# OpenCore

Reference Manual (0.6.~~1~~.2)

[2020.10.04]

# 1 Introduction

This document provides information on OpenCore user configuration file format used to setup the correct functioning of the 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 to be documentation or implementation bugs ~~, and are requested to be reported through~~ which should be reported via the Acidanthera Bugtracker. ~~Errata~~ An errata sheet is available in OpenCorePkg repository.

This document is structured as a specification ~~,~~ and is not meant to provide a ~~step by step algorithm for configuring~~ step-by-step guide to configuring an end-user ~~board support package~~ Board Support Package (BSP). The intended audience of the document ~~are~~ is anticipated to be programmers and engineers with a basic understanding of macOS internals and UEFI ~~functioning~~functionality. For these reasons, this document is available exclusively in English, and all other sources or translations of this document are unofficial and may contain errors.

Third-party articles, utilities, books, and alike, may be more useful for a wider audience as they could provide guide-like material. However, they are ~~prone~~ subject to their authors' preferences, tastes, ~~this document misinterpretation, and essential~~ misinterpretations of this document, and unavoidable obsolescence. In ~~case you use these sources, for example,~~ cases of using such sources, such as Dortania's OpenCore Install Guide and related material, please ~~ensure to follow this document for every made decision and judge its~~ refer back to this document on every decision made and re-evaluate potential consequences.

~~Be warned~~ Please note that regardless of the sources used~~you~~, users are required to fully understand every ~~dedicated~~ OpenCore configuration option~~and concept prior to reporting any issues in~~, and the principles behind them, before posting issues to the 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.

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`.
- `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. `OpenDuetPkg` is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system~~you can install~~ `OpenDuetPkg` can be installed with a dedicated tool — BootInstall (bundled with OpenCore). Third-party utilities can be used to perform this on systems ~~different from~~ other than 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 is hosted in acidanthera/audk. The required patches for the package are present in `Patches` directory.

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

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

```
git clone --depth=1 https://github.com/acidanthera/audk UDK
cd UDK
git submodule update --init --recommend-shallow
git clone --depth=1 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~~ the UDK root:

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/MdeModulePkg
-I/UefiPackages/MdeModulePkg/Include
-I/UefiPackages/MdeModulePkg/Include/X64
-I/UefiPackages/OpenCorePkg/Include/AMI
-I/UefiPackages/OpenCorePkg/Include/Acidanthera
-I/UefiPackages/OpenCorePkg/Include/Apple
-I/UefiPackages/OpenCorePkg/Include/Apple/X64
-I/UefiPackages/OpenCorePkg/Include/Duet
-I/UefiPackages/OpenCorePkg/Include/Generic
-I/UefiPackages/OpenCorePkg/Include/Intel
-I/UefiPackages/OpenCorePkg/Include/Microsoft
-I/UefiPackages/OpenCorePkg/Include/VMware
-I/UefiPackages/OvmfPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
-DNO_MSABI_VA_FUNCS=1
```

Listing 2: ECC Configuration

**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~~ the patch being rejected.

**Organisation**. The codebase is contained in `OpenCorePkg` 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.

**Failsafe**: All zero
**Description**: Match table signature to be equal to this value unless all zero.

In the majority of the cases ACPI patches are not useful and harmful:

- Avoid renaming devices with ACPI patches. This may fail or perform improper renaming of unrelated devices (e.g. `EC` and `EC0`), be unnecessary, or even fail to rename devices in select tables. For ACPI consistency it is much safer to rename devices at I/O Registry level, as done by WhateverGreen.
- ~~Avoid~~ Try to avoid patching `_OSI` to support a higher level of feature sets ~~unless absolutely required~~whenever possible. Commonly this enables a number of hacks on APTIO firmwares, which result in the need to add more patches. Modern firmwares generally do not need it at all, and those that do are fine with much smaller patches. However, laptop vendors usually rely on this method to determine the availability of functions like modern I2C input support, thermal adjustment and custom feature additions.
- Avoid patching embedded controller event `_Qxx` just for enabling brightness keys. The conventional process to find these keys usually involves massive modification on DSDT and SSDTs and the debug kext is not stable on newer systems. Please switch to built-in brightness key discovery of BrightnessKeys instead.
- 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~~inject shutdown fix on certain computers, the original method can be replaced with a dummy name by patching `_~~Q11~~PTS` with `~~XQ11~~ZPTS` and adding a callback to original method.

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~~ the system unstable due to area relocation. If ~~you need such changes you may utilise~~ such changes are needed, the utilisation of "proxy" patching or the padding of `NOP` to the remaining area might be taken into account.

## 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.

# 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~~ this is used 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~~, the following requirements should be fulfilled:

- Most up-to-date UEFI firmware (check ~~your~~ the 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~~ this option should be checked first whenever erratic boot failures are encountered.
- `DisableIoMapper` quirk enabled, or `VT-d` disabled in firmware settings if present, or ACPI DMAR table deleted.
- **No** 'slide' boot argument present in NVRAM or anywhere else. It is not necessary unless ~~you cannot boot~~ the system cannot be booted at all or ~~see~~ `No slide values are usable! Use custom slide!` message can be seen in the log.
- `CFG Lock` (MSR `0xE2` write protection) disabled in firmware settings if present. Consider patching it if ~~you have enough skills and~~ no option is available (for advanced users only). See VerifyMsrE2 notes for more details.
- `CSM` (Compatibility Support Module) disabled in firmware settings if present. ~~You may need to flash GOP ROM on~~ On NVIDIA 6xx/AMD 2xx or older, GOP ROM may have to be flashed first. Use GopUpdate (see the second post) or AMD UEFI GOP MAKER in case ~~you are not sure how~~of any potential confusion.
- `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` may have to be disabled in firmware settings present.

When debugging sleep issues ~~you may want to (temporarily) disable~~ Power Nap and automatic power off may be (temporarily) disabled, 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~~ ACPI tables should be looked up 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 `DevirtualiseMmio`. This means that the firmware will be able to directly communicate with this memory region during operating system functioning, because the region this value is in will be assigned a virtual address.

The addresses written here must be part of the memory map, have `EfiMemoryMappedIO` type and `EFI_MEMORY_RUNTIME` attribute (highest bit) set. To find the list of the candidates the debug log can be used.

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 address will be devirtualised unless set to `true`.

## 5.4 Quirks Properties

1. `AvoidRuntimeDefrag`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Protect from boot.efi runtime memory defragmentation.

This option fixes UEFI runtime services (date, time, NVRAM, power control, etc.) support on many firmwares using SMM backing for select services like variable storage. SMM may try to access physical addresses, but they get moved by boot.efi.

*Note*: Most but Apple and VMware firmwares need this quirk.

2. `DevirtualiseMmio`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Remove runtime attribute from select MMIO regions.

This option reduces stolen memory footprint from the memory map by removing runtime bit for known memory regions. This quirk may result in the increase of KASLR slides available, but is not necessarily compatible with the target board without additional measures. In general this frees from 64 to 256 megabytes of memory (present in the debug log), and on some platforms it is the only way to boot macOS, which otherwise fails with allocation error at bootloader stage.

This option is generally useful on all firmwares except some very old ones, like Sandy Bridge. On select firmwares it may require a list of exceptional addresses that still need to get their virtual addresses for proper NVRAM and hibernation functioning. Use `MmioWhitelist` section to do this.

3. `DisableSingleUser`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Disable single user mode.

This is a security option ~~allowing one to restrict~~ that restricts the activation of single user mode ~~usage~~ by ignoring `CMD+S` hotkey and `-s` boot argument. The behaviour with this quirk enabled is supposed to match T2-based model behaviour. ~~Read~~ Refer to this archived article to understand how to use single user mode with this quirk enabled.

4. `DisableVariableWrite`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Protect from macOS NVRAM write access.

This is a security option ~~allowing one to restrict~~ that restricts NVRAM access in macOS. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `OpenRuntime.efi`.

*Note*: This quirk can also be used as an ugly workaround to buggy UEFI runtime services implementations that fail to write variables to NVRAM and break the rest of the operating system.

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

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

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

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

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

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

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

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

   *Note*: This quirk may potentially weaken firmware security, please use `RebuildAppleMemoryMap` if ~~your~~ the firmware supports memory attributes table (MAT). Refer to `OCABC: MAT support is 1/0` log entry to determine whether MAT is supported.

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

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

   *Note*: The necessity of this quirk is determined by early boot crashes of the firmware. Do not use this ~~unless you fully understand~~ without a full understanding of the consequences.

9. `ProtectMemoryRegions`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Protect memory regions from incorrect access.

   Some firmwares incorrectly map select memory regions:

   - CSM region can be marked as boot services code or data, which leaves it as free memory for XNU kernel.
   - MMIO regions can be marked as reserved memory and stay unmapped, but may be required to be accessible at runtime for NVRAM support.

   This quirk attempts to fix types of these regions, e.g. ACPI NVS for CSM or MMIO for MMIO.

*Note*: The necessity of this quirk is determined by artifacts, sleep wake issues, and boot failures. In general only very old firmwares need this quirk.

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

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

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

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

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

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

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

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

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

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

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

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

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

    Apple kernel has several limitations in parsing UEFI memory map:

    - Memory map size must not exceed 4096 bytes as Apple kernel maps it as a single 4K page. Since some firmwares have very large memory maps (approximately over 100 entries) Apple kernel will crash at boot.

# 7 Kernel

## 7.1 Introduction

This section allows to apply different kinds of kernelspace modifications on Apple Kernel (XNU). The modifications currently provide driver (kext) injection, kernel and driver patching, and driver blocking.

## 7.2 Properties

1. `Add`
   **Type**: `plist array`
   **Failsafe**: Empty
   **Description**: Load selected kernel drivers from `OC/Kexts` directory.

   Designed to be filled with `plist dict` values, describing each driver. See Add Properties section below. Kernel driver load order follows the item order in the array, thus the dependencies should be written prior to their consumers.

   To track the dependency order~~one can~~, inspect the `OSBundleLibraries` key in the `Info.plist` of the kext. Any kext mentioned in the `OSBundleLibraries` of the other kext must ~~be~~ precede this kext.

   *Note*: Kexts may have inner kexts (`Plug-Ins`) in their bundle. Each inner kext must be added separately.

2. `Block`
   **Type**: `plist array`
   **Failsafe**: Empty
   **Description**: Remove selected kernel drivers from prelinked kernel.

   Designed to be filled with `plist dictionary` values, describing each blocked driver. See Block Properties section below.

3. `Emulate`
   **Type**: `plist dict`
   **Description**: Emulate select hardware in kernelspace via parameters described in Emulate Properties section below.

4. `Force`
   **Type**: `plist array`
   **Failsafe**: Empty
   **Description**: Load kernel drivers from system volume if they are not cached.

   Designed to be filled with `plist dict` values, describing each driver. See Force Properties section below. This section resolves the problem of injecting drivers that depend on other drivers, which are not cached otherwise. The issue normally affects older operating systems, where various dependency kexts, like `IOAudioFamily` or `IONetworkingFamily` may not be present in the kernel cache by default. Kernel driver load order follows the item order in the array, thus the dependencies should be written prior to their consumers. `Force` happens before `Add`.

   *Note*: The signature of the "forced" kernel drivers is not checked anyhow, making the use of this feature extremely dangerous and undesired for secure boot. This feature may not work on encrypted partitions in newer operating systems.

5. `Patch`
   **Type**: `plist array`
   **Failsafe**: Empty
   **Description**: Perform binary patches in kernel and drivers prior to driver addition and removal.

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

6. `Quirks`
   **Type**: `plist dict`
   **Description**: Apply individual kernel and driver quirks described in Quirks Properties section below.

**Failsafe**: Empty string
**Description**: Adds kernel driver on specified macOS version or newer.

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

8. `PlistPath`
**Type**: `plist string`
**Failsafe**: Empty string
**Description**: Kext `Info.plist` path relative to bundle (e.g. `Contents/Info.plist`).

## 7.4 Block Properties

1. `Arch`
**Type**: `plist string`
**Failsafe**: `Any`
**Description**: Kext block architecture (`Any`, `i386`, `x86_64`).

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 kernel driver will not be blocked unless set to `true`.

4. `Identifier`
**Type**: `plist string`
**Failsafe**: Empty string
**Description**: Kext bundle identifier (e.g. `com.apple.driver.AppleTyMCEDriver`).

5. `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.

6. `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~~ primarily serves for three needs:

- Enabling support of an unsupported CPU model ~~.~~ (e.g. Intel Pentium).
- Enabling support of a CPU model that is not yet supported by a specific version of macOS which usually is old.
- Enabling XCPM support for an unsupported CPU variant.

*Note 1*: It may also be the case that the CPU model is supported but there is no power management supported (e.g. virtual machines). In this case, `MinKernel` and `MaxKernel` can be set to restrict CPU virtualisation and dummy power management patches to the particular macOS kernel version.

*Note 2*: 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, `C3 06 03 00` stands for CPUID `0x0306C3` (Haswell).

*Note 3*: For XCPM support it is recommended to use the following combinations.

- Haswell-E (`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`

~~Keep in mind,~~ *Note 4*: Note that the following configurations are unsupported by XCPM (at least out of the box):

- Consumer Ivy Bridge (`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~~ should manually be patched to enforce XCPM on these CPUs instead of ~~using~~ this option.
- Low-end CPUs (e.g. Haswell+ 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`.

3. `DummyPowerManagement`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Requirement**: 10.4
   **Description**: Disables `AppleIntelCpuPowerManagement`.

   *Note 1*: This option is a preferred alternative to `NullCpuPowerManagement.kext` for CPUs without native power management driver in macOS.

   *Note 2*: While this option is usually needed to disable `AppleIntelCpuPowerManagement` on unsupported platforms, it can also be used to disable this kext in other situations (e.g. with `Cpuid1Data` left blank).

4. `MaxKernel`
   **Type**: `plist string`
   **Failsafe**: Empty string
   **Description**: Emulates CPUID and applies `DummyPowerManagement` on specified macOS version or older.

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

5. `MinKernel`
   **Type**: `plist string`
   **Failsafe**: Empty string
   **Description**: Emulates CPUID and applies `DummyPowerManagement` on specified macOS version or newer.

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

## 7.6 Force Properties

1. `Arch`
   **Type**: `plist string`
   **Failsafe**: `Any`
   **Description**: Kext architecture (`Any`, `i386`, `x86_64`).

2. `BundlePath`
   **Type**: `plist string`

4. `Count`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Number of patch occurrences to apply. `0` applies the patch to all occurrences found.

5. `Enabled`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: This kernel patch will not be used unless set to `true`.

6. `Find`
   **Type**: `plist data`
   **Failsafe**: Empty data
   **Description**: Data to find. Can be set to empty for immediate replacement at `Base`. Must equal to `Replace` in size otherwise.

7. `Identifier`
   **Type**: `plist string`
   **Failsafe**: Empty string
   **Description**: Kext bundle identifier (e.g. `com.apple.driver.AppleHDA`) or `kernel` for kernel patch.

8. `Limit`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Maximum number of bytes to search for. Can be set to `0` to look through the whole kext or kernel.

9. `Mask`
   **Type**: `plist data`
   **Failsafe**: Empty data
   **Description**: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Can be set to empty data to be ignored. Must equal to `Replace` in size otherwise.

10. `MaxKernel`
    **Type**: `plist string`
    **Failsafe**: Empty string
    **Description**: Patches data on specified macOS version or older.

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

11. `MinKernel`
    **Type**: `plist string`
    **Failsafe**: Empty string
    **Description**: Patches data on specified macOS version or newer.

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

12. `Replace`
    **Type**: `plist data`
    **Failsafe**: Empty data
    **Description**: Replacement data of one or more bytes.

13. `ReplaceMask`
    **Type**: `plist data`
    **Failsafe**: Empty data
    **Description**: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Can be set to empty data to be ignored. Must equal to `Replace` in size otherwise.

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

## 7.8 Quirks Properties

1. `AppleCpuPmCfgLock`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Requirement**: ~~10.6 (64-bit)~~10.4
   **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~~ The bundled `VerifyMsrE2` tool ~~. Select firmwares~~ can be used to check its state. Some firmware have this register locked only 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~~ the firmware image in UEFITool and find `CFG Lock` unicode string. If it is not present, ~~your~~ the firmware may not have this option and ~~you should stop~~the proccess should therefore be discontinued.
   (c) Extract the `Setup.bin` PE32 Image Section (the ~~one~~ UEFITool found) through the `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~~ the 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`
   **Requirement**: 10.8 (not required for older)
   **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`
   **Requirement**: 10.8 (not required for older)
   **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-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`
   **Requirement**: 10.8 (not required for older)
   **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`
   **Requirement**: ~~10.6 (64-bit)~~10.4
   **Description**: Performs GUID patching for `UpdateSMBIOSMode Custom` mode. Usually relevant for Dell laptops.

6. `DisableIoMapper`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Requirement**: 10.8 (not required for older)
   **Description**: Disables `IOMapper` support in XNU (VT-d), which may conflict with the firmware implementation.

   *Note*: This option is a preferred alternative to 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. `DisableLinkeditJettison`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Requirement**: 11.0
   **Description**: Disables `__LINKEDIT` jettison code.

   This option lets `Lilu.kext` and possibly some others function in macOS Big Sur with best performance without `keepsyms=1` boot argument.

8. `DisableRtcChecksum`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Requirement**: ~~10.6 (64-bit)~~10.4
   **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~~AppleRtcRam protocol description if this is desired.

9. ~~DummyPowerManagement~~ExtendBTFeatureFlags
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Requirement**: ~~10.6 (64-bit)~~10.8
   **Description**: ~~Disables~~ Set ~~AppleIntelCpuPowerManagement~~FeatureFlags to `0x0F` for full functionality of Bluetooth, including Continuity.

   *Note*: This option is a ~~preferred alternative to `NullCpuPowerManagement.kext` for CPUs without native power management driver in macOS.~~substitution for BT4LEContinuityFixup.kext, which does not function properly due to late patching progress.

10. `ExternalDiskIcons`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: ~~10.6 (64-bit)~~10.4
    **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.

11. `IncreasePciBarSize`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: 10.10
    **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.

12. `LapicKernelPanic`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: 10.6 (64-bit)
    **Description**: Disables kernel panic on LAPIC interrupts.

13. `LegacyCommpage`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: 10.4 - 10.6
    **Description**: Replaces the default 64-bit commpage bcopy implementation with one that does not require SSSE3, useful for legacy platforms. This prevents a `commpage no match for last` panic due to no available 64-bit bcopy functions that do not require SSSE3.

14. `PanicNoKextDump`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: 10.13 (not required for older)
    **Description**: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

15. `PowerTimeoutKernelPanic`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: 10.15 (not required for older)
    **Description**: Disables kernel panic on setPowerState timeout.

    An additional security measure was added to macOS Catalina (10.15) causing kernel panic on power change timeout for Apple drivers. Sometimes it may cause issues on misconfigured hardware, notably digital audio, which sometimes fails to wake up. For debug kernels `setpowerstate_panic=0` boot argument should be used, which is otherwise equivalent to this quirk.

16. `ThirdPartyDrives`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: 10.6 (~~64-bit,~~ not required for older)
    **Description**: Apply vendor patches to IOAHCIBlockStorage.kext to enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

    *Note*: This option may be avoided on user preference. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called `trimforce`. Starting from 10.15 this utility creates `EnableTRIM` variable in `APPLE_BOOT_VARIABLE_GUID` namespace with `01 00 00 00` value.

17. `XhciPortLimit`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Requirement**: 10.11 (not required for older)
    **Description**: Patch various kexts (AppleUSBXHCI.kext, AppleUSBXHCIPCI.kext, IOUSBHostFamily.kext) to remove USB port count limit of 15 ports.

    *Note*: This option should be avoided whenever possible. USB port limit is imposed by the amount of used bits in locationID format and there is no possible way to workaround this without heavy OS modification. The only valid solution is to limit the amount of used ports to 15 (discarding some). More details can be found on AppleLife.ru.

## 7.9 Scheme Properties

These properties are particularly relevant for older macOS operating systems. For more details on how to install and troubleshoot such macOS installation refer to Legacy Apple OS.

1. `FuzzyMatch`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Use `kernelcache` with different checksums when available.

   On macOS 10.6 and earlier `kernelcache` filename has a checksum, which essentially is `adler32` from SMBIOS product name and EfiBoot device path. On certain firmwares EfiBoot device path differs between UEFI and macOS due to ACPI or hardware specifics, rendering `kernelcache` checksum as always different.

   This setting allows matching the latest `kernelcache` with a suitable architecture when the `kernelcache` without suffix is unavailable, improving macOS 10.6 boot performance on several platforms.

2. `KernelArch`
   **Type**: `plist string`
   **Failsafe**: `Auto`
   **Description**: Prefer specified kernel architecture (`Auto`, `i386`, `i386-user32`, `x86_64`) when available.

   On macOS 10.7 and earlier XNU kernel can boot with architectures different from the usual `x86_64`. This setting will use the specified architecture to boot macOS when it is supported by the macOS and the configuration:

   - `Auto` — Choose the preferred architecture automatically.
   - `i386` — Use `i386` (32-bit) kernel when available.
   - `i386-user32` — Use `i386` (32-bit) kernel when available and force the use of 32-bit userspace on 64-bit capable processors if supported by the operating system. On macOS 64-bit capable processors are assumed to support `SSSE3`. This is not the case for older 64-bit capable Pentium processors, which cause some applications to crash on macOS 10.6. ~~The~~ This behaviour corresponds to `-legacy` kernel boot argument. This option is unavailable for 10.4 and 10.5 when running on 64-bit firmware due to an uninitialised 64-bit segment in the XNU kernel, which causes AppleEFIRuntime to incorrectly execute 64-bit code as 16-bit code.
   - `x86_64` — Use `x86_64` (64-bit) kernel when available.

   Below is the algorithm determining the kernel architecture.

   (a) `arch` argument in image arguments (e.g. when launched via UEFI Shell) or in `boot-args` variable overrides any compatibility checks and forces the specified architecture, completing this algorithm.
   (b) OpenCore build architecture restricts capabilities to `i386` and `i386-user32` mode for the 32-bit firmware variant.
   (c) Determined EfiBoot version restricts architecture choice:
      - 10.4-10.5 — `i386` or `i386-user32` (only on 32-bit firmware)
      - 10.6 ~~-10.7~~ — `i386`, `i386-user32`, or `x86_64`
      - 10.7 — `i386` or `x86_64`
      - 10.8 or newer — `x86_64`
   (d) If `KernelArch` is set to `Auto` and `SSSE3` is not supported by the CPU, capabilities are restricted to `i386-user32` if supported by EfiBoot.
   (e) Board identifier (from SMBIOS) based on EfiBoot version disables `x86_64` support on an unsupported model if any `i386` variant is supported. `Auto` is not consulted here as the list is not overridable in EfiBoot.
   (f) `KernelArch` restricts the support to the explicitly specified architecture (when not set to `Auto`) if the architecture remains present in the capabilities.
   (g) The best supported architecture is chosen in this order: `x86_64`, `i386`, `i386-user32`.

   Unlike macOS 10.7, where select boards identifiers are treated as the `i386` only machines, and macOS 10.5 or earlier, where `x86_64` is not supported by the macOS kernel, macOS 10.6 is very special. The architecture choice on macOS 10.6 depends on many factors including not only the board identifier, but also macOS product type (client vs server), macOS point release, and RAM amount. The detection of them all is complicated and not practical, because several point releases had genuine bugs and failed to properly perform the server detection in the first place. For this reason OpenCore on macOS 10.6 will fallback to `x86_64` architecture whenever it is supported by the board at all, just like on macOS 10.7. As a reference here is the 64-bit Mac model compatibility corresponding to actual EfiBoot behaviour on macOS 10.6.8 and 10.7.5.

| Model | 10.6 (minimal) | 10.6 (client) | 10.6 (server) | 10.7 (any) |
|---|---|---|---|---|
| Macmini | 4,x (Mid 2010) | 5,x (Mid 2011) | 4,x (Mid 2010) | 3,x (Early 2009) |
| MacBook | Unsupported | Unsupported | Unsupported | 5,x (2009/09) |
| MacBookAir | Unsupported | Unsupported | Unsupported | 2,x (Late 2008) |
| MacBookPro | 4,x (Early 2008) | 8,x (Early 2011) | 8,x (Early 2011) | 3,x (Mid 2007) |
| iMac | 8,x (Early 2008) | 12,x (Mid 2011) | 12,x (Mid 2011) | 7,x (Mid 2007) |
| MacPro | 3,x (Early 2008) | 5,x (Mid 2010) | 3,x (Early 2008) | 3,x (Early 2008) |
| Xserve | 2,x (Early 2008) | 2,x (Early 2008) | 2,x (Early 2008) | 2,x (Early 2008) |

*Note*: `3+2` and `6+4` hotkeys to choose the preferred architecture are unsupported due to being handled by EfiBoot and thus being hard to properly detect.

3. `KernelCache`
   **Type**: `plist string`
   **Failsafe**: `Auto`
   **Description**: Prefer specified kernel cache type (`Auto`, `Cacheless`, `Mkext`, `Prelinked`) when available.

   Different variants of macOS support different kernel caching variants designed to improve boot performance. This setting ~~allows to prevent using~~ prevents the use of faster kernel caching variants if slower variants are available for debugging and stability reasons. I.e., by specifying `Mkext`~~one will disable~~, `Prelinked` will be disabled for e.g. 10.6 but not for 10.7.

   The list of available kernel caching types and its current support in OpenCore is listed below.

| macOS | i386 NC | i386 MK | i386 PK | x86_64 NC | x86_64 MK | x86_64 PK | x86_64 K |
|---|---|---|---|---|---|---|---|
| 10.4 | ~~NO~~ YES | ~~NO~~ YES (V1) | NO (V1) | — | — | — | — |
| 10.5 | ~~NO~~ YES | ~~NO~~ YES (V1) | NO (V1) | — | — | — | — |
| 10.6 | ~~NO~~ YES | ~~NO~~ YES (V2) | ~~NO~~ YES (V2) | YES | YES (V2) | YES (V2) | — |
| 10.7 | ~~NO~~ YES | — | ~~NO~~ YES (V3) | YES | — | YES (V3) | — |
| 10.8-10.9 | — | — | — | YES | — | YES (V3) | — |
| 10.10-10.15 | — | — | — | — | — | YES (V3) | — |
| 11.0+ | — | — | — | — | — | YES (V3) | YES |

*Note*: First version (V1) of 32-bit `prelinkedkernel` is unsupported due to kext symbol tables being corrupted by the tools. On these versions `Auto` will block `prelinkedkernel` booting. This also makes `keepsyms=1` for kext frames broken on these systems.

- 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~~ when planning to use them.

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

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

## 8.2 Properties

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

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

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

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

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

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

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

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

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

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

## 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 colours according to UEFI specification. The value of black background and black foreground (`0`) is reserved. List of colour names:

   - `0x00` — `EFI_BLACK`

- 0x01 — EFI_BLUE
- 0x02 — EFI_GREEN
- 0x03 — EFI_CYAN
- 0x04 — EFI_RED
- 0x05 — EFI_MAGENTA
- 0x06 — EFI_BROWN
- 0x07 — EFI_LIGHTGRAY
- 0x08 — EFI_DARKGRAY
- 0x09 — EFI_LIGHTBLUE
- 0x0A — EFI_LIGHTGREEN
- 0x0B — EFI_LIGHTCYAN
- 0x0C — EFI_LIGHTRED
- 0x0D — EFI_LIGHTMAGENTA
- 0x0E — EFI_YELLOW
- 0x0F — EFI_WHITE
- 0x00 — EFI_BACKGROUND_BLACK
- 0x10 — EFI_BACKGROUND_BLUE
- 0x20 — EFI_BACKGROUND_GREEN
- 0x30 — EFI_BACKGROUND_CYAN
- 0x40 — EFI_BACKGROUND_RED
- 0x50 — EFI_BACKGROUND_MAGENTA
- 0x60 — EFI_BACKGROUND_BROWN
- 0x70 — EFI_BACKGROUND_LIGHTGRAY

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

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

   - `None` — Avoid hibernation ~~for your own good~~(Recommended).
   - `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. `Reset NVRAM`).

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

4. `PickerAttributes`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Sets specific attributes for picker.

   Different pickers may be configured through the attribute mask containing OpenCore-reserved (`BIT0~BIT15`) and OEM-specific (`BIT16~BIT31`) values.

   Current OpenCore values include:

   - 0x0001 — `OC_ATTR_USE_VOLUME_ICON`, provides custom icons for boot entries:

**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~~ the boot argument `kcsuffix=development` should be added. Use `uname -a` command to ensure that ~~your~~ the current loaded kernel is a development (or a debug) kernel.

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

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

3. `DisableWatchDog`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Select firmwares may not succeed in quickly booting the operating system, especially in debug mode, which results in watch dog timer aborting the process. This option turns off watch dog timer.

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

5. `DisplayLevel`
**Type**: `plist integer`, 64 bit
**Failsafe**: `0`
**Description**: EDK II debug level bitmask (sum) showed onscreen. Unless `Target` enables console (onscreen) printing, onscreen debug output will not be visible. The following levels are supported (discover more in DebugLib.h):

- `0x00000002` (bit `1`) — `DEBUG_WARN` in `DEBUG`, `NOOPT`, `RELEASE`.
- `0x00000040` (bit `6`) — `DEBUG_INFO` in `DEBUG`, `NOOPT`.
- `0x00400000` (bit `22`) — `DEBUG_VERBOSE` in custom builds.
- `0x80000000` (bit `31`) — `DEBUG_ERROR` in `DEBUG`, `NOOPT`, `RELEASE`.

6. `SerialInit`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Perform serial port initialisation.

This option will perform serial port initialisation within OpenCore prior to enabling (any) debug logging. Serial port configuration is defined via PCDs at compile time in `gEfiMdeModulePkgTokenSpaceGuid` GUID. Default values as found in `MdeModulePkg.dec` are as follows:

- `PcdSerialBaudRate` — Baud rate: `115200`.
- `PcdSerialLineControl` — Line control: no parity, 8 data bits, 1 stop bit.

See more details in `Debugging` section.

7. `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~~ this option is needed.

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

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

   **Drivers and tools**:

   - `BMF` — OpenCanopy, bitmap font
   - `BS` — Bootstrap

- GSTT — GoptStop
- HDA — AudioDxe
- KKT — KeyTester
- MMDD — MmapDump
- OCPAVP — PavpProvision
- OCRST — ResetSystem
- OCUI — OpenCanopy
- OC — OpenCore main
- VMOPT — VerifyMemOpt

**Libraries**:

- AAPL — OcDebugLogLib, Apple EfiBoot logging
- OCABC — OcAfterBootCompatLib
- OCAE — OcAppleEventLib
- OCAK — OcAppleKernelLib
- OCAU — OcAudioLib
- OCAV — OcAppleImageVerificationLib
- OCA —- OcAcpiLib
- OCBP — OcAppleBootPolicyLib
- OCB — OcBootManagementLib
- OCCL — OcAppleChunkListLib
- OCCPU — OcCpuLib
- OCC — OcConsoleLib
- OCDC — OcDriverConnectionLib
- OCDH — OcDataHubLib
- OCDI — OcAppleDiskImageLib
- ~~OCFSQ — OcFileLib, UnblockFs quirk~~
- OCFS — OcFileLib
- OCFV — OcFirmwareVolumeLib
- OCHS — OcHashServicesLib
- OCI4 — OcAppleImg4Lib
- OCIC — OcImageConversionLib
- OCII — OcInputLib
- OCJS — OcApfsLib
- OCKM — OcAppleKeyMapLib
- OCL — OcDebugLogLib
- OCMCO — OcMachoLib
- OCME — OcHeciLib
- OCMM — OcMemoryLib
- OCPI — OcFileLib, partition info
- OCPNG — OcPngLib
- OCRAM — OcAppleRamDiskLib
- OCRTC — OcRtcLib
- OCSB — OcAppleSecureBootLib
- OCSMB — OcSmbiosLib
- OCSMC — OcSmcLib
- OCST — OcStorageLib
- OCS — OcSerializedLib
- OCTPL — OcTemplateLib
- OCUC — OcUnicodeCollationLib
- OCUT — OcAppleUserInterfaceThemeLib
- OCXML — OcXmlLib

## 8.5 Security Properties

1. AllowNvramReset
   **Type**: plist boolean
   **Failsafe**: false

**Description**: Allow `CMD+OPT+P+R` handling and enable showing `NVRAM Reset` entry in boot picker.

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

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

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

3. `ApECID`
   **Type**: `plist integer`, 64 bit
   **Failsafe**: `0`
   **Description**: Apple Enclave Identifier.

   Setting this value to any non-zero 64-bit integer will allow using personalised Apple Secure Boot identifiers. ~~If you want to~~ To use this setting, make sure to generate a random 64-bit number with a cryptographically secure random number generator. With this value set and `SecureBootModel` valid and not `Disabled` it is possible to achieve `Full Security` of Apple Secure Boot.

   To start using personalised Apple Secure Boot~~you will have to reinstall~~ , the operating system ~~or personalise it. Until your~~ will have to be reinstalled or personalised. Unless the operating system is personalised~~you will only be able to load~~ , macOS DMG recovery ~~. If you do not have DMG recovery you could always download it~~ cannot be loaded. If DMG recovery is missing, it can be downloaded with `macrecovery` utility and put to `com.apple.recovery.boot` as explained in Tips and Tricks section. ~~Keep in mind~~ Note that DMG loading needs to be set to `Signed` to use any DMG with Apple Secure Boot.

   To personalise an existing operating system use `bless` command after loading to macOS DMG recovery. Mount the system volume partition, unless it has already been mounted, and execute the following command:

```
bless bless --folder "/Volumes/Macintosh HD/System/Library/CoreServices" \
  --bootefi --personalize
```

   When reinstalling the operating system, ~~keep in mind~~ note that current versions of macOS Installer, tested as of 10.15.6, will usually run out of free memory on the `/var/tmp` partition when trying to install macOS with the personalised Apple Secure Boot. Soon after downloading the macOS installer image an `Unable to verify macOS` error message will appear. To workaround this issue allocate a dedicated RAM disk of 2 MBs for macOS personalisation by entering the following commands in macOS recovery terminal before starting the installation:

```
disk=$(hdiutil attach -nomount ram://4096)
diskutil erasevolume HFS+ SecureBoot $disk
diskutil unmount $disk
mkdir /var/tmp/OSPersonalizationTemp
diskutil mount -mountpoint /var/tmp/OSPersonalizationTemp $disk
```

4. `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~~ A dedicated terminal command can be used to perform authenticated restarts: `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.

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~~ This option should be used without any warranty exclusively on the boards known to be compatible.

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

6. `DmgLoading`
   **Type**: `plist string`
   **Failsafe**: `Signed`
   **Description**: Define Disk Image (DMG) loading policy used for macOS Recovery.

   Valid values:

   - `Disabled` — loading DMG images will fail. `Disabled` policy will still let macOS Recovery to load in most cases as there usually are `boot.efi` files compatible with Apple Secure Boot. Manually downloaded DMG images stored in `com.apple.recovery.boot` directories will not load, however.
   - `Signed` — only Apple-signed DMG images will load. Due to Apple Secure Boot design `Signed` policy will let any Apple-signed macOS Recovery to load regardless of Apple Secure Boot state, which may not always be desired.
   - `Any` — any DMG images will mount as normal filesystems. `Any` policy is strongly not recommended and will cause a boot failure when Apple Secure Boot is activated.

7. `EnablePassword`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable password protection to allow sensitive operations.

   Password protection ensures that sensitive operations like booting a non-default operating system (e.g. macOS recovery or a tool), resetting NVRAM storage, trying to boot into a non-default mode (e.g. verbose mode or safe mode) are not allowed without explicit user authentication by a custom password. Currently password and salt are hashed with 5000000 iterations of SHA-512.

   *Note*: This functionality is currently in development and is not ready for daily usage.

8. `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
```

9. `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.

10. `PasswordHash`
    **Type**: `plist data` 64 bytes
    **Failsafe**: all zero
    **Description**: Password hash used when `EnabledPassword` is set.

11. `PasswordSalt`
    **Type**: `plist data`
    **Failsafe**: empty
    **Description**: Password salt used when `EnabledPassword` is set.

12. `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 should be performed:

    - 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 is required to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this~~you are recommended to at least~~ , it is recommended to enable UEFI SecureBoot ~~with~~ using a custom certificate ~~, and~~ and to sign `OpenCore.efi` and `BOOTx64.efi` with ~~your~~ a 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.

13. `ScanPolicy`
**Type**: `plist integer`, 32 bit
**Failsafe**: `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 and old SATA.
- `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`

14. `SecureBootModel`
**Type**: `plist string`
**Failsafe**: `Default`
**Description**: Apple Secure Boot hardware model.

Sets Apple Secure Boot hardware model and policy. Specifying this value defines which operating systems will be bootable. Operating systems shipped before the specified model was released will not boot. Valid values:

- `Default` — Recent available model, currently set to `j137`.
- `Disabled` — No model, Secure Boot will be disabled.
- `j137` — `iMacPro1,1 (December 2017) minimum macOS 10.13.2 (17C2111)`
- `j680` — `MacBookPro15,1 (July 2018) minimum macOS 10.13.6 (17G2112)`
- `j132` — `MacBookPro15,2 (July 2018) minimum macOS 10.13.6 (17G2112)`
- `j174` — `Macmini8,1 (October 2018) minimum macOS 10.14 (18A2063)`
- `j140k` — `MacBookAir8,1 (October 2018) minimum macOS 10.14.1 (18B2084)`
- `j780` — `MacBookPro15,3 (May 2019) minimum macOS 10.14.5 (18F132)`
- `j213` — `MacBookPro15,4 (July 2019) minimum macOS 10.14.5 (18F2058)`
- `j140a` — `MacBookAir8,2 (July 2019) minimum macOS 10.14.5 (18F2058)`
- `j152f` — `MacBookPro16,1 (November 2019) minimum macOS 10.15.1 (19B2093)`
- `j160` — `MacPro7,1 (December 2019) minimum macOS 10.15.1 (19B88)`
- `j230k` — `MacBookAir9,1 (March 2020) minimum macOS 10.15.3 (19D2064)`
- `j214k` — `MacBookPro16,2 (May 2020) minimum macOS 10.15.4 (19E2269)`
- `j223` — `MacBookPro16,3 (May 2020) minimum macOS 10.15.4 (19E2265)`
- `j215` — `MacBookPro16,4 (June 2020) minimum macOS 10.15.5 (19F96)`
- `j185` — `iMac20,1 (August 2020) minimum macOS 10.15.6 (19G2005)`
- `j185f` — `iMac20,2 (August 2020) minimum macOS 10.15.6 (19G2005)`

`PlatformInfo` and `SecureBootModel` are independent, allowing to enabling Apple Secure Boot with any SMBIOS. Setting `SecureBootModel` to any valid value but `Disabled` is equivalent to `Medium Security` of Apple Secure Boot. ~~To achieve `Full Security` one will need to also specify~~ The `ApECID` value must also be specified to achieve `Full Security`.

Enabling Apple Secure Boot is more demanding to incorrect configurations, buggy macOS installations, and unsupported setups. Things to ~~keep in mind~~consider:

(a) ~~Just like on~~ As with T2 Macs~~you will not be able to install any~~ , unsigned kernel drivers and several signed kernel drivers, including NVIDIA Web Drivers, cannot be installed.
(b) The list of cached drivers may be different, resulting in the need to change the list of `Added` or `Forced` kernel drivers. For example, `IO80211Family` cannot be injected in this case.
(c) System volume alterations on operating systems with sealing, like macOS 11, may result in the operating system being unbootable. Do not try to disable system volume encryption unless ~~you disable~~ Apple Secure Boot is disabled.
(d) If ~~your~~ the platform requires certain settings, but they were not enabled, because the obvious issues did not trigger before, ~~you may get boot failure~~ boot failure might occur. Be extra careful with `IgnoreInvalidFlexRatio` or `HashServices`.
(e) Operating systems released before Apple Secure Boot landed (e.g. macOS 10.12 or earlier) will still boot until UEFI Secure Boot is enabled. This is so, because from Apple Secure Boot point they are treated as incompatible and are assumed to be handled by the firmware just like Microsoft Windows is.
(f) On older CPUs (e.g. before Sandy Bridge) enabling Apple Secure Boot might cause slightly slower loading by up to 1 second.
(g) Since `Default` value will increase with time to support the latest major release operating system, it is not recommended to use `ApECID` and `Default` value together.

Sometimes the already installed operating system may have outdated Apple Secure Boot manifests on the `Preboot` partition causing boot failure. If ~~you see the~~ there is "OCB: Apple Secure Boot prohibits this boot entry,

# 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.

## 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 deleted. I.e. to overwrite an existing variable value add the variable name to the `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. `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 `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 third-party scripts may require `ExposeSensitiveData` set to `0x3` to provide `boot-path` variable with OpenCore EFI partition UUID.

**Warning**: This feature is very dangerous as it passes unprotected data to ~~your~~ firmware variable services. Use it only when no hardware NVRAM implementation is provided by the firmware or it is incompatible.

4. `LegacyOverwrite`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Permits overwriting firmware variables from `nvram.plist`.

   *Note*: Only variables accessible from the operating system will be overwritten.

5. `LegacySchema`
   **Type**: `plist dict`
   **Description**: Allows setting select NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

   ~~You can use~~ `*` value can be used to accept all variables for select GUID.

   **WARNING**: Choose variables very carefully, as nvram.plist is not vaulted. For instance, do not put `boot-args` or `csr-active-config`, as this can bypass SIP.

6. `WriteFlash`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enables writing to flash memory for all added variables.

   *Note*: This value is recommended to be enabled on most firmwares, but is left configurable for firmwares that may have issues with NVRAM variable storage garbage collection or alike.

To read NVRAM variable value from macOS~~one could use~~ `nvram` ~~by concatenating variable~~ could be used by concatenating GUID and name ~~separated by~~ variables separated by a `:` symbol. For example, `nvram 7C436110-AB2A-4BBB-A880-FE`

A continuously updated variable list can be found in a corresponding document: NVRAM Variables.

## 9.3 Mandatory Variables

**Warning**: These variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Using `PlatformInfo` is the recommend way of setting these variables.

The following variables are mandatory for macOS functioning:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeatures`
  32-bit `FirmwareFeatures`. Present on all Macs to avoid extra parsing of SMBIOS tables
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeaturesMask`
  32-bit `FirmwareFeaturesMask`. Present on all Macs to avoid extra parsing of SMBIOS tables.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:MLB`
  `BoardSerialNumber`. Present on newer Macs (2013+ at least) to avoid extra parsing of SMBIOS tables, especially in `boot.efi`.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM`
  Primary network adapter MAC address or replacement value. Present on newer Macs (2013+ at least) to avoid accessing special memory region, especially in `boot.efi`.

## 9.4 Recommended Variables

The following variables are recommended for faster startup or other improvements:

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:csr-active-config`
  32-bit System Integrity Protection bitmask. Declared in XNU source code in csr.h.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeatures`
  Combined `FirmwareFeatures` and `ExtendedFirmwareFeatures`. Present on newer Macs to avoid extra parsing of SMBIOS tables
- `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 is needed.
- 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 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)
  - arch=i386 (force kernel architecture to i386, see KernelArch)
  - 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 on 10.7+)
  - -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~~2 — enables serial logging for `EXITBS:START` onwards.
  - `3` — enables serial logging when debug protocol is missing.
  - `4` — enables serial logging unconditionally.
- `timestamps=VALUE` — enables timestamp logging.
  - `0` — disables timestamp logging.
  - `1` — enables timestamp logging (default).
- `log=VALUE` — deprecated starting from 10.15.
  - `1` — AppleLoggingConOutOrErrSet/AppleLoggingConOutOrErrPrint (classical ConOut/StdErr)
  - `2` — AppleLoggingStdErrSet/AppleLoggingStdErrPrint (StdErr or serial?)
  - `4` — AppleLoggingFileSet/AppleLoggingFilePrint (BOOTER.LOG/BOOTER.OLD file on EFI partition)

# 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 `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 are specified (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.

   **Warning**: It is strongly discouraged set this option to `false` when intending to update platform information. The only reason to do that is when doing minor correction of the SMBIOS present and alike. In all other cases not using `Automatic` may lead to hard to debug errors.

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`
   **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. `SystemMemoryStatus`
   **Type**: `plist string`
   **Failsafe**: `Auto`
   **Description**: Indicates whether system memory is upgradable in `PlatformFeature`. This controls the visibility of the Memory tab in About This Mac.

   Valid values:

- `Auto` — use the original `PlatformFeature` value.
- `Upgradable` — explicitly unset `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` (`0x2`) in `PlatformFeature`.
- `Soldered` — explicitly set `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` (`0x2`) in `PlatformFeature`.

*Note*: On certain Mac models (namely `MacBookPro10,x` and any `MacBookAir`), SPMemoryReporter.spreporter will ignore `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` and assume that system memory is non-upgradable.

4. `ProcessorType` **Type**: `plist integer`
   **Failsafe**: `0` (Automatic)
   **Description**: Refer to SMBIOS `ProcessorType`.

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

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

7. `SystemUUID`
   **Type**: `plist string`, GUID
   **Failsafe**: OEM specified
   **Description**: Refer to SMBIOS `SystemUUID`.

8. `MLB`
   **Type**: `plist string`
   **Failsafe**: `OPENCORE_MLB_SN11`
   **Description**: Refer to SMBIOS `BoardSerialNumber`.

9. `ROM`
   **Type**: `plist data`, 6 bytes
   **Failsafe**: all zero
   **Description**: Refer to `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM`.

## 10.3 DataHub Properties

1. `PlatformName`
   **Type**: `plist string`
   **Failsafe**: Not installed
   **Description**: Sets `name` in `gEfiMiscSubClassGuid`. Value found on Macs is `platform` in ASCII.

2. `SystemProductName`
   **Type**: `plist string`
   **Failsafe**: Not installed
   **Description**: Sets `Model` in `gEfiMiscSubClassGuid`. Value found on Macs is equal to SMBIOS `SystemProductName` in Unicode.

3. `SystemSerialNumber`
   **Type**: `plist string`
   **Failsafe**: Not installed
   **Description**: Sets `SystemSerialNumber` in `gEfiMiscSubClassGuid`. Value found on Macs is equal to SMBIOS `SystemSerialNumber` in Unicode.

4. `SystemUUID`
   **Type**: `plist string`, GUID
   **Failsafe**: Not installed
   **Description**: Sets `system-id` in `gEfiMiscSubClassGuid`. Value found on Macs is equal to SMBIOS `SystemUUID`.

5. `BoardProduct`
   **Type**: `plist string`
   **Failsafe**: Not installed

**Description**: Sets `board-id` in `gEfiMiscSubClassGuid`. Value found on Macs is equal to SMBIOS `BoardProduct` in ASCII.

6. `BoardRevision`
   **Type**: `plist data`, 1 byte
   **Failsafe**: `0`
   **Description**: Sets `board-rev` in `gEfiMiscSubClassGuid`. Value found on Macs seems to correspond to internal board revision (e.g. `01`).

7. `StartupPowerEvents`
   **Type**: `plist integer`, 64-bit
   **Failsafe**: `0`
   **Description**: Sets `StartupPowerEvents` in `gEfiMiscSubClassGuid`. Value found on Macs is power management state bitmask, normally 0. Known bits read by `X86PlatformPlugin.kext`:

   - `0x00000001` — Shutdown cause was a `PWROK` event (Same as `GEN_PMCON_2` bit 0)
   - `0x00000002` — Shutdown cause was a `SYS_PWROK` event (Same as `GEN_PMCON_2` bit 1)
   - `0x00000004` — Shutdown cause was a `THRMTRIP#` event (Same as `GEN_PMCON_2` bit 3)
   - `0x00000008` — Rebooted due to a SYS_RESET# event (Same as `GEN_PMCON_2` bit 4)
   - `0x00000010` — Power Failure (Same as `GEN_PMCON_3` bit 1 `PWR_FLR`)
   - `0x00000020` — Loss of RTC Well Power (Same as `GEN_PMCON_3` bit 2 `RTC_PWR_STS`)
   - `0x00000040` — General Reset Status (Same as `GEN_PMCON_3` bit 9 `GEN_RST_STS`)
   - `0xffffff80` — SUS Well Power Loss (Same as `GEN_PMCON_3` bit 14)
   - `0x00010000` — Wake cause was a ME Wake event (Same as PRSTS bit 0, `ME_WAKE_STS`)
   - `0x00020000` — Cold Reboot was ME Induced event (Same as `PRSTS` bit 1 `ME_HRST_COLD_STS`)
   - `0x00040000` — Warm Reboot was ME Induced event (Same as `PRSTS` bit 2 `ME_HRST_WARM_STS`)
   - `0x00080000` — Shutdown was ME Induced event (Same as `PRSTS` bit 3 `ME_HOST_PWRDN`)
   - `0x00100000` — Global reset ME Wachdog Timer event (Same as `PRSTS` bit 6)
   - `0x00200000` — Global reset PowerManagment Wachdog Timer event (Same as `PRSTS` bit 15)

8. `InitialTSC`
   **Type**: `plist integer`, 64-bit
   **Failsafe**: `0`
   **Description**: Sets `InitialTSC` in `gEfiProcessorSubClassGuid`. Sets initial TSC value, normally 0.

9. `FSBFrequency`
   **Type**: `plist integer`, 64-bit
   **Failsafe**: ~~Automatic~~0 (Automatic)
   **Description**: Sets `FSBFrequency` in `gEfiProcessorSubClassGuid`.

   Sets CPU FSB frequency. This value equals to CPU nominal frequency divided by CPU maximum bus ratio and is specified in Hz. Refer to `MSR_NEHALEM_PLATFORM_INFO` (`CEh`) MSR value to determine maximum bus ratio on modern Intel CPUs.

   *Note*: This value is not used on Skylake and newer but is still provided to follow suit.

10. `ARTFrequency`
    **Type**: `plist integer`, 64-bit
    **Failsafe**: ~~Automatic~~0 (Automatic)
    **Description**: Sets `ARTFrequency` in `gEfiProcessorSubClassGuid`.

    This value contains CPU ART frequency, also known as crystal clock frequency. Its existence is exclusive to Skylake generation and newer. The value is specified in Hz, and is normally 24 MHz for client Intel segment, 25 MHz for server Intel segment, and 19.2 MHz for Intel Atom CPUs. macOS till 10.15 inclusive assumes 24 MHz by default.

    *Note*: On Intel Skylake X ART frequency may be a little less (approx. 0.25%) than 24 or 25 MHz due to special EMI-reduction circuit as described in Acidanthera Bugtracker.

11. `DevicePathsSupported`
    **Type**: `plist integer`, 32-bit
    **Failsafe**: Not installed

**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~~0 (Automatic)
    **SMBIOS**: `APPLE_SMBIOS_TABLE_TYPE131 - ProcessorType`
    **Description**: Combined of Processor Major and Minor types.

    Automatic value generation tries to provide most accurate value for the currently installed CPU. When this fails please make sure to create an issue and provide `sysctl machdep.cpu` and `dmidecode` output. For a full list of available values and their limitations (the value will only apply if the CPU core count matches) refer to Apple SMBIOS definitions header here.

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~~ the system to unbootable state or even cause permanent firmware damage. Some of the known drivers are listed below:

| | |
|---|---|
| AudioDxe* | HDA audio support driver in UEFI firmwares for most Intel and some other analog audio controllers. Staging driver, refer to acidanthera/bugtracker#740 for known issues in AudioDxe. |
| CrScreenshotDxe* | Screenshot making driver saving images to the root of OpenCore partition (ESP) or any available writeable filesystem upon pressing F10. This is a modified version of CrScreenshotDxe driver by Nikolaj Schlej. |
| ExFatDxe | Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmwares. For Sandy Bridge and earlier CPUs ExFatDxeLegacy driver should be used due to the lack of RDRAND instruction support. |
| HfsPlus | Proprietary HFS file system driver with bless support commonly found in Apple firmwares. For Sandy Bridge and earlier CPUs HfsPlusLegacy driver should be used due to the lack of RDRAND instruction support. |
| HiiDatabase* | HII services support driver from MdeModulePkg. This driver is included in most firmwares starting with Ivy Bridge generation. Some applications with the GUI like UEFI Shell may need this driver to work properly. |
| EnhancedFatDxe | FAT filesystem driver from FatPkg. This driver is embedded in all UEFI firmwares, and cannot be used from OpenCore. It is known that multiple firmwares have a bug in their FAT support implementation, which leads to corrupted filesystems on write attempt. Embedding this driver within the firmware may be required in case writing to EFI partition is needed during the boot process. |
| NvmExpressDxe* | NVMe support driver from MdeModulePkg. This driver is included in most firmwares starting with Broadwell generation. For Haswell and earlier embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed. |
| OpenCanopy* | OpenCore plugin implementing graphical interface. |
| OpenRuntime* | OpenCore plugin implementing OC_FIRMWARE_RUNTIME protocol. |
| OpenUsbKbDxe* | USB keyboard driver adding the support of AppleKeyMapAggregator protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin KeySupport, which may work better or worse depending on the firmware. |
| PartitionDxe | Proprietary partition management driver with Apple Partitioning Scheme support commonly found in Apple firmwares. This driver can be used to support loading older DMG recoveries such as macOS 10.9 using Apple Partitioning Scheme. For Sandy Bridge and earlier CPUs PartitionDxeLegacy driver should be used due to the lack of RDRAND instruction support. |
| Ps2KeyboardDxe* | PS/2 keyboard driver from MdeModulePkg. OpenDuetPkg and some firmwares may not include this driver, but it is necessary for PS/2 keyboard to work. Note, unlike OpenUsbKbDxe this driver has no AppleKeyMapAggregator support and thus requires KeySupport to be enabled. |
| Ps2MouseDxe* | PS/2 mouse driver from MdeModulePkg. Some very old laptop firmwares may not include this driver, but it is necessary for touchpad to work in UEFI graphical interfaces, such as OpenCanopy. |
| UsbMouseDxe* | USB mouse driver from MdeModulePkg. Some virtual machine firmwares like OVMF may not include this driver, but it is necessary for mouse to work in UEFI graphical interfaces, such as OpenCanopy. |
| VBoxHfs | HFS file system driver with bless support. This driver is an alternative to a closed source HfsPlus driver commonly found in Apple firmwares. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit. |
| XhciDxe* | XHCI USB controller support driver from MdeModulePkg. This driver is included in most firmwares starting with Sandy Bridge generation. For earlier firmwares or legacy systems it may be used to support external USB 3.0 PCI cards. |

Driver marked with * are bundled with OpenCore. To compile the drivers from UDK (EDK II) ~~use~~ the same command ~~you normally use~~ used for OpenCore compilation can be taken, but choose a corresponding package:

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

## 11.3 Tools and Applications

Standalone tools may help to debug firmware and hardware. Some of the known tools are listed below. While some tools can be launched from within OpenCore, see more details in the Tools subsection of the configuration, most should be run separately either directly or from `Shell`.

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

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

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

Listing 3: Blessing tool

*Note 1*: ~~You may have to copy~~ `/System/Library/CoreServices/BridgeVersion.bin` should be copied to `/Volumes/VOLNAME/DIR`
*Note 2*: To be able to use `bless` ~~you may have to~~ disabling System Integrity Protection is necessary.
*Note 3*: To be able to boot ~~you may have to~~ Secure Boot might be disabled if present.

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

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

## 11.4 OpenCanopy

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

OpenCanopy requires graphical resources located in `Resources` directory to run. Sample resources (fonts and images) can be found in OcBinaryData repository. ~~You can find customised icons~~ Customised icons can be found over the internet (e.g. here or there).

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

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

- `Cursor` — Mouse cursor (mandatory).
- `Selected` — Selected item (mandatory).
- `Selector` — Selecting item (mandatory).
- `HardDrive` — Generic OS (mandatory).
- `Apple` — Apple OS.
- `AppleRecv` — Apple Recovery OS.
- `AppleTM` — Apple Time Machine.

## 11.7 APFS Properties

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

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

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

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

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

   APFS verbose output can be useful for debugging.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## 11.9 Input Properties

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

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

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

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

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

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

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

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

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

4. `KeySupport`
   **Type**: `plist boolean`

**Failsafe**: `false`
**Description**: Enable internal keyboard input translation to `AppleKeyMapAggregator` protocol.

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

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

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

*Note*: Currently `V1`, `V2`, and `AMI` unlike `Auto` only do filtering of the particular specified protocol. This may change in the future versions.

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

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

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

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

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

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

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

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

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

## 11.10 Output Properties

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

Currently two renderers are supported: `Builtin` and `System`. `System` renderer uses firmware services for text rendering. `Builtin` bypassing firmware services and performs text rendering on its own. Different renderers

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

## 11.12 Quirks Properties

1. `DeduplicateBootOrder`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Remove duplicate entries in `BootOrder` variable in `EFI_GLOBAL_VARIABLE_GUID`.

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

   By redirecting `Boot` prefixed variables to a separate GUID namespace with the help of `RequestBootVarRouting` quirk we achieve multiple goals:

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

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

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

   This quirk removes all duplicates in `BootOrder` variable attempting to resolve the consequences of the bugs upon OpenCore loading. It is recommended to use this key along with `BootProtect` option.

2. `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.

3. `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.

4. `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.

5. `RequestBootVarRouting`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Request redirect of all `Boot` prefixed variables from `EFI_GLOBAL_VARIABLE_GUID` to `OC_VENDOR_VARIABLE_GUID`.

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

6. `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.

7. `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).

   *Note*: Some firmwares may not allocate memory areas used by S3 (sleep) and S4 (hibernation) code unless CSM is enabled causing wake failures. After comparing the memory maps with CSM disabled and enabled, these areas can be found in the lower memory and can be fixed up by doing the reservation. See `Sample.plist` for more details.

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. Type
   **Type**: plist string
   **Failsafe**: Reserved
   **Description**: Memory region type matching the UEFI specification memory descriptor types. Mapping:

   - Reserved — EfiReservedMemoryType
   - LoaderCode — EfiLoaderCode
   - LoaderData — EfiLoaderData
   - BootServiceCode — EfiBootServicesCode
   - BootServiceData — EfiBootServicesData
   - RuntimeCode — EfiRuntimeServicesCode
   - RuntimeData — EfiRuntimeServicesData
   - Available — EfiConventionalMemory
   - Persistent — EfiPersistentMemory
   - UnusableMemory — EfiUnusableMemory
   - ACPIReclaimMemory — EfiACPIReclaimMemory
   - ACPIMemoryNVS — EfiACPIMemoryNVS
   - MemoryMappedIO — EfiMemoryMappedIO
   - MemoryMappedIOPortSpace — EfiMemoryMappedIOPortSpace
   - PalCode — EfiPalCode

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

# 12 Troubleshooting

## 12.1 Legacy Apple OS

Older operating systems may be more complicated to install, but sometimes can be necessary to use for all kinds of reasons. While a compatible board identifier and CPUID are the obvious requiremenets for proper functioning of an older operating system, there are many other less obvious things to ~~keep in mind~~consider. This section tries to cover a common set of issues relevant to installing older macOS operating systems.

### 12.1.1 macOS 10.8 and 10.9

- Disk images on these systems use Apple Partitioning Scheme and will require the proprietary `PartitionDxe` driver to run DMG recovery and installation. It is possible to set `DmgLoading` to `Disabled` to run the recovery without DMG loading avoiding the need for `PartitionDxe`.
- Cached kernel images often do not contain family drivers for networking (`IONetworkingFamily`) or audio (`IOAudioFamily`) requiring ~~one to use~~ the use of `Force` loading in order to inject networking or audio drivers.

### 12.1.2 macOS 10.7

- All previous issues apply.
- Many kexts, including `Lilu` and its plugins, are unsupported on macOS 10.7 and older as they require newer kernel APIs, which are not part of the macOS 10.7 SDK.
- Prior to macOS 10.8 KASLR sliding is not supported, which will result in memory allocation failures on firmwares that utilise lower memory for their own purposes. Refer to acidanthera/bugtracker#1125 for tracking.
- 32-bit kernel interaction is unsupported and will lead to issues like kernel patching or injection failure.

### 12.1.3 macOS 10.6

- All previous issues apply.
- Last released installer images for macOS 10.6 are macOS 10.6.7 builds `10J3250` (for `MacBookPro8,x`) and `10J4139` (for `iMac12,x`), without Xcode). These images are limited to their target model identifiers and have no `-no_compat_check` boot argument support. Modified images (with `ACDT` suffix) without model restrictions can be found here, assuming ~~that you legally own~~ macOS 10.6 is legally owned. Read `DIGEST.txt` for more details. ~~Keep in mind,~~ Note that these are the earliest tested versions of macOS 10.6 with OpenCore.

~~You can also patch out model checking yourself~~ Model checking may also be erased by editing `OSInstall.mpkg` with e.g. `Flat Package Editor` by making `Distribution` script to always return `true` in `hwbeModelCheck` function. Since updating the only file in the image and not corrupting other files can be difficult and may cause slow booting due to kernel cache date changes, it is recommended to script image rebuilding as shown below:

```
#!/bin/bash
# Original.dmg is original image, OSInstall.mpkg is patched package
mkdir RO
hdiutil mount Original.dmg -noverify -noautoopen -noautoopenrw -noautofsck -mountpoint RO
cp RO/.DS_Store DS_STORE
hdiutil detach RO -force
rm -rf RO
hdiutil convert Original.dmg -format UDRW -o ReadWrite.dmg
mkdir RW
xattr -c OSInstall.mpkg
hdiutil mount ReadWrite.dmg -noverify -noautoopen -noautoopenrw -noautofsck -mountpoint RW
cp OSInstall.mpkg RW/System/Installation/Packages/OSInstall.mpkg
killall Finder fseventsd
rm -rf RW/.fseventsd
cp DS_STORE RW/.DS_Store
hdiutil detach RW -force
rm -rf DS_STORE RW
```

```
hdiutil convert ReadWrite.dmg -format UDZO -o ReadOnly.dmg
```

## 12.2 UEFI Secure Boot

OpenCore is designed to provide a secure boot chain between ~~your firmware and your~~ firmware and operating system. On most x86 platforms trusted loading is implemented via UEFI Secure Boot model. Not only OpenCore fully supports this model, but it also extends its capabilities to ensure sealed configuration via vaulting and provide trusted loading to the operating systems using custom verification, such as Apple Secure Boot. Proper secure boot chain requires several steps and careful configuration of select settings as explained below:

1. Enable Apple Secure Boot by setting `SecureBootModel` ~~if you need~~ to run macOS. Note, that not every macOS is compatible with Apple Secure Boot and there are several other restrictions as explained in Apple Secure Boot section.
2. Disable DMG loading by setting `DmgLoading` to `Disabled` if ~~you are concerned~~ users have concerns of loading old vulnerable DMG recoveries. This is **not** required, but recommended. For the actual tradeoffs see the details in DMG loading section.
3. Make sure that APFS JumpStart functionality restricts the loading of old vulnerable drivers by setting `MinDate` and `MinVersion` to `0`. More details are provided in APFS JumpStart section. An alternative is to install `apfs.efi` driver manually.
4. Make sure that ~~you do not need~~ `Force` driver loading ~~and can still boot~~ is not needed and all the operating systems ~~you need~~are still bootable.
5. Make sure that `ScanPolicy` restricts loading from undesired devices. It is a good idea to prohibit all removable drivers or unknown filesystems.
6. Sign all the installed drivers and tools with ~~your~~ the private key. Do not sign tools that provide administrative access to ~~your~~ the computer, like UEFI Shell.
7. Vault ~~your~~ the configuration as explained Vaulting section.
8. Sign all OpenCore binaries (`BOOTX64.efi`, `BOOTIa32.efi`, `Bootstrap.efi`, `OpenCore.efi`) used on this system with the same private key.
9. Sign all third-party operating system (not made by Microsoft or Apple) bootloaders if ~~you need them~~needed. For Linux there is an option to install Microsoft-signed Shim bootloader as explained on e.g. Debian Wiki.
10. Enable UEFI Secure Boot in ~~your~~ firmware preferences and install the certificate with a private key~~you own~~. Details on how to generate a certificate can be found in various articles, like this one, and are out of the scope of this document. If ~~you need to launch Windows you~~ Windows is needed one will also need to add the Microsoft Windows Production CA 2011. ~~If you need to~~ To launch option ROMs or ~~decided~~ to use signed Linux drivers~~you will also need the~~ , Microsoft UEFI Driver Signing CA will also be needed.
11. Password-protect changing firmware settings to ensure that UEFI Secure Boot cannot be disabled without ~~your~~ the user's knowledge.

## 12.3 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~~consider:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- 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~~ If there still are 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 is required. 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 may be downloaded and installed 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~~ If there is a previous version of Boot Camp installed ~~you will have to remove it~~ it should be removed 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~~ the rest may still have to be addressed 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 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~~ the partition will have to ~~relabel the partition~~ be relabelled manually. This can be done with many 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
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 419430366
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

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

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

Command (? for help): w
```

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

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

Listing 4: Relabeling Windows volume

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

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

## 12.4 Debugging

Similar to other projects working with hardware OpenCore supports auditing and debugging. The use of `NOOPT` or `DEBUG` build modes instead of `RELEASE` can produce a lot more debug output. With `NOOPT` source level debugging with GDB or IDA Pro is also available. For GDB check OpenCore Debug page. For IDA Pro~~you will need IDA Pro~~, version 7.3 or newer ~~, refer to~~ is needed, and Debugging the XNU Kernel with IDA Pro ~~for more details~~may also help.

To obtain the log during boot ~~you can make the use of~~ serial port debugging can be used. Serial port debugging is enabled in `Target`, e.g. `0xB` for onscreen with serial. To initialise serial within OpenCore use `SerialInit` configuration option. For macOS ~~your best choice are~~ the best choice is CP2102-based UART devices. Connect motherboard `TX` to USB UART `RX`, and motherboard `GND` to USB UART `GND`. Use `screen` utility to get the output, or download GUI software, such as CoolTerm.

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

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

## 12.5 Tips and Tricks

1. **How to debug boot failure?**

   Normally it is enough to obtain the actual error message. For this ensure that:

   - ~~You have a~~ A `DEBUG` or `NOOPT` version of OpenCore is used.
   - 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~~boot arguments are needed. 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~~ users are strongly encouraged to use the `RequestBootVarRouting` quirk, which will preserve ~~your~~ the 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~~ macrecovery.py ~~, builtin tool~~can be used.

   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 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 decide which `Booter` quirks to use?**

   These quirks originate from `AptioMemoryFix` 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 should be enabled:

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