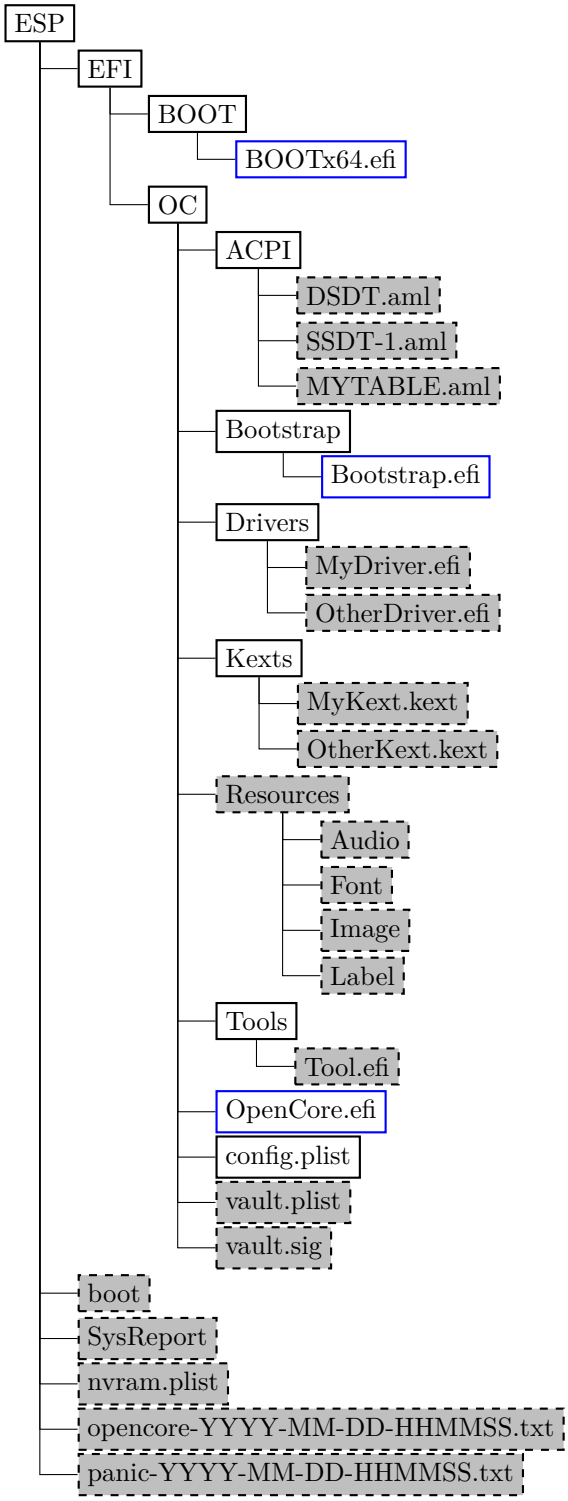

Figure 1. Directory Structure

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.
- `config.plist`
  `OC Config`.
- `vault.plist`
  Hashes for all files potentially loadable by `OC Config`.
- ~~`config.plist` `OC Config`.~~
- `vault.sig`
  Signature for `vault.plist`.
- `SysReport`
  Directory containing system reports generated by `SysReport` option.
- `nvram.plist`
  OpenCore variable import file.
- `opencore-YYYY-MM-DD-HHMMSS.txt`
  OpenCore log file.
- `panic-YYYY-MM-DD-HHMMSS.txt`
  Kernel panic 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~~OpenDuetPkg is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system you can install ~~DuetPkg~~OpenDuetPkg with a dedicated tool — BootInstall (bundled with OpenCore). Third-party utilities can be used to perform this on systems different from macOS.

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 and .~~

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

```
git clone https://github.com/acidanthera/audk UDK
cd UDK
git clone https://github.com/acidanthera/EfiPkg
git clone https://github.com/acidanthera/MacInfoPkg
git clone https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

For IDE usage Xcode projects are available in the root of the repositories. Another approach could be Sublime Text with EasyClangComplete plugin. Add `.clang_complete` file with similar content to your UDK root:

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/MdeModulePkg
-I/UefiPackages/MdeModulePkg/Include
-I/UefiPackages/MdeModulePkg/Include/X64
-I/UefiPackages/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
-I/UefiPackages/MacInfoPkg/Include
-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

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~~ contained in `OpenCorePkg` ~~are primary packages, and `EfiPkg`, `MacInfoPkg.dsc`) are dependent packages.~~ repository, which is the primary EDK II package.

- 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.
- Do not use `RETURN_STATUS`. Assume `EFI_STATUS` to be a matching superset that is to be always used when `BOOLEAN` is not enough.
- Security violations should halt the system or cause a forced reboot.

Codestyle. The codebase follows EDK II codestyle with few changes and clarifications.

- Write inline documentation for the functions and variables only once: in headers, where a header prototype is available, and inline for `static` variables and functions.
- Use line length of 120 characters or less, preferably 100 characters.
- Use spaces after casts, e.g. `(VOID *)(UINTN) Variable`.
- Use SPDX license headers as shown in acidanthera/bugtracker#483.

~~Debugging.~~

3.5 Debugging

The codebase incorporates EDK II debugging and few custom features to improve the experience.

- Use module prefixes, 2-5 letters followed by a colon (`:`), for debug messages. For `OpenCorePkg` use `OC:`, for libraries and drivers use their own unique prefixes.
- Do not use dots (`.`) in the end of debug messages and separate `EFI_STATUS`, printed by `%r`, with a hyphen (e.g. `OCRAM: Allocation of %u bytes failed - %r\n`).
- Use `DEBUG_CODE_BEGIN ()` and `DEBUG_CODE_END ()` constructions to guard debug checks that may potentially reduce the performance of release builds and are otherwise unnecessary.
- Use `DEBUG` macro to print debug messages during normal functioning, and `RUNTIME_DEBUG` for debugging after `EXIT_BOOT_SERVICES`.
- Use `DEBUG_VERBOSE` debug level to leave debug messages for future debugging of the code, which are currently not necessary. By default `DEBUG_VERBOSE` messages are ignored even in `DEBUG` builds.
- Use `DEBUG_INFO` debug level for all non critical messages (including errors) and `DEBUG_BULK_INFO` for extensive messages that should not appear in NVRAM log that is heavily limited in size. These messages are ignored in `RELEASE` builds.
- Use `DEBUG_ERROR` to print critical human visible messages that may potentially halt the boot process, and `DEBUG_WARN` for all other human visible errors, `RELEASE` builds included.

When trying to find the problematic change it is useful to rely on `git-bisect` functionality.

4 ACPI

4.1 Introduction

ACPI (Advanced Configuration and Power Interface) is an open standard to discover and configure computer hardware. ACPI specification defines the standard tables (e.g. `DSDT`, `SSDT`, `FACS`, `DMAR`) and various methods (e.g. `_DSM`, `_PRW`) for implementation. Modern hardware needs little changes to maintain ACPI compatibility, yet some of those are provided as a part of OpenCore.

To compile and disassemble ACPI tables iASL compiler can be used developed by ACPICA. GUI front-end to iASL compiler can be downloaded from Acidanthera/MaciASL.

ACPI changes apply globally (to every operating system) with the following effective order:

- `Patch` is processed.
- `Delete` is processed.
- `Add` is processed.
- `Quirks` are processed.

Applying the changes globally resolves the problems of incorrect operating system detection, which is not possible before the operating system boots according to the ACPI specification, operating system chainloading, and harder ACPI debugging. For this reason it may be required to carefully use `_OSI` method when writing the changes.

Applying the patches early makes it possible to write so called "proxy" patches, where the original method is patched in the original table and is implemented in the patched table.

4.2 Properties

1. `Add`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected tables from `OC/ACPI` directory.

   Designed to be filled with `plist dict` values, describing each ~~block~~ add entry. See Add Properties section below.

2. ~~Block~~`Delete`
   Type: `plist array`
   Failsafe: Empty
   Description: Remove selected tables from ACPI stack.

   Designed to be filled with `plist dict` values, describing each ~~block~~ delete entry. See Delete Properties section below.

3. `Patch`
   Type: `plist array`
   Failsafe: Empty
   Description: Perform binary patches in ACPI tables before table addition or removal.

   Designed to be filled with `plist dictionary` values describing each patch entry. See Patch Properties section below.

4. `Quirks`
   Type: `plist dict`
   Description: Apply individual ACPI quirks described in Quirks Properties section below.

4.3 Add Properties

1. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This ACPI table will not be added unless set to `true`.

3. `Path`
   Type: `plist string`
   Failsafe: Empty string
   Description: File paths meant to be loaded as ACPI tables. Example values include `DSDT.aml`, `SubDir/SSDT-8.aml`, `SSDT-USBX.aml`, etc.

   ACPI table load order follows the item order in the array. All ACPI tables load from `OC/ACPI` directory.

   Note: All tables but tables with `DSDT` table identifier (determined by parsing data not by filename) insert new tables into ACPI stack. `DSDT`, unlike the rest, performs replacement of DSDT table.

4.4 ~~Block~~ Delete Properties

1. `All`
   Type: `plist boolean`
   Failsafe: `false`
   Description: If set to `true`, all ACPI tables matching the condition will be ~~dropped~~deleted. Otherwise only first matched table.

2. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This ACPI table will not be removed unless set to `true`.

4. `OemTableId`
   Type: `plist data`, 8 bytes
   Failsafe: All zero
   Description: Match table OEM ID to be equal to this value unless all zero.

5. `TableLength`
   Type: `plist integer`
   Failsafe: `0`
   Description: Match table size to be equal to this value unless `0`.

6. `TableSignature`
   Type: `plist data`, 4 bytes
   Failsafe: All zero
   Description: Match table signature to be equal to this value unless all zero.

   Note: Make sure not to specify table signature when the sequence needs to be replaced in multiple places. Especially when performing different kinds of renames.

4.5 Patch Properties

1. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. `Count`
   Type: `plist integer`

- Try to avoid hacky changes like renaming `_PRW` or `_DSM` whenever possible.

Several cases, where patching actually does make sense, include:

- Refreshing `HPET` (or another device) method header to avoid compatibility checks by `_OSI` on legacy hardware. `_STA` method with `if ((OSFL () == Zero)) { If (HPTE) ... Return (Zero)` content may be forced to always return 0xF by replacing `A0 10 93 4F 53 46 4C 00` with `A4 0A 0F A3 A3 A3 A3 A3`.
- To provide custom method implementation with in an SSDT, for instance, to report functional key presses on a laptop, the original method can be replaced with a dummy name by patching `_Q11` with `XQ11`.

Tianocore AcpiAml.h source file may help understanding ACPI opcodes.

Note: Patches of different `Find` and `Replace` lengths are unsupported as they may corrupt ACPI tables and make you system unstable due to area relocation. If you need such changes you may utilities "proxy" patching or `NOP` the remaining area.

4.6 Quirks Properties

1. `FadtEnableReset`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Provide reset register and flag in FADT table to enable reboot and shutdown.

   Mainly required on legacy hardware and few laptops. Can also fix power-button shortcuts. Not recommended unless required.

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

3. `RebaseRegions`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Attempt to heuristically relocate ACPI memory regions. Not recommended.

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

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

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

4. `ResetHwSig`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reset `FACS` table `HardwareSignature` value to `0`.

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

5. `ResetLogoStatus`
   Type: `plist boolean`

5 Booter

5.1 Introduction

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

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

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

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

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

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

5.2 Properties

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

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

5.3 MmioWhitelist Properties

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

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`, ~~`ProtectCsmRegion`~~`ProtectMemoryRegions`, or ~~`ShrinkMemoryMap`~~`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. `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 at runtime, Apple kernel will crash at calling UEFI runtime services, unless driver `.data` section has `EfiRuntimeServicesData` type.

    To workaround these limitations this quirk applies memory attributes table permissions to memory map passed to Apple kernel and optionally attempts to unify contiguous slots of similar types if the resulting memory map exceeds 4 KB.

    Note 1: Since many firmwares come with incorrect memory protection table this quirk often comes in pair with `SyncRuntimePermissions`.

    Note 2: The necessity of this quirk is determined by early boot failures. This quirk replaces `EnableWriteUnprotector` on firmwares supporting memory attributes table (MAT).

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

6 DeviceProperties

6.1 Introduction

Device configuration is provided to macOS with a dedicated buffer, called `EfiDevicePathPropertyDatabase`. This buffer is a serialised map of DevicePaths to a map of property names and their values.

Property data can be debugged with gfxutil. To obtain current property data use the following command in macOS:

```
ioreg -lw0 -p IODeviceTree -n efi -r -x | grep device-properties |
  sed 's/.*<//;s/>.*//' > /tmp/device-properties.hex &&
  gfxutil /tmp/device-properties.hex /tmp/device-properties.plist &&
  cat /tmp/device-properties.plist
```

6.2 Properties

1. `Add`
   Type: `plist dict`
   Description: Sets device properties from a map (`plist dict`) of deivce paths to a map (`plist dict`) of variable names and their values in `plist metadata` format. Device paths must be provided in canonic string format (e.g. `PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x0)`). Properties will only be set if not present and not ~~blocked~~deleted.

   Note: Currently properties may only be (formerly) added by the original driver, so unless a separate driver was installed, there is no reason to ~~block~~ delete the variables.

2. ~~Block~~Delete
   Type: `plist dict`
   Description: Removes device properties from a map (`plist dict`) of deivce paths to an array (`plist array`) of variable names in `plist string` format.

6.3 Common Properties

Some known properties include:

- `device-id`
  User-specified device identifier used for I/O Kit matching. Has 4 byte data type.
- `vendor-id`
  User-specified vendor identifier used for I/O Kit matching. Has 4 byte data type.
- `AAPL,ig-platform-id`
  Intel GPU framebuffer identifier used for framebuffer selection on Ivy Bridge and newer. Has 4 byte data type.
- `AAPL,snb-platform-id`
  Intel GPU framebuffer identifier used for framebuffer selection on Sandy Bridge. Has 4 byte data type.
- `layout-id`
  Audio layout used for AppleHDA layout selection. Has 4 byte data type.

4. `MaxKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Blocks kernel driver on specified macOS version or older.

   Note: Refer to `Add MaxKernel` description for matching logic.

5. `MinKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Blocks kernel driver on specified macOS version or newer.

   Note: Refer to `Add MaxKernel` description for matching logic.

7.5 Emulate Properties

1. `Cpuid1Data`
   Type: `plist data`, 16 bytes
   Failsafe: All zero
   Description: Sequence of `EAX`, `EBX`, `ECX`, `EDX` values to replace `CPUID (1)` call in XNU kernel.

   This property serves for two needs:

   - Enabling support of an unsupported CPU model.
   - Enabling XCPM support for an unsupported CPU variant.

   Normally it is only the value of `EAX` that needs to be taken care of, since it represents the full CPUID. The remaining bytes are to be left as zeroes. Byte order is Little Endian, so for example, ~~`A9`~~ `C3` `06 03 00` stands for CPUID ~~`0x0306A90`~~`0x0306C3` (~~Ivy Bridge~~Haswell).

   For XCPM support it is recommended to use the following combinations.

   - Haswell-E (~~`0x306F2`~~`0x0306F2`) to Haswell (`0x0306C3`):
     `Cpuid1Data`: `C3 06 03 00 00 00 00 00 00 00 00 00 00 00 00 00`
     `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`
   - Broadwell-E (`0x0406F1`) to Broadwell (`0x0306D4`):
     `Cpuid1Data`: `D4 06 03 00 00 00 00 00 00 00 00 00 00 00 00 00`
     `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`

   ~~Further explanations can be found at . See~~ Keep in mind, that the following configurations are unsupported (at least out of the box):

   - Consumer Ivy Bridge (~~`Special NOTESfor`~~ `0x0306A9`) as Apple disabled XCPM for Ivy Bridge and recommends legacy power management for these CPUs. You will need to manually patch `_xcpm_bootstrap` to force XCPM on these CPUs instead of using this option.
   - Low-end CPUs (e.g. Haswell+ ~~low-end.~~ Pentium) as they are not supported properly by macOS. Legacy hacks for older models can be found in the `Special NOTES` section of acidanthera/bugtracker#365.

2. `Cpuid1Mask`
   Type: `plist data`, 16 bytes
   Failsafe: All zero
   Description: Bit mask of active bits in `Cpuid1Data`.

   When each `Cpuid1Mask` bit is set to 0, the original CPU bit is used, otherwise set bits take the value of `Cpuid1Data`.

7.6 Patch Properties

1. `Base`
   Type: `plist string`
   Failsafe: Empty string
   Description: Selects symbol-matched base for patch lookup (or immediate replacement) by obtaining the address of provided symbol name. Can be set to empty string to be ignored.

13. `Skip`
    Type: `plist integer`
    Failsafe: `0`
    Description: Number of found occurrences to be skipped before replacement is done.

7.7 Quirks Properties

1. `AppleCpuPmCfgLock`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables `PKG_CST_CONFIG_CONTROL` (`0xE2`) MSR modification in AppleIntelCPUPowerManagement.kext, commonly causing early kernel panic, when it is locked from writing.

   Certain firmwares lock `PKG_CST_CONFIG_CONTROL` MSR register. To check its state one can use bundled `VerifyMsrE2` tool. Select firmwares have this register locked on some cores only.

   As modern firmwares provide `CFG Lock` setting, which allows configuring `PKG_CST_CONFIG_CONTROL` MSR register lock, this option should be avoided whenever possible. For several APTIO firmwares not displaying `CFG Lock` setting in the GUI it is possible to access the option directly:

   (a) Download UEFITool and IFR-Extractor.
   (b) Open your firmware image in UEFITool and find `CFG Lock` unicode string. If it is not present, your firmware may not have this option and you should stop.
   (c) Extract the `Setup.bin` PE32 Image Section (the one UEFITool found) through `Extract Body` menu option.
   (d) Run IFR-Extractor on the extracted file (e.g. `./ifrextract Setup.bin Setup.txt`).
   (e) Find `CFG Lock, VarStoreInfo (VarOffset/VarName):` in `Setup.txt` and remember the offset right after it (e.g. `0x123`).
   (f) Download and run Modified GRUB Shell compiled by brainsucker or use a newer version by datasone.
   (g) Enter `setup_var 0x123 0x00` command, where `0x123` should be replaced by your actual offset, and reboot.

   WARNING: Variable offsets are unique not only to each motherboard but even to its firmware version. Never ever try to use an offset without checking.

2. `AppleXcpmCfgLock`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables `PKG_CST_CONFIG_CONTROL` (`0xE2`) MSR modification in XNU kernel, commonly causing early kernel panic, when it is locked from writing (XCPM power management).

   Note: This option should be avoided whenever possible. See `AppleCpuPmCfgLock` description for more details.

3. `AppleXcpmExtraMsrs`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables multiple MSR access critical for select CPUs, which have no native XCPM support.

   This is normally used in conjunction with `Emulate` section on Haswell-E, Broadwell-E, ~~Skylake-X~~Skylake-SP, and similar CPUs. More details on the XCPM patches are outlined in acidanthera/bugtracker#365.

   Note: Additional not provided patches will be required for Ivy Bridge or Pentium CPUs. It is recommended to use `AppleIntelCpuPowerManagement.kext` for the former.

4. `AppleXcpmForceBoost`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Forces maximum performance in XCPM mode.

   This patch writes `0xFF00` to `MSR_IA32_PERF_CONTROL` (`0x199`), effectively setting maximum multiplier for all the time.

   Note: While this may increase the performance, this patch is strongly discouraged on all systems but those explicitly dedicated to scientific or media calculations. In general only certain Xeon models benefit from the patch.

5. `CustomSMBIOSGuid`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Performs GUID patching for `UpdateSMBIOSMode Custom` mode. Usually relevant for Dell laptops.

6. `DisableIoMapper`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables `IOMapper` support in XNU (VT-d), which may conflict with the firmware implementation.

   Note: This option is a preferred alternative to ~~dropping~~ deleting `DMAR` ACPI table and disabling VT-d in firmware preferences, which does not break VT-d support in other systems in case they need it.

7. `DisableRtcChecksum`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables primary checksum (`0x58-0x59`) writing in AppleRTC.

   Note 1: This option will not protect other areas from being overwritten, see RTCMemoryFixup kernel extension if this is desired.

   Note 2: This option will not protect areas from being overwritten at firmware stage (e.g. macOS bootloader), see `AppleRtc` protocol description if this is desired.

8. `DummyPowerManagement`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables `AppleIntelCpuPowerManagement`.

   Note: This option is a preferred alternative to `NullCpuPowerManagement.kext` for CPUs without native power management driver in macOS.

9. `ExternalDiskIcons`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

   Note: This option should be avoided whenever possible. Modern firmwares usually have compatible AHCI controllers.

10. `IncreasePciBarSize`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Increases 32-bit PCI bar size in IOPCIFamily from 1 to 4 GBs.

    Note: This option should be avoided whenever possible. In general the necessity of this option means misconfigured or broken firmware.

11. `LapicKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Disables kernel panic on LAPIC interrupts.

12. `PanicNoKextDump`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

13. `PowerTimeoutKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Disables kernel panic on setPowerState timeout.

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 (`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\Microsoftdebian\Boot\bootmgfwgrubx64.efi` for ~~Microsoft~~ 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 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.~~

8.4 Debug Properties

1. `AppleDebug`
   Type: `plist boolean`
   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. `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.

   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

   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.

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.
   - `Secure` — require `vault.sig` signature file for `vault.plist` in `OC` directory. This includes `Basic` integrity checking but also attempts to build a trusted bootchain.

   `vault.plist` file should contain SHA-256 hashes for all files used by OpenCore. Presence of this file is highly recommended to ensure that unintentional file modifications (including filesystem corruption) do not happen unnoticed. To create this file automatically use `create_vault.sh` script. Regardless of the underlying filesystem, path name and case must match between `config.plist` and `vault.plist`.

   `vault.sig` file should contain a raw 256 byte RSA-2048 signature from SHA-256 hash of `vault.plist`. The signature is verified against the public key embedded into `OpenCore.efi`. To embed the public key you should do either of the following:

   - Provide public key during the `OpenCore.efi` compilation in `OpenCoreVault.c` file.
   - Binary patch `OpenCore.efi` replacing zeroes with the public key between `=BEGIN OC VAULT=` and `==END OC VAULT==` ASCII markers.

   RSA public key 520 byte format description can be found in Chromium OS documentation. To convert public key from X.509 certificate or from PEM file use RsaTool.

   The complete set of commands to:

   - Create `vault.plist`.
   - Create a new RSA key (always do this to avoid loading old configuration).
   - Embed RSA key into `OpenCore.efi`.
   - Create `vault.sig`.

   Can look as follows:

```
cd /Volumes/EFI/EFI/OC
/path/to/create_vault.sh .
/path/to/RsaTool -sign vault.plist vault.sig vault.pub
off=$(($(strings -a -t d OpenCore.efi | grep "=BEGIN OC VAULT=" | cut -f1 -d' ')+16))
dd of=OpenCore.efi if=vault.pub bs=1 seek=$off count=528 conv=notrunc
rm vault.pub
```

   Note 1: While it may appear obvious, but you have to use an external method to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this you are recommended to at least enable UEFI SecureBoot with a custom certificate, and sign `OpenCore.efi` and `BOOTx64.efi` with your custom key. More details on customising secure boot on modern firmwares can be found in Taming UEFI SecureBoot paper (in Russian).

   Note 2: `vault.plist` and `vault.sig` are used regardless of this option when `vault.plist` is present or public key is embedded into `OpenCore.efi`. Setting this option will only ensure configuration sanity, and abort the boot process otherwise.

9. `ScanPolicy`
   Type: `plist integer`, 32 bit
   Failsafe: ~~0xF0103~~0x10F0103
   Description: Define operating system detection policy.

This value allows to prevent scanning (and booting) from untrusted source based on a bitmask (sum) of select flags. As it is not possible to reliably detect every file system or device type, this feature cannot be fully relied upon in open environments, and the additional measures are to be applied.

Third party drivers may introduce additional security (and performance) measures following the provided scan policy. Scan policy is exposed in `scan-policy` variable of `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` GUID for UEFI Boot Services only.

- `0x00000001` (bit `0`) — `OC_SCAN_FILE_SYSTEM_LOCK`, restricts scanning to only known file systems defined as a part of this policy. File system drivers may not be aware of this policy, and to avoid mounting of undesired file systems it is best not to load its driver. This bit does not affect dmg mounting, which may have any file system. Known file systems are prefixed with `OC_SCAN_ALLOW_FS_`.
- `0x00000002` (bit `1`) — `OC_SCAN_DEVICE_LOCK`, restricts scanning to only known device types defined as a part of this policy. This is not always possible to detect protocol tunneling, so be aware that on some systems it may be possible for e.g. USB HDDs to be recognised as SATA. Cases like this must be reported. Known device types are prefixed with `OC_SCAN_ALLOW_DEVICE_`.
- `0x00000100` (bit `8`) — `OC_SCAN_ALLOW_FS_APFS`, allows scanning of APFS file system.
- `0x00000200` (bit `9`) — `OC_SCAN_ALLOW_FS_HFS`, allows scanning of HFS file system.
- `0x00000400` (bit `10`) — `OC_SCAN_ALLOW_FS_ESP`, allows scanning of EFI System Partition file system.
- `0x00000800` (bit `11`) — `OC_SCAN_ALLOW_FS_NTFS`, allows scanning of NTFS (Msft Basic Data) file system.
- `0x00001000` (bit `12`) — `OC_SCAN_ALLOW_FS_EXT`, allows scanning of EXT (Linux Root) file system.
- `0x00010000` (bit `16`) — `OC_SCAN_ALLOW_DEVICE_SATA`, allow scanning SATA devices.
- `0x00020000` (bit `17`) — `OC_SCAN_ALLOW_DEVICE_SASEX`, allow scanning SAS and Mac NVMe devices.
- `0x00040000` (bit `18`) — `OC_SCAN_ALLOW_DEVICE_SCSI`, allow scanning SCSI devices.
- `0x00080000` (bit `19`) — `OC_SCAN_ALLOW_DEVICE_NVME`, allow scanning NVMe devices.
- `0x00100000` (bit `20`) — `OC_SCAN_ALLOW_DEVICE_ATAPI`, allow scanning CD/DVD devices.
- `0x00200000` (bit `21`) — `OC_SCAN_ALLOW_DEVICE_USB`, allow scanning USB devices.
- `0x00400000` (bit `22`) — `OC_SCAN_ALLOW_DEVICE_FIREWIRE`, allow scanning FireWire devices.
- `0x00800000` (bit `23`) — `OC_SCAN_ALLOW_DEVICE_SDCARD`, allow scanning card reader devices.
- `0x01000000` (bit `24`) — `OC_SCAN_ALLOW_DEVICE_PCI`, allow scanning devices directly connected to PCI bus (e.g. VIRTIO).

Note: Given the above description, `0xF0103` value is expected to allow scanning of SATA, SAS, SCSI, and NVMe devices with APFS file system, and prevent scanning of any devices with HFS or FAT32 file systems in addition to not scanning APFS file systems on USB, CD, and FireWire drives. The combination reads as:

- `OC_SCAN_FILE_SYSTEM_LOCK`
- `OC_SCAN_DEVICE_LOCK`
- `OC_SCAN_ALLOW_FS_APFS`
- `OC_SCAN_ALLOW_DEVICE_SATA`
- `OC_SCAN_ALLOW_DEVICE_SASEX`
- `OC_SCAN_ALLOW_DEVICE_SCSI`
- `OC_SCAN_ALLOW_DEVICE_NVME`

8.6 Entry Properties

1. `Arguments`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used as boot arguments (load options) of the specified entry.

2. `Auxiliary`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This entry will not be listed by default when `HideAuxiliary` is set to `true`.

3. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

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~~deleted. I.e. to overwrite an existing variable value add the variable name to the ~~Block~~Delete 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~~Delete
   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~~Delete (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

Booter arguments, similar to `boot-args` but for `boot.efi`. Accepts a set of arguments, which are hexadecimal 64-bit values with or without `0x`. At different stages `boot.efi` will request different debugging (logging) modes (e.g. after `ExitBootServices` it will only print to serial). Several booter arguments control whether these requests will succeed. The list of known requests is covered below:

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

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

- `boot-save-log=VALUE` — debug log save mode for normal boot.
  - `0`
  - `1`
  - `2` — (default).
  - `3`
  - `4` — (save to file).
- `wake-save-log=VALUE` — debug log save mode for hibernation wake.
  - `0` — disabled.
  - `1`
  - `2` — (default).
  - `3` — (unavailable).
  - `4` — (save to file, unavailable).
- `breakpoint=VALUE` — enables debug breaks (missing in production `boot.efi`).
  - `0` — disables debug breaks on errors (default).
  - `1` — enables debug breaks on errors.
- `console=VALUE` — enables console logging.
  - `0` — disables console logging.
  - `1` — enables console logging when debug protocol is missing (default).
  - `2` — enables console logging unconditionally (unavailable).
- `embed-log-dt=VALUE` — enables DeviceTree logging.
  - `0` — disables DeviceTree logging (default).
  - `1` — enables DeviceTree logging.
- `kc-read-size=VALUE` — Chunk size used for buffered I/O from network or disk for prelinkedkernel reading and related. Set to 1MB (0x100000) by default, can be tuned for faster booting.
- `log-level=VALUE` — log level bitmask.
  - `0x01` — enables trace logging (default).
- `serial=VALUE` — enables serial logging.
  - `0` — disables serial logging (default).
  - `1` — enables serial logging for `EXITBS:END` onwards.
  - `1` — enables serial logging for `EXITBS:START` onwards.
  - `3` — enables serial logging when debug protocol is missing.
  - `4` — enables serial logging unconditionally.
- `timestamps=VALUE` — enables timestamp logging.
  - `0` — disables timestamp logging.
  - `1` — enables timestamp logging (default).
- `log=VALUE` — deprecated starting from 10.15.
  - `1` — AppleLoggingConOutOrErrSet/AppleLoggingConOutOrErrPrint (classical ConOut/StdErr)

10 PlatformInfo

Platform information is comprised of several identification fields generated or filled manually to be compatible with macOS services. The base part of the configuration may be obtained from ~~package~~`AppleModels`, which itself generates a set of interfaces based on a database in YAML format. These fields are written to three select destinations:

- SMBIOS
- Data Hub
- NVRAM

Most of the fields specify the overrides in SMBIOS, and their field names conform to EDK2 SmBios.h header file. However, several important fields reside in Data Hub and NVRAM. Some of the values can be found in more than one field and/or destination, so there are two ways to control their update process: manual, where one specifies all the values (the default), and semi-automatic, where (`Automatic`) only select values are specified, and later used for system configuration.

To inspect SMBIOS contents dmidecode utility can be used. Version with macOS specific enhancements can be downloaded from Acidanthera/dmidecode.

10.1 Properties

1. `Automatic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Generate PlatformInfo based on `Generic` section instead of using values from `DataHub`, `NVRAM`, and `SMBIOS` sections.

   Enabling this option is useful when `Generic` section is flexible enough~~.~~:

   - When enabled `SMBIOS`, `DataHub`, and `PlatformNVRAM` data is unused.
   - When disabled `Generic` section is unused.

2. `UpdateDataHub`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update Data Hub fields. These fields are read from `Generic` or `DataHub` sections depending on `Automatic` value.

3. `UpdateNVRAM`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update NVRAM fields related to platform information.

   These fields are read from `Generic` or `PlatformNVRAM` sections depending on `Automatic` value. All the other fields are to be specified with `NVRAM` section.

   If `UpdateNVRAM` is set to `false` the aforementioned variables can be updated with `NVRAM` section. If `UpdateNVRAM` is set to `true` the behaviour is undefined when any of the fields are present in `NVRAM` section.

4. `UpdateSMBIOS`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update SMBIOS fields. These fields are read from `Generic` or `SMBIOS` sections depending on `Automatic` value.

5. `UpdateSMBIOSMode`
   Type: `plist string`
   Failsafe: `Create`
   Description: Update SMBIOS fields approach:

   - `TryOverwrite` — `Overwrite` if new size is <= than the page-aligned original and there are no issues with legacy region unlock. `Create` otherwise. Has issues with some firmwares.
   - `Create` — Replace the tables with newly allocated EfiReservedMemoryType at AllocateMaxAddress without any fallbacks.

- `Overwrite` — Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
- `Custom` — Write SMBIOS tables (`gEfiSmbios(3)TableGuid`) to `gOcCustomSmbios(3)TableGuid` to workaround firmwares overwriting SMBIOS contents at ExitBootServices. Otherwise equivalent to `Create`. Requires patching AppleSmbios.kext and AppleACPIPlatform.kext to read from another GUID: `"EB9D2D31"` - `"EB9D2D35"` (in ASCII), done automatically by `CustomSMBIOSGuid` quirk.

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

6. `Generic`
   Type: `plist dictonary`
   Optional: When `Automatic` is `false`
   Description: Update all fields. This section is read only when `Automatic` is active.

7. `DataHub`
   Type: `plist dictonary`
   Optional: When `Automatic` is `true`
   Description: Update Data Hub fields. This section is read only when `Automatic` is not active.

8. `PlatformNVRAM`
   Type: `plist dictonary`
   Optional: When `Automatic` is `true`
   Description: Update platform NVRAM fields. This section is read only when `Automatic` is not active.

9. `SMBIOS`
   Type: `plist dictonary`
   Optional: When `Automatic` is `true`
   Description: Update SMBIOS fields. This section is read only when `Automatic` is not active.

10.2 Generic Properties

1. `SpoofVendor`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Sets SMBIOS vendor fields to `Acidanthera`.

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

2. `AdviseWindows`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Forces Windows support in `FirmwareFeatures`.

   Added bits to `FirmwareFeatures`:

   - `FW_FEATURE_SUPPORTS_CSM_LEGACY_MODE` (`0x1`) - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being not the first partition on the disk.
   - `FW_FEATURE_SUPPORTS_UEFI_WINDOWS_BOOT` (`0x20000000`) - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being the first partition on the disk.

3. `SystemProductName`
   Type: `plist string`
   Failsafe: `MacPro6,1`
   Description: Refer to SMBIOS `SystemProductName`.

4. `SystemSerialNumber`
   Type: `plist string`
   Failsafe: `OPENCORE_SN1`
   Description: Refer to SMBIOS `SystemSerialNumber`.

10.4 PlatformNVRAM Properties

1. `BID`
   Type: `plist string`
   Failsafe: Not installed
   Description: Specifies the value of NVRAM variable `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID`.

2. `ROM`
   Type: `plist data`, 6 bytes
   Failsafe: Not installed
   Description: Specifies the values of NVRAM variables `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM` and `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM`.

3. `MLB`
   Type: `plist string`
   Failsafe: Not installed
   Description: Specifies the values of NVRAM variables `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB` and `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:MLB`.

4. `FirmwareFeatures`
   Type: `plist data`, 8 bytes
   Failsafe: Not installed
   Description: This variable comes in pair with `FirmwareFeaturesMask`. Specifies the values of NVRAM variables:

   - `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeatures`
   - `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeatures`

5. `FirmwareFeaturesMask`
   Type: `plist data`, 8 bytes
   Failsafe: Not installed
   Description: This variable comes in pair with `FirmwareFeatures`. Specifies the values of NVRAM variables:

   - `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeaturesMask`
   - `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask`

10.5 SMBIOS Properties

1. `BIOSVendor`
   Type: `plist string`
   Failsafe: OEM specified
   SMBIOS: BIOS Information (Type 0) — Vendor
   Description: BIOS Vendor. All rules of `SystemManufacturer` do apply.

2. `BIOSVersion`
   Type: `plist string`
   Failsafe: OEM specified
   SMBIOS: BIOS Information (Type 0) — BIOS Version
   Description: Firmware version. This value gets updated and takes part in update delivery configuration and macOS version compatibility. This value could look like `MM71.88Z.0234.B00.1809171422` in older firmwares, and is described in BiosId.h. In newer firmwares it should look like `236.0.0.0.0` or `220.230.16.0.0 (iBridge: 16.16.2542.0.0,0)`. iBridge version is read from `BridgeOSVersion` variable, and is only present on macs with T2.

```
Apple ROM Version
 BIOS ID:      MBP151.88Z.F000.B00.1811142212
 Model:        MBP151
 EFI Version:  220.230.16.0.0
 Built by:     root@quinoa
 Date:         Wed Nov 14 22:12:53 2018
 Revision:     220.230.16 (B&I)
 ROM Version:  F000_B00
 Build Type:   Official Build, RELEASE
```

```
Compiler:     Apple LLVM version 10.0.0 (clang-1000.2.42)
UUID:         E5D1475B-29FF-32BA-8552-682622BA42E1
UUID:         151B0907-10F9-3271-87CD-4BF5DBECACF5
```

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

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

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

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

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

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

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

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

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

21. `ChassisSerialNumber`
    Type: `plist string`
    Failsafe: OEM specified
    SMBIOS: System Enclosure or Chassis (Type 3) — Version
    Description: Should match `SystemSerialNumber`.

22. `ChassisAssetTag`
    Type: `plist string`
    Failsafe: OEM specified
    SMBIOS: System Enclosure or Chassis (Type 3) — Asset Tag Number
    Description: Chassis type name. Varies, could be empty or `MacBook-Aluminum`.

23. `PlatformFeature`
    Type: `plist integer`, 32-bit
    Failsafe: `0xFFFFFFFF`
    SMBIOS: `APPLE_SMBIOS_TABLE_TYPE133` - `PlatformFeature`
    Description: Platform features bitmask. Refer to AppleFeatures.h for more details. Use `0xFFFFFFFF` value to not provide this table.

24. `SmcVersion`
    Type: `plist data`, 16 bytes
    Failsafe: All zero
    SMBIOS: `APPLE_SMBIOS_TABLE_TYPE134` - `Version`
    Description: ASCII string containing SMC version in upper case. Missing on T2 based Macs. Ignored when zero.

25. `FirmwareFeatures`
    Type: `plist data`, 8 bytes
    Failsafe: `0`
    SMBIOS: `APPLE_SMBIOS_TABLE_TYPE128` - `FirmwareFeatures` and `ExtendedFirmwareFeatures`
    Description: 64-bit firmware features bitmask. Refer to AppleFeatures.h for more details. Lower 32 bits match `FirmwareFeatures`. Upper 64 bits match `ExtendedFirmwareFeatures`.

26. `FirmwareFeaturesMask`
    Type: `plist data`, 8 bytes
    Failsafe: `0`
    SMBIOS: `APPLE_SMBIOS_TABLE_TYPE128` - `FirmwareFeaturesMask` and `ExtendedFirmwareFeaturesMask`
    Description: Supported bits of extended firmware features bitmask. Refer to AppleFeatures.h for more details. Lower 32 bits match `FirmwareFeaturesMask`. Upper 64 bits match `ExtendedFirmwareFeaturesMask`.

27. `ProcessorType`
    Type: `plist integer`, 16-bit
    Failsafe: Automatic
    SMBIOS: `APPLE_SMBIOS_TABLE_TYPE131` - `ProcessorType`
    Description: Combined of Processor Major and Minor types.

28. `MemoryFormFactor`
    Type: `plist integer`, 8-bit
    Failsafe: OEM specified
    SMBIOS: Memory Device (Type 17) — Form Factor
    Description: Memory form factor. On Macs it should be DIMM or SODIMM.

11 UEFI

11.1 Introduction

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

11.2 Drivers

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

- `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`
   Failsafe: None
   Description: Configure audio backend support described in Audio Properties section below.

   Audio support provides a way for upstream protocols to interact with the selected hardware and audio resources. All audio resources should reside in `\EFI\OC\Resources\Audio` directory. Currently the only supported audio

file format is WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

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

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

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

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

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

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

4. `Drivers`
   Type: `plist array`
   Failsafe: None
   Description: Load selected drivers from `OC/Drivers` directory.

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

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

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

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

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

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

9. `ReservedMemory`
   Type: `plist array`

11.8 Audio Properties

1. `AudioCodec`
   Type: `plist integer`
   Failsafe: `0`
   Description: Codec address on the specified audio controller for audio support.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6. `PlayChime`
   Type: `plist boolean`

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.

11.11 ProtocolOverrides Properties

1. `AppleAudio`
   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~~ delete 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~~ 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. ~~`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.

8. `RequestBootVarRouting`
   Type: `plist boolean`
   Failsafe: `false`

Description: Request redirect of all `Boot` prefixed variables from `EFI_GLOBAL_VARIABLE_GUID` to `OC_VENDOR_VARIABLE_GUID`.

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

9. `TscSyncTimeout`
   Type: `plist integer`
   Failsafe: `0`
   Description: Attempts to perform TSC synchronisation with a specified timeout.

   The primary purpose of this quirk is to enable early bootstrap TSC synchronisation on some server and laptop models when running a debug XNU kernel. For the debug kernel the TSC needs to be kept in sync across the cores before any kext could kick in rendering all other solutions problematic. The timeout is specified in microseconds and depends on the amount of cores present on the platform, the recommended starting value is `500000`.

   This is an experimental quirk, which should only be used for the aforementioned problem. In all other cases the quirk may render the operating system unstable and is not recommended. The recommended solution in the other cases is to install a kernel driver like VoodooTSCSync, TSCAdjustReset, or CpuTscSync (a more specialised variant of VoodooTSCSync for newer laptops).

   Note: The reason this quirk cannot replace the kernel driver is because it cannot operate in ACPI S3 mode (sleep wake) and because the UEFI firmwares provide very limited multicore support preventing the precise update of the MSR registers.

10. `UnblockFsConnect`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Some firmwares block partition handles by opening them in By Driver mode, which results in File System protocols being unable to install.

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

11.13 ReservedMemory Properties

1. `Address`
   Type: `plist integer`
   Failsafe: `0`
   Description: Start address of the reserved memory region, which should be allocated as reserved effectively marking the memory of this type inaccessible to the operating system.

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

2. `Comment`
   Type: `plist string`
   Failsafe: Empty string
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

3. `Size`
   Type: `plist integer`
   Failsafe: `0`
   Description: Size of the reserved memory region, must be page-aligned (4 KBs).

4. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This region will not be reserved unless set to `true`.

12 Troubleshooting

12.1 Windows support

Can I install Windows?

While no official Windows support is provided, 64-bit UEFI Windows installations (Windows 8 and above) prepared with Boot Camp are supposed to work. Third-party UEFI installations as well as systems partially supporting UEFI boot, like Windows 7, might work with some extra precautions. Things to keep in mind:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- ~~To install Windows, macOS, and OpenCore on the same drive you can specify Windows bootloader path (\EFI\Microsoft\Boot\bootmgfw.efi) in BlessOverride section.~~
- All the modifications applied (to ACPI, NVRAM, SMBIOS, etc.) are supposed to be operating system agnostic, i.e. apply equally regardless of the OS booted. This enables Boot Camp software experience on Windows.
- macOS requires the first partition to be EFI System Partition, and does not support the default Windows layout. While OpenCore does have a workaround for this, it is highly recommend not to rely on it and install properly.
- Windows may need to be reactivated. To avoid it consider setting SystemUUID to the original firmware UUID. Be warned, on old firmwares it may be invalid, i.e. not random. In case you still have issues, consider using HWID or KMS38 license or making the use `Custom UpdateSMBIOSMode`. Other nuances of Windows activation are out of the scope of this document and can be found online.

What additional software do I need?

To enable operating system switching and install relevant drivers in the majority of cases you will need Windows support software from Boot Camp. For simplicity of the download process or when configuring an already installed Windows version a third-party utility, Brigadier, can be used successfully. Note, that you may have to download and install 7-Zip prior to using Brigadier.

Remember to always use the latest version of Windows support software from Boot Camp, as versions prior to 6.1 do not support APFS, and thus will not function correctly. To download newest software pass most recent Mac model to Brigadier, for example `./brigadier.exe -m iMac19,1`. To install Boot Camp on an unsupported Mac model afterwards run PowerShell as Administrator and enter `msiexec /i BootCamp.msi`. In case you already have a previous version of Boot Camp installed you will have to remove it first by running `msiexec /x BootCamp.msi` command. `BootCamp.msi` file is located in `BootCamp/Drivers/Apple` directory and can be reached through Windows Explorer.

While Windows support software from Boot Camp solves most of compatibility problems, sometimes you may have to address some of them manually:

- To invert mouse wheel scroll direction `FlipFlopWheel` must be set to `1` as explained on SuperUser.
- `RealTimeIsUniversal` must be set to `1` to avoid time desync between Windows and macOS as explained on SuperUser (this one is usually not needed).
- To access Apple filesystems like HFS and APFS separate software may need to be installed. Some of the known ~~tools~~ utilities are: Apple HFS+ driver (hack for Windows 10), HFSExplorer, MacDrive, Paragon APFS, Paragon HFS+, TransMac, etc. Remember to never ever attempt to modify Apple file systems from Windows as this often leads to irrecoverable data loss.

Why do I see `Basic data partition` in Boot Camp Startup Disk control panel?

Boot Camp control panel uses GPT partition table to obtain each boot option name. After installing Windows separately you will have to relabel the partition manually. This can be done with many ~~tools~~ utilities including open-source gdisk utility. Reference example:

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

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

- 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 debug macOS boot failure?**
   - Refer to boot-args values like debug=0x100, keepsyms=1, -v, and similar.
   - Do not forget about AppleDebug and ApplePanic properties.
   - Take care of Booter, Kernel, and UEFI quirks.
   - Consider using serial port to inspect early kernel boot failures. For this you may need debug=0x108, serial=5, and msgbuf=1048576 arguments. Refer to the patches in Sample.plist when dying before serial init.
   - Always read the logs carefully.
3. **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.
4. **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.
5. **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 ~~toolfrom~~ macrecovery.py, builtin tool.

   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 utilities to download an offline image.
6. **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.
7. **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~~ on MacRumors.com.
8. **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 or in the ACPI section of this document.
9. **How can I ~~migrate from AptioMemoryFix~~decide which Booter quirks to use?**

   ~~Behaviour similar to that of~~ These quirks originate from 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.~~ driver but provide a wider set of changes specific to modern systems. Note, that `OpenRuntime` driver is required for most configurations. To get a configuration similar to `AptioMemoryFix` you may try enabling the following set of quirks:

- `ProvideConsoleGop` (UEFI quirk)
- `AvoidRuntimeDefrag`
- `DiscardHibernateMap`
- `EnableSafeModeSlide`
- `EnableWriteUnprotector`
- `ForceExitBootServices`
- `ProtectMemoryRegions`
- `ProvideCustomSlide`
- ~~`SetupVirtualMap`~~`RebuildAppleMemoryMap`
- ~~`ShrinkMemoryMap`~~`SetupVirtualMap`

However, as of today such set is strongly discouraged as some of these quirks are not necessary to be enabled or need additional quirks. For example, `DevirtualiseMmio` and `ProtectUefiServices` are often required, while `DiscardHibernateMap` and `ForceExitBootServices` are rarely necessary.

Unfortunately for some quirks like `RebuildAppleMemoryMap`, `EnableWriteUnprotector`, `ProtectMemoryRegions`, `RebuildAppleMemoryMap`, `SetupVirtualMap`, and `SyncRuntimePermissions` there is no definite approach even on similar systems, so trying all their combinations may be required for optimal setup. Refer to individual quirk descriptions in this document for more details.