# OpenCore

Reference Manual (0.6.7.8)

[2021.03.03]

# 1 Introduction

This document provides information on the format of the OpenCore user configuration file ~~format~~ used to set up 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 issues which should be reported via the Acidanthera Bugtracker. 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 guide to configuring an end-user Board Support Package (BSP). The intended audience of the document is anticipated to be programmers and engineers with a basic understanding of macOS internals and UEFI 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 similar, may be more useful for a wider audience as they could provide guide-like material. However, they are subject to their authors' preferences, ~~tastes,~~ misinterpretations of this document, and unavoidable obsolescence. In cases of using such sources, such as Dortania's OpenCore Install Guide and related material, please refer back to this document on every decision made and re-evaluate potential ~~consequences~~implications.

Please note that regardless of the sources used, users are required to fully understand every OpenCore configuration option, and the principles behind them, before posting issues to the Acidanthera Bugtracker.

*Note*: Creating this document would not have been possible without the invaluable contributions from other people: Andrey1970, Goldfish64, dakanji, PMheart, and several others, with the full list available in OpenCorePkg history.

## 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 multidata` — 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 (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.

# 2 Configuration

## 2.1 Configuration Terms

- `OC config` — OpenCore Configuration file in `plist` format named `config.plist`. It provides an extensible way to configure OpenCore and is structured to be separated into multiple named sections situated under the root `plist dictionary`. These sections may have `plist array` or `plist dictionary` types and are described in corresponding sections of this document.
- `valid key` — `plist key` object of `OC config` described in this document or its future revisions. Besides explicitly described `valid keys`, keys starting with the `#` symbol (e.g. `#Hello`) are also considered `valid keys` and while they behave as comments, effectively discarding their values, they are still required to be valid `plist objects`. All other `plist keys` are not valid, and their presence results in `undefined behaviour`.
- `valid value` — valid `plist object` of `OC config` described in this document that matches all the additional requirements in specific `plist object` descriptions if any.
- `invalid value` — valid `plist object` of `OC config` described in this document that is of other `plist type`, does not conform to additional requirements found in specific `plist object` descriptions (e.g. value range), or missing from the corresponding collection. `Invalid values` are read with or without an error message as any possible value of this `plist object` in an undetermined manner (i.e. the values may not be same across the reboots). Whilst reading an `invalid value` is equivalent to reading certain defined `valid values`, applying incompatible values to the host system may result in `undefined behaviour`.
- `optional value` — `valid value` of `OC config` described in this document that reads in a certain defined manner provided in specific `plist object` description (instead of `invalid value`) when not present in `OC config`. All other cases of `invalid value` do still apply. Unless explicitly marked as `optional value`, any other value is required to be present and reads to `invalid value` if missing.
- `fatal behaviour` — behaviour leading to boot termination. Implementations shall prevent the boot process from continuing until the host system is restarted. It is permitted, but not required, to execute cold reboots or to show warning messages in such cases.
- `undefined behaviour` — behaviour not prescribed by this document. Implementations may take any measures including, but not limited to, measures associated with `fatal behaviour`, assumptions of any state or value, or disregarding any associated states or values. This is however subject to such measures not negatively impacting upon system integrity.

## 2.2 Configuration Processing

The `OC config` file is guaranteed to be processed at least once if found. ~~Depending on~~ Subject to the OpenCore bootstrapping mechanism, the presence of multiple `OC config` files may lead to the reading of any of them. It is permissible for no `OC Config` file to be present on disk. In such cases, if the implementation does not abort the boot process, all values shall follow the rules of `invalid values` and `optional values`.

The `OC config` file has restrictions on size, nesting levels, and number of keys:

- The `OC config` file size shall not exceed `32 MBs`.
- The `OC config` file shall not have more than `32` nesting levels.
- The `OC config` file may have up to `32,768` XML nodes within each `plist object`.
  - One `plist dictionary` item is counted as a pair of nodes

Reading malformed `OC config` files results in `undefined behaviour`. Examples of malformed `OC config` files include the following:

- `OC config` files that do not conform to `DTD PLIST 1.0`.
- `OC config` files with unsupported or non-conformant `plist objects` found in this document.
- `OC config` files violating restrictions on size, nesting levels, and number of keys.

It is recommended, but not required, to abort loading malformed `OC config` files and to continue as if an `OC config` file is not present. For forward compatibility, it is recommended, but not required, for the implementation to warn about the use of `invalid values`.

# 4 ACPI

## 4.1 Introduction

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

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

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

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

Applying the changes globally resolves the problems of incorrect operating system detection (consistent with the ACPI specification, not possible before the operating system boots), operating system chainloading, and difficult ACPI debugging. Hence, more attention may be required when writing changes to `_OSI`.

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

There are several sources of ACPI tables and workarounds. Commonly used ACPI tables are provided with OpenCore, VirtualSMC, VoodooPS2, and WhateverGreen releases. Besides those, several third-party instructions may be found on the AppleLife Laboratory and DSDT subforums (e.g. Battery register splitting guide). A slightly more user-friendly explanation of some tables included with OpenCore can also be found in Dortania's Getting started with ACPI guide. For more exotic cases, there are several alternatives such as daliansky's ACPI sample collection. Note however that the quality of the suggested solutions will be variable.

## 4.2 Properties

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

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

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

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

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

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

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

**Failsafe**: All zero (Match any table signature)
**Description**: Match table signature equal to this value.

In most cases, ACPI patches are not useful and are 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 certain tables. For ACPI consistency it is much safer to rename devices at the I/O Registry level, as done by WhateverGreen.
- Avoid patching `_OSI` to support a higher feature set level whenever possible. While this enables a number of workarounds on APTIO firmware, it typically results in a need for additional patches. ~~Modern firmware generally does not need this~~ These are not usually needed on modern firmware and smaller patches work well on firmware that does. However, laptop vendors often rely on this method to determine the availability of functions such as modern I2C input support, thermal adjustment and custom feature additions.
- Avoid patching embedded controller event `_Qxx` just to enable brightness keys. The conventional process to find these keys typically involves significant modifications to DSDT and SSDT files and in addition, the debug kext is not stable on newer systems. Please use the built-in brightness key discovery in BrightnessKeys instead.
- Avoid making ad hoc changes such as renaming `_PRW` or `_DSM` whenever possible.

Some cases where patching is actually useful 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 a custom method implementation within an SSDT, to inject shutdown fixes on certain computers for instance, the original method can be replaced with a dummy name by patching `_PTS` with `ZPTS` and adding a callback to the original method.

The Tianocore AcpiAml.h source file may help with better understanding ACPI opcodes.

*Note*: Patches of different `Find` and `Replace` lengths are unsupported as they may corrupt ACPI tables and make the system unstable due to area relocation. If such changes are needed, the utilisation of "proxy" patching or the padding of `NOP` to the remaining area could be considered.

## 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 a few newer 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 flaws that result in boot crashes. Reference: Debugging AppleACPIPlatform on 10.13 by Alex James (also known as theracermaster). The issue was 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 the underlying firmware implementation. Among the position-independent code, ACPI tables may contain the physical addresses of MMIO areas used for device configuration, typically grouped by region (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 results in the shift of the addresses in the aforementioned `OperationRegion` constructions.

# 5 Booter

## 5.1 Introduction

This section allows the application of different types of UEFI modifications to operating system bootloaders, primarily the Apple bootloader (`boot.efi`). The modifications currently provide various patches and environment alterations for different firmware types. Some of these features were originally implemented as part of AptioMemoryFix.efi, which is no longer maintained. Refer to the Tips and Tricks section for instructions on migration.

If this is used for the first time on customised firmware, the following requirements should be met before starting:

- Most up-to-date UEFI firmware (check 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 the ASUS WS-X299-PRO, this option results in adverse effects and must be disabled. While no other motherboards with the same issue are known, 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 the system cannot be booted at all or `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 no option is available (for advanced users only). See VerifyMsrE2 notes for more details.
- `CSM` (Compatibility Support Module) disabled in firmware settings if present. 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 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 `Thunderbolt support`, `Intel SGX`, and `Intel Platform Trust` may have to be disabled in firmware settings present.

When debugging sleep issues, Power Nap and automatic power off (which appear to sometimes cause wake to black screen or boot loop issues on older platforms) may be temporarily disabled. The specific issues may vary, but ~~generally~~ ACPI tables should typically be looked at first.

Here is an example of a defect found ~~in~~ on 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 be reset by hardware changes and in certain other circumstances. To view their current state, use the `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 the MmioWhitelist Properties section below.

2. `Patch`
   **Type**: `plist array`
   **Failsafe**: Empty
   **Description**: Perform binary patches in booter.

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

3. `Quirks`
   **Type**: `plist dict`
   **Description**: Apply individual booter quirks described in the 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. The debug log can be used to find the list of the candidates.

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

3. `Enabled`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Exclude MMIO address from the devirtualisation procedure.

## 5.4 Patch Properties

1. `Arch`
   **Type**: `plist string`
   **Failsafe**: `Any` (Apply to any supported architecture)
   **Description**: Booter patch architecture (`i386`, `x86_64`).

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

3. `Count`
   **Type**: `plist integer`
   **Failsafe**: `0` (Apply to all occurrences found)
   **Description**: Number of patch occurrences to apply.

4. `Enabled`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Set to `true` to activate this booter patch.

5. `Find`
   **Type**: `plist data`
   **Failsafe**: Empty
   **Description**: Data to find. Must be equal to `Replace` in size if set.

6. `Identifier`
   **Type**: `plist string`
   **Failsafe**: `Any` (Match any booter)
   **Description**: `Apple` for macOS booter (~~generally~~ typically `boot.efi`); or a name with a suffix, such as `bootmgfw.efi`, for a specific booter.

7. `Limit`
   **Type**: `plist integer`
   **Failsafe**: `0` (Search the entire booter)
   **Description**: Maximum number of bytes to search for.

18. `SyncRuntimePermissions`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Update memory permissions for the runtime environment.

    Some types of firmware fail to properly handle runtime permissions:

    - They incorrectly mark `OpenRuntime` as not executable in the memory map.
    - They incorrectly mark `OpenRuntime` as not executable in the memory attributes table.
    - They lose entries from the memory attributes table after `OpenRuntime` is loaded.
    - They mark items in the memory attributes table as read-write-execute.

    This quirk ~~tries to update~~ attempts to update the memory map and memory attributes table to correct this.

    *Note*: The need for this quirk is indicated by early boot failures. Only firmware released after 2017 is typically affected.

# 7 Kernel

## 7.1 Introduction

This section allows the application of 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 the 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, inspect the `OSBundleLibraries` key in the `Info.plist` of the kext. Any kext mentioned in the `OSBundleLibraries` of the other kext must 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 the Block Properties section below.

3. `Emulate`
   **Type**: `plist dict`
   **Description**: Emulate certain hardware in kernelspace via parameters described in the 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 the 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~~ typically affects older operating systems, where various dependency kexts, such as `IOAudioFamily` or `IONetworkingFamily` may not be present in the kernel cache by default. The 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 the Patch Properties section below.

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

*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~~ Only the value of `EAX`~~that needs to be taken care of, since it~~ , which represents the full CPUID~~. The remaining bytes are to~~ , typically needs to be accounted for and remaining bytes should be left as zeroes. ~~Byte~~ The byte order is Little Endian~~, so for~~ . 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`

*Note 4*: Be aware 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. `_xcpm_bootstrap` should manually be patched to enforce XCPM on these CPUs instead of this option.
- Low-end CPUs (e.g. Haswell+ Pentium) as they are not supported properly by macOS. Legacy workarounds 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 typically 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
   **Description**: Emulates CPUID and applies `DummyPowerManagement` on specified macOS version or older.

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

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

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

## 7.6 Force Properties

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

## 7.8 Quirks Properties

1. `AppleCpuPmCfgLock`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Requirement**: 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.

   Some types of firmware lock the `PKG_CST_CONFIG_CONTROL` MSR register and the bundled `VerifyMsrE2` tool can be used to check its state. Note that some types of firmware only have this register locked on some cores.

   As modern firmware provide a `CFG Lock` setting that allows configuring the `PKG_CST_CONFIG_CONTROL` MSR register lock, this option should be avoided whenever possible. On APTIO firmware that do not provide a `CFG Lock` setting in the GUI, it is possible to access the option directly:

   (a) Download UEFITool and IFR-Extractor.
   (b) Open the firmware image in UEFITool and find `CFG Lock` unicode string. If it is not present, the firmware may not have this option and the process should therefore be discontinued.
   (c) Extract the `Setup.bin` PE32 Image Section (the 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 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 certain CPUs, which have no native XCPM support.

   This is ~~normally~~ typically used in conjunction with the `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. Only certain Xeon models typically benefit from the patch.

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

The algorithm used to determine the preferred kernel architecture is set out below.

(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 — `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 certain board 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 the macOS product type (client vs server), macOS point release, and amount of RAM. The detection of all these is complicated and impractical, as several point releases had implementation ~~defects~~ flaws resulting in a failure to properly execute the server detection in the first place. For this reason, OpenCore on macOS 10.6 falls back on the `x86_64` architecture whenever it is supported by the board, as it is on macOS 10.7.

A 64-bit Mac model compatibility matrix corresponding to actual EfiBoot behaviour on macOS 10.6.8 and 10.7.5 is outlined below.

| 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 as they are handled by EfiBoot and hence, difficult to 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 prevents the use of faster kernel caching variants if slower variants are available for debugging and stability reasons. ~~I.e.~~That is, by specifying `Mkext`, `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 KC |
|---|---|---|---|---|---|---|---|
| 10.4 | YES | YES (V1) | NO (V1) | — | — | — | — |
| 10.5 | YES | YES (V1) | NO (V1) | — | — | — | — |
| 10.6 | YES | YES (V2) | YES (V2) | YES | YES (V2) | YES (V2) | — |
| 10.7 | YES | — | YES (V3) | YES | — | YES (V3) | — |
| 10.8-10.9 | — | — | — | YES | — | YES (V3) | — |
| 10.10-10.15 | — | — | — | — | — | YES (V3) | — |
| 11+ | — | — | — | — | — | YES (V3) | YES |

To display all entries, the picker menu can be reloaded into "Extended Mode" by pressing the `Spacebar` key. Hiding auxiliary entries may increase boot performance on multi-disk systems.

4. `LauncherOption`
   **Type**: `plist string`
   **Failsafe**: `Disabled`
   **Description**: Register the launcher option in the firmware preferences for persistence.

   Valid values:

   - `Disabled` — do nothing.
   - `Full` — create or update the top priority boot option in UEFI variable storage at bootloader startup.
     - For this option to work, `RequestBootVarRouting` is required to be enabled.
   - `Short` — create a short boot option instead of a complete one.
     - This variant is useful for some older types of firmware, typically from Insyde, that are unable to manage full device paths.

   This option allows integration with third-party operating system installation and upgrades (which may overwrite the `\EFI\BOOT\BOOTx64.efi` file). The BOOTx64.efi file is no longer used for bootstrapping OpenCore if a custom option is created. The custom path used for bootstrapping can be specified by using the `LauncherPath` option.

   *Note 1*: Some types of firmware may have ~~defective NVRAM implementation~~ NVRAM implementation flaws, no boot option support, or other incompatibilities. While unlikely, the use of this option may ~~cause~~ result in boot failures and should only be used exclusively on ~~the~~ boards known to be compatible. Refer to acidanthera/bugtracker#1222 for some known issues ~~with~~ affecting Haswell and other boards.

   *Note 2*: While NVRAM resets executed from OpenCore would not typically erase the boot option created in `Bootstrap`, executing NVRAM resets prior to loading OpenCore will erase the boot option. Therefore, for significant implementation updates~~(e.g. in~~ , such as was the case with OpenCore 0.6.4~~)~~, an NVRAM reset should be ~~performed~~ executed with `Bootstrap` disabled, after which it can be ~~reenabled~~re-enabled.

5. `LauncherPath`
   **Type**: `plist string`
   **Failsafe**: `Default`
   **Description**: Launch path for the `LauncherOption` property.

   `Default` points to `OpenCore.efi`. User specified paths, e.g. `\EFI\SomeLauncher.efi`, can be used to provide custom loaders, which are supposed to load `OpenCore.efi` themselves.

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

   Different OpenCore 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:
     For `Tools`, OpenCore will attempt loading a custom icon and fallback to a default icon on failure:
     - `ResetNVRAM` — `Resources\Image\ResetNVRAM.icns` — `ResetNVRAM.icns` from icons directory.
     - `Tools\<TOOL_RELATIVE_PATH>.icns` — icon near the tool file with appended `.icns` extension.

     For custom boot `Entries`, OpenCore will attempt loading a custom icon and fallback to the volume icon or the default icon on failure:
     - `<ENTRY_PATH>.icns` — icon near the entry file with appended `.icns` extension.

     For all other entries, OpenCore will attempt loading a volume icon by searching as follows, and will fallback to the default icon on failure:
     - `.VolumeIcon.icns` file at `Preboot` volume in per-volume directory (`/System/Volumes/Preboot/{GUID}/` when mounted at the default location within macOS) for APFS (if present).

  - `.VolumeIcon.icns` file at the `Preboot` volume root (`/System/Volumes/Preboot/`, when mounted at the default location within macOS) for APFS (otherwise).
  - `.VolumeIcon.icns` file at the volume root for other filesystems.

  *Note 1*: The Apple picker partially supports placing a volume icon file at the operating system's `Data` volume root, `/System/Volumes/Data/`, when mounted at the default location within macOS. This approach is flawed: the file is neither accessible to OpenCanopy nor to the Apple picker when FileVault 2, which is meant to be the default choice, is enabled. Therefore, OpenCanopy does not attempt supporting Apple's approach. A volume icon file may be placed at the root of the `Preboot` volume for compatibility with both OpenCanopy and the Apple picker, or use the `Preboot` per-volume location as above with OpenCanopy as a preferred alternative to Apple's approach.

  *Note 2*: Be aware that using a volume icon on any drive overrides the normal OpenCore picker behaviour for that drive of selecting the appropriate icon depending on whether the drive is internal or external.

- `0x0002` — `OC_ATTR_USE_DISK_LABEL_FILE`, provides custom rendered titles for boot entries:
  - `.disk_label` (`.disk_label_2x`) file near bootloader for all filesystems.
  - `<TOOL_NAME>.lbl` (`<TOOL_NAME>.l2x`) file near tool for `Tools`.

  Prerendered labels can be generated via the `disklabel` utility or the `bless` command. When disabled or missing text labels, (`.contentDetails` or `.disk_label.contentDetails`) are to be rendered instead.
- `0x0004` — `OC_ATTR_USE_GENERIC_LABEL_IMAGE`, provides predefined label images for boot entries without custom entries. This may however give less detail for the actual boot entry.
- `0x0008` — `OC_ATTR_HIDE_THEMED_ICONS`, prefers builtin icons for certain icon categories to match the theme style. For example, this could force displaying the builtin Time Machine icon. Requires `OC_ATTR_USE_VOLUME_ICON`.
- `0x0010` — `OC_ATTR_USE_POINTER_CONTROL`, enables pointer control in the OpenCore picker when available. For example, this could make use of mouse or trackpad to control UI elements.

7. `PickerAudioAssist`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable screen reader by default in the OpenCore picker.

   For the macOS bootloader, screen reader preference is set in the `preferences.efires` archive in the `isVOEnabled.int32` file and is controlled by the operating system. For OpenCore screen reader support, this option is an independent equivalent. Toggling screen reader support in both the OpenCore picker and the macOS bootloader FileVault 2 login window can also be done by using the `Command` + `F5` key combination.

   *Note*: The screen reader requires working audio support. Refer to the `UEFI Audio Properties` section for more details.

8. `PollAppleHotKeys`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable `modifier hotkey` handling in the OpenCore picker.

   In addition to `action hotkeys`, which are partially described in the `PickerMode` section and are ~~normally~~ typically handled by Apple BDS, modifier keys handled by the operating system bootloader (`boot.efi`) also exist. These keys allow changing the behaviour of the operating system by providing different boot modes.

   On certain firmware, using modifier keys may be problematic due to driver incompatibilities. To workaround this problem, this option allows registering certain hotkeys in a more permissive manner from within the OpenCore picker. Such extensions include support for tapping on keys in addition to holding and pressing `Shift` along with other keys instead of only pressing the `Shift` key, which is not detectable on many PS/2 keyboards.

   This list of known `modifier hotkeys` includes:

   - `CMD+C+MINUS` — disable board compatibility checking.
   - `CMD+K` — boot release kernel, similar to `kcsuffix=release`.
   - `CMD+S` — single user mode.
   - `CMD+S+MINUS` — disable KASLR slide, requires disabled SIP.
   - `CMD+V` — verbose mode.
   - `Shift` — safe mode.

# 9 NVRAM

## 9.1 Introduction

~~Has `plist dict` type and allows to set volatile~~ This section allows setting non-volatile UEFI variables commonly ~~referred~~ described as NVRAM variables. Refer to `man nvram` for more details. ~~macOS~~ The macOS operating system extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication~~, and thus supplying several NVRAM~~. Hence, the supply of several NVRAM variables is required for ~~proper macOS functioning~~ the proper functioning of macOS.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' the NVRAM variable belongs to. ~~macOS uses~~ The macOS operating system makes use of 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 the PlatformNVRAM or Generic subsections of the PlatformInfo section. Please ensure that variables ~~of this section never collide with them, as~~ set in this section do not conflict with items in those subsections as the implementation behaviour is undefined otherwise.

~~For proper macOS functioning it is often required to use~~ The `OC_FIRMWARE_RUNTIME` protocol implementation, currently offered as a part of the `OpenRuntime` driver~~. While it brings any~~, is often required for macOS to function properly. While this brings many benefits, there are ~~certain limitations which arise depending on the use~~ some limitations that should be considered for certain use cases.

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~~, tools must be aware of the `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 multidata` format. GUIDs must be provided in canonic string format in upper or lower case (e.g. `8BE4DF61-93CA-11D2-AA0D-00E098032B8C`).

   ~~Created variables get~~ The `EFI_VARIABLE_BOOTSERVICE_ACCESS` and `EFI_VARIABLE_RUNTIME_ACCESS` attributes of created variables are set. Variables will only be set if not present or deleted. ~~I.e.~~ That is, to overwrite an existing variable value, add the variable name to the `Delete` section. This approach enables ~~to provide default values till~~ the provision of default values until the operating system takes the lead.

   *Note*: ~~If~~ The implementation behaviour is undefined when the `plist key` does not conform to ~~GUID format, behaviour is undefined~~the GUID format.

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~~ a NVRAM variable file named `nvram.plist` from EFI volume root.

   This file must have a 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 the `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 the OpenCore EFI partition UUID.

**Warning**: This feature ~~is very dangerous~~can be dangerous, as it passes unprotected data to firmware variable services. ~~Use it only~~ Only use when no hardware NVRAM implementation is provided by the firmware or ~~it~~ when the NVRAM implementation 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 certain NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

   `*` value can be used to accept all variables for certain GUID.

   **WARNING**: Choose variables ~~very~~ carefully, as the nvram.plist file is not vaulted. For instance, do not ~~put~~ include `boot-args` or `csr-active-config`, as ~~this can~~ these can be used to bypass SIP.

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

   *Note*: ~~It is recommended to have this value~~ This value should be enabled on most types of firmware but ~~it~~ is left configurable to account for firmware that may have issues with NVRAM variable storage garbage collection or similar.

~~To~~ The `nvram` command can be used to read NVRAM variable ~~value from macOS, `nvram` could be used by concatenating~~ values from macOS by concatenating the GUID and name variables separated by a `:` symbol. For example, `nvram 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args`.

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 the PlatformNVRAM or Generic subsections of the PlatformInfo section. Using `PlatformInfo` is the recommended 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:

  - `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.
    * `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)
  - `debug=VALUE` — deprecated starting from 10.15.
    * `1` — enables print something to BOOTER.LOG (stripped code implies there may be a crash)
    * `2` — enables perf logging to /efi/debug-log in the device three
    * `4` — enables timestamp printing for styled printf calls
  - `level=VALUE` — deprecated starting from 10.15. Verbosity level of DEBUG output. Everything but `0x80000000` is stripped from the binary, and this is the default value.

  *Note*: ~~To see~~ Enable the `AppleDebug` option to display verbose output from `boot.efi` on modern macOS versions~~enable `AppleDebug` option~~. This will save the log to the general OpenCore log file. For versions before 10.15.4, set `bootercfg` to `log=1`. This will print verbose output onscreen.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg-once`
  Booter arguments override removed after first launch. Otherwise equivalent to `bootercfg`.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:efiboot-perf-record`
  Enable performance log saving in `boot.efi`. Performance log is saved to physical memory and is pointed ~~by~~ to by the `efiboot-perf-record-data` and `efiboot-perf-record-size` variables. Starting from 10.15.4, it can also be saved to the OpenCore log by setting the `AppleDebug` option.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:fmm-computer-name`
  Current saved host name. ASCII string.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:nvda_drv`
  NVIDIA Web Driver control variable. Takes ASCII digit `1` or `0` to enable or disable installed driver.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:run-efi-updater`
  Override EFI firmware updating support in macOS (MultiUpdater, ThorUtil, and so on). Setting this to `No` or alternative boolean-castable value will prevent any firmware updates in macOS starting with 10.10 at least.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:StartupMute`
  Mute startup chime sound in firmware audio support. 8-bit integer. The value of `0x00` means unmuted. Missing variable or any other value means muted.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:SystemAudioVolume`
  System audio volume level for firmware audio support. 8-bit integer. The bit of `0x80` means muted. Lower bits are used to encode volume range specific to installed audio codec. The value is capped by `MaximumBootBeepVolume` AppleHDA layout value to avoid too loud audio playback in the firmware.

# 10 PlatformInfo

Platform information ~~is comprised~~ consists 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 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 all the values are specified (the default), and semi-automatic, where (`Automatic`) only certain values are specified, and later used for system configuration.

~~To inspect SMBIOS contents~~ The dmidecode utility can be used ~~. Version~~ to inspect SMBIOS contents and a 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 the `Generic` section instead of using values from the `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 to set~~ Setting this option to `false` is strongly discouraged when intending to update platform information. ~~The only reason to do so is if making~~ A `false` setting is typically only valid for minor corrections to SMBIOS values on legacy Apple hardware. In all other cases, setting `Automatic` to `false` may lead to ~~hard to debug errors , due to~~ hard-to-debug errors resulting from inconsistent or invalid settings.

2. `CustomMemory`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Use custom memory configuration defined in the `Memory` section. This completely replaces any existing memory configuration in SMBIOS, and is only active when `UpdateSMBIOS` is set to `true`.

3. `UpdateDataHub`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Update Data Hub fields. These fields are read from the `Generic` or `DataHub` sections depending on the setting of the `Automatic` ~~value~~property.

   *Note*: The implementation of the Data Hub protocol in EFI firmware on ~~essentially~~ virtually all systems, including Apple hardware, means that existing Data Hub entries cannot be overridden~~, while new~~. New entries are added to the end of the Data Hub instead, with macOS ignoring ~~them~~old entries. This can be worked around by ~~reinstalling~~ replacing the Data Hub protocol using the `ProtocolOverrides` section. Refer to the `DataHub` protocol override description for details.

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

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

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

5. `UpdateSMBIOS`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Update SMBIOS fields. These fields are read from the `Generic` or `SMBIOS` sections depending on the setting of the `Automatic` ~~value~~property.

6. `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 on some types of firmware.
   - `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 firmware 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 the `Custom` approach ~~is making~~ that it makes SMBIOS updates exclusive to macOS, avoiding a collision with existing Windows activation and custom OEM software but potentially obstructing the operation of Apple-specific tools.

7. `UseRawUuidEncoding`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Use raw encoding for SMBIOS UUIDs.

   Each UUID `AABBCCDD-EEFF-GGHH-IIJJ-KKLLMMNNOOPP` is essentially a hexadecimal 16-byte number. It can be encoded in two ways:

   - `Big Endian` — by writing all the bytes as they are without making any order changes (`{AA BB CC DD EE FF GG HH II JJ KK LL MM NN OO PP}`). This method is also known as RFC 4122 encoding or `Raw` encoding.
   - `Little Endian` — by interpreting the bytes as numbers and using Little Endian byte representation (`{DD CC BB AA FF EE HH GG II JJ KK LL MM NN OO PP}`).

   The SMBIOS specification did not explicitly specify the encoding format for the UUID up to SMBIOS 2.6, where it stated that `Little Endian` encoding shall be used. This led to the confusion in both firmware implementations and system software as different vendors used different encodings prior to that.

   - Apple uses the `Big Endian` format everywhere but it ignores SMBIOS UUID within macOS.
   - `dmidecode` uses the `Big Endian` format for SMBIOS 2.5.x or lower and the `Little Endian` format for 2.6 and newer. Acidanthera dmidecode prints all ~~the~~ three.
   - Windows uses the `Little Endian` format everywhere, but ~~it~~ this only affects the visual representation of the values.

   OpenCore always sets a recent SMBIOS version (currently 3.2) when generating the modified DMI tables. If `UseRawUuidEncoding` is enabled, ~~then~~ the `Big Endian` format is used to store the `SystemUUID` data. Otherwise, the `Little Endian` format is used.

   *Note*: ~~Since~~ This preference does not affect UUIDs used in DataHub and NVRAM as they are not standardised and are added by Apple~~, this preference does not affect them~~. Unlike SMBIOS, they are always stored in the `Big Endian` format.

8. `Generic`
   **Type**: `plist dictionary`
   **Description**: Update all fields in `Automatic` mode.

*Note*: This section is ignored but may not be removed when `Automatic` is `false`.

9. `DataHub`
   **Type**: `plist dictionary`
   **Description**: Update Data Hub fields in non-`Automatic` mode.

   *Note*: This section is ignored and may be removed when `Automatic` is `true`.

10. `Memory`
    **Type**: `plist dictionary`
    **Description**: Define custom memory configuration.

    *Note*: This section is ignored and may be removed when `CustomMemory` is `false`.

11. `PlatformNVRAM`
    **Type**: `plist dictionary`
    **Description**: Update platform NVRAM fields in non-`Automatic` mode.

    *Note*: This section is ignored and may be removed when `Automatic` is `true`.

12. `SMBIOS`
    **Type**: `plist dictionary`
    **Description**: Update SMBIOS fields in non-`Automatic` mode.

    *Note*: This section is ignored and may be removed when `Automatic` is `true`.

## 10.2 Generic Properties

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

   It ~~is~~ can be dangerous to use ~~Apple~~ "Apple" in SMBIOS vendor fields for reasons ~~given~~ outlined in the `SystemManufacturer` description. However, certain firmware may not provide valid values otherwise, which could obstruct the operation of 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~~ an EFI partition that is 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~~ an EFI partition that is the first partition on the disk.

3. `MaxBIOSVersion`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Sets `BIOSVersion` to `9999.999.999.999.999`, recommended for legacy Macs when using `Automatic` PlatformInfo, to avoid BIOS updates in unofficially supported macOS versions.

4. `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.

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

This value contains CPU ART frequency, also known as crystal clock frequency. Its existence is exclusive to the Skylake generation and newer. The value is specified in Hz, and is normally 24 MHz for the client Intel segment, 25 MHz for the 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**: `0` (Not installed)
**Description**: Sets `DevicePathsSupported` in `gEfiMiscSubClassGuid`. Must be set to `1` for AppleACPIPlatform.kext to append SATA device paths to `Boot####` and `efi-boot-device-data` variables. Set to `1` on all modern Macs.

12. `SmcRevision`
**Type**: `plist data`, 6 bytes
**Failsafe**: Empty (Not installed)
**Description**: Sets `REV` in `gEfiMiscSubClassGuid`. Custom property read by `VirtualSMC` or `FakeSMC` to generate SMC `REV` key.

13. `SmcBranch`
**Type**: `plist data`, 8 bytes
**Failsafe**: Empty (Not installed)
**Description**: Sets `RBr` in `gEfiMiscSubClassGuid`. Custom property read by `VirtualSMC` or `FakeSMC` to generate SMC `RBr` key.

14. `SmcPlatform`
**Type**: `plist data`, 8 bytes
**Failsafe**: Empty (Not installed)
**Description**: Sets `RPlt` in `gEfiMiscSubClassGuid`. Custom property read by `VirtualSMC` or `FakeSMC` to generate SMC `RPlt` key.

## 10.4 Memory Properties

1. `DataWidth`
**Type**: `plist integer`, 16-bit
**Failsafe**: `0xFFFF` (unknown)
**SMBIOS**: Memory Device (Type 17) — Data Width
**Description**: Specifies the data width, in bits, of the memory. A `DataWidth` of `0` and a `TotalWidth` of `8` indicates that the device is being used solely to provide 8 error-correction bits.

2. `Devices`
**Type**: `plist array`
**Failsafe**: Empty
**Description**: Specifies the custom memory devices to be added.

Designed to be filled with `plist dictionary` values, describing each memory device. See the Memory Devices Properties section below. This should include all memory slots, even if unpopulated.

3. `ErrorCorrection`
**Type**: `plist integer`, 8-bit
**Failsafe**: `0x03`
**SMBIOS**: Physical Memory Array (Type 16) — Memory Error Correction
**Description**: Specifies the primary hardware error correction or detection method supported by the memory.

- `0x01` — Other
- `0x02` — Unknown
- `0x03` — None

- `0x04` — Parity
- `0x05` — Single-bit ECC
- `0x06` — Multi-bit ECC
- `0x07` — CRC

4. `FormFactor`
   **Type**: `plist integer`, 8-bit
   **Failsafe**: `0x02`
   **SMBIOS**: Memory Device (Type 17) — Form Factor
   **Description**: Specifies the form factor of the memory. On Macs, this should typically be DIMM or SODIMM. Commonly used form factors are listed below.

   When `CustomMemory` is `false`, this value is automatically set based on Mac product name.

   When `Automatic` is `true`, the original value from the the corresponding Mac model will be set if available. Otherwise, the value from `OcMacInfoLib` will be set. When `Automatic` is `false`, a user-specified value will be set if available. Otherwise, the original value from the firmware will be set. If no value is provided, the fallback value (`zero`) will be set.

   - `0x01` — Other
   - `0x02` — Unknown
   - `0x09` — DIMM
   - `0x0D` — SODIMM
   - `0x0F` — FB-DIMM

5. `MaxCapacity`
   **Type**: `plist integer`, 64-bit
   **Failsafe**: `0`
   **SMBIOS**: Physical Memory Array (Type 16) — Maximum Capacity
   **Description**: Specifies the maximum amount of memory, in bytes, supported by the system.

6. `TotalWidth`
   **Type**: `plist integer`, 16-bit
   **Failsafe**: `0xFFFF` (unknown)
   **SMBIOS**: Memory Device (Type 17) — Total Width
   **Description**: Specifies the total width, in bits, of the memory, including any check or error-correction bits. If there are no error-correction bits, this value should be equal to `DataWidth`.

7. `Type`
   **Type**: `plist integer`, 8-bit
   **Failsafe**: `0x02`
   **SMBIOS**: Memory Device (Type 17) — Memory Type
   **Description**: Specifies the memory type. Commonly used types are listed below.

   - `0x01` — Other
   - `0x02` — Unknown
   - `0x0F` — SDRAM
   - `0x12` — DDR
   - `0x13` — DDR2
   - `0x14` — DDR2 FB-DIMM
   - `0x18` — DDR3
   - `0x1A` — DDR4
   - `0x1B` — LPDDR
   - `0x1C` — LPDDR2
   - `0x1D` — LPDDR3
   - `0x1E` — LPDDR4

8. `TypeDetail`
   **Type**: `plist integer`, 16-bit
   **Failsafe**: `0x4`
   **SMBIOS**: Memory Device (Type 17) — Type Detail
   **Description**: Specifies additional memory type information.

**SMBIOS**: System Enclosure or Chassis (Type 3) — Type
**Description**: Chassis type, refer to Table 17 — System Enclosure or Chassis Types for more details.

20. `ChassisVersion`
    **Type**: `plist string`
    **Failsafe**: Empty (OEM specified)
    **SMBIOS**: System Enclosure or Chassis (Type 3) — Version
    **Description**: Should match `BoardProduct`.

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

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

23. `PlatformFeature`
    **Type**: `plist integer`, 32-bit
    **Failsafe**: `0xFFFFFFFF` (OEM specified on Apple hardware, do not provide the table otherwise)
    **SMBIOS**: `APPLE_SMBIOS_TABLE_TYPE133` - `PlatformFeature`
    **Description**: Platform features bitmask. Refer to AppleFeatures.h for more details. Missing on older Macs.

24. `SmcVersion`
    **Type**: `plist data`, 16 bytes
    **Failsafe**: All zero (OEM specified on Apple hardware, do not provide the table otherwise)
    **SMBIOS**: `APPLE_SMBIOS_TABLE_TYPE134` - `Version`
    **Description**: ASCII string containing SMC version in upper case. Missing on T2 based Macs.

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

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

27. `ProcessorType`
    **Type**: `plist integer`, 16-bit
    **Failsafe**: `0` (Automatic)
    **SMBIOS**: `APPLE_SMBIOS_TABLE_TYPE131` - `ProcessorType`
    **Description**: Combined of Processor Major and Minor types.

    Automatic value generation ~~tries to provide~~ attempts to provide the most accurate value for the currently installed CPU. When this fails~~please make sure to create~~, please raise 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~~, refer to the Apple SMBIOS definitions header here.

# 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~~ loading additional UEFI modules ~~and/or apply tweaks for~~ as well as applying tweaks to 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 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 firmware 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 firmware. For Sandy Bridge and earlier CPUs, the ExFatDxeLegacy driver should be used due to the lack of RDRAND instruction support. |
| HfsPlus | Recommended. Proprietary HFS file system driver with bless support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the 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 types of firmware starting with the Ivy Bridge generation. Some applications with GUI, such as UEFI Shell, may need this driver to work properly. |
| EnhancedFatDxe | FAT filesystem driver from FatPkg. This driver is embedded in all UEFI firmware and cannot be used from OpenCore. Several types of firmware have defective FAT support implementation that may lead to corrupted filesystems on write attempts. Embedding this driver within the firmware may be required in case writing to the EFI partition is needed during the boot process. |
| NvmExpressDxe* | NVMe support driver from MdeModulePkg. This driver is included in most firmware starting with the 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 support for 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. |
| OpenPartitionDxe* | Partition management driver with Apple Partitioning Scheme support. This driver can be used to support loading older DMG recoveries such as macOS 10.9 using Apple Partitioning Scheme. OpenDuet already includes this driver. |
| Ps2KeyboardDxe* | PS/2 keyboard driver from MdeModulePkg. OpenDuetPkg and some types of firmware 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 firmware may not include this driver but it is necessary for the touchpad to work in UEFI graphical interfaces such as OpenCanopy. |
| OpenHfsPlus* | HFS file system driver with bless support. This driver is an alternative to a closed source HfsPlus driver commonly found in Apple firmware. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit. |
| UsbMouseDxe* | USB mouse driver from MdeModulePkg. Some virtual machine firmware such as OVMF may not include this driver but it is necessary for the mouse to work in UEFI graphical interfaces such as OpenCanopy. |
| XhciDxe* | XHCI USB controller support driver from MdeModulePkg. This driver is included in most types of firmware starting with the Sandy Bridge generation. For earlier firmware 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) the same command 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
```

formats are MP3 and WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: `[audio type]_[audio localisation]_[audio path].[audio ext]`. For unlocalised files filename does not include the language code and looks as follows: `[audio type]_[audio path].[audio ext]`. Audio extension can either be `mp3` or `wav`.

- Audio type can be `OCEFIAudio` for OpenCore audio files or `AXEFIAudio` for macOS bootloader audio files.
- Audio localisation is a two letter language code (e.g. `en`) with an exception for Chinese, Spanish, and Portuguese. Refer to `APPLE_VOICE_OVER_LANGUAGE_CODE` definition for the list of all supported localisations.
- Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to `APPLE_VOICE_OVER_AUDIO_FILE` definition. For OpenCore audio paths refer to `OC_VOICE_OVER_AUDIO_FILE` definition. The only exception is OpenCore boot chime file, which is `OCEFIAudio_VoiceOver_Boot.mp3`.

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

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

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

   *Note*: Some types of firmware, particularly those made by Apple, only connect the boot drive to speed up the boot process. Enable this option to be able to see all the boot options when running multiple drives.

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

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

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

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

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

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

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

9. `ReservedMemory`
   **Type**: `plist array`
   **Description**: Designed to be filled with `plist dict` values, describing memory areas ~~exquisite to particular~~ exclusive to specific firmware and hardware functioning, which should not be used by the operating system. ~~An example~~ Examples of such memory ~~region could be~~ regions could be the second 256 MB corrupted by the Intel HD 3000 or an area with faulty RAM. See the ReservedMemory Properties section below.

## 11.7 APFS Properties

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

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

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

   ~~Instead of~~ Every handle is connected recursively instead of the partition handle connection ~~normally~~ typically used for APFS driver loading~~every handle is connected recursively~~. This may ~~take more time than usual but can~~ result in additional time being taken but can sometimes be the only way to access APFS partitions on ~~some types of firmware~~certain firmware, such as those 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~~ Permits APFS USB hot plug which enables loading APFS drivers, both at OpenCore startup ~~but also during the OpenCore picker . This permits APFS USB hot plug.~~ and during OpenCore picker dusplay. Disable if not required.

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

   The APFS driver date connects the APFS driver with the calendar release date. ~~Older versions of APFS drivers may contain unpatched vulnerabilities , which~~ Apple ultimately drops support for older macOS releases and APFS drivers from such releases may contain vulnerabilities that can be used to ~~inflict harm to the computer~~ compromise a computer if such drivers are used after support ends. This option permits restricting APFS drivers to ~~only recent releases~~current macOS versions.

   - `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.

The APFS driver version connects the APFS driver with the macOS release. ~~APFS drivers from~~ Apple ultimately drops support for older macOS releases ~~will become unsupported and thus may contain unpatched vulnerabilities , which~~ and APFS drivers from such releases may contain vulnerabilities that can be used to ~~inflict harm to the computer~~ compromise a computer if such drivers are used after support ends. This option permits restricting APFS drivers to ~~only modern~~ current 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`.

## 11.8 Audio Properties

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

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

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

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

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

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

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

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

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

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

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

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

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

**Failsafe**: `false`
**Description**: Activate audio support by connecting to a backend driver.

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

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

~~Screen~~ The screen reader will use this volume level ~~,~~ when the calculated volume level is ~~less~~ lower than `MinimumVolume` ~~. Boot chime sound~~ and the boot chime will not play if the calculated volume level is ~~less~~ lower than `MinimumVolume`.

6. `PlayChime`
**Type**: `plist string`
**Failsafe**: `Auto`
**Description**: Play chime sound at startup.

Enabling this setting plays ~~boot chime through~~ the boot chime using the builtin audio support. ~~Volume~~ The volume level is determined by the `MinimumVolume` and `VolumeAmplifier` settings ~~and~~ as well as the `SystemAudioVolume` NVRAM variable. Possible values include:

- `Auto` — Enables chime when `StartupMute` NVRAM variable is not present or set to `00`.
- `Enabled` — Enables chime unconditionally.
- `Disabled` — Disables chime unconditionally.

*Note*: `Enabled` can be used in separate from `StartupMute` NVRAM variable to avoid conflicts when the firmware is able to play the boot chime.

7. `ResetTrafficClass`
**Type**: `plist boolean`
**Failsafe**: `false`
**Description**: Set HDA Traffic Class Select Register to `TC0`.

AppleHDA kext will function correctly only if `TCSEL` register is configured to use `TC0` traffic class. Refer to Intel I/O Controller Hub 9 (ICH9) Family Datasheet (or any other ICH datasheet) for more details about this register.

*Note*: This option is independent from `AudioSupport`. If AppleALC is used it is preferred to use AppleALC `alctsel` property instead.

8. `SetupDelay`
**Type**: `plist integer`
**Failsafe**: `0`
**Description**: Audio codec reconfiguration delay in microseconds.

Some codecs require a vendor-specific delay after the reconfiguration (e.g. volume setting). This option makes it configurable. A typical delay can be up to 0.5 seconds.

9. `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 such as the 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~~ setting this timeout based on the platform. The recommended value ~~that works on~~ for 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 a slightly higher value on slower platforms for more responsive input.

   Pressing keys one after the other results in delays of at least `6` and `10` milliseconds for the same platforms.

   *Note*: Some platforms may require different values, which may be 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~~ prudent to check every value from `1` to `25`.

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

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

4. `KeySupportMode`
   **Type**: `plist string`
   **Failsafe**: `Auto`
   **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.

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

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

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

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

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

   Currently the only supported variant is `ASUS`, using specialised protocol available on certain Z87 and Z97 ASUS boards. More details can be found in `LongSoft/UefiTool#116`. The value of this property cannot be empty if `PointerSupport` is enabled.

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

   This option allows ~~to update~~ updating the firmware architecture timer period with the specified value in `100` nanosecond units. Setting a lower value ~~generally~~ typically 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`. In case of issues, this option can be left as `0`.

## 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 support a different set of options. It is recommended to use `Builtin` renderer, as it supports HiDPI mode and uses full screen resolution.

   UEFI firmware ~~generally~~ typically supports `ConsoleControl` with two rendering modes: `Graphics` and `Text`. Some types of firmware do not support `ConsoleControl` and rendering modes. OpenCore and macOS expect text to only be shown in `Graphics` mode and graphics to be drawn in any mode. Since this is not required by UEFI specification, exact behaviour varies.

   Valid values are combinations of text renderer and rendering mode:

   - `BuiltinGraphics` — Switch to `Graphics` mode and use `Builtin` renderer with custom `ConsoleControl`.
   - `BuiltinText` — Switch to `Text` mode and use `Builtin` renderer with custom `ConsoleControl`.
   - `SystemGraphics` — Switch to `Graphics` mode and use `System` renderer with custom `ConsoleControl`.
   - `SystemText` — Switch to `Text` mode and use `System` renderer with custom `ConsoleControl`.
   - `SystemGeneric` — Use `System` renderer with system `ConsoleControl` assuming it behaves correctly.

   The use of `BuiltinGraphics` is ~~generally~~ straightforward. For most platforms, it is necessary to enable `ProvideConsoleGop` ~~,~~ and set `Resolution` to `Max`. The `BuiltinText` variant is an alternative `BuiltinGraphics` for some very old and defective laptop firmware, which can only draw in `Text` mode.

   The use of `System` protocols is more complicated. Typically, the preferred setting is `SystemGraphics` or `SystemText`. Enabling `ProvideConsoleGop`, setting `Resolution` to `Max`, enabling `ReplaceTabWithSpace` is useful on almost all platforms. `SanitiseClearScreen`, `IgnoreTextInGraphics`, and `ClearScreenOnModeSwitch` are more specific, and their use depends on the firmware.

   *Note*: Some Macs, such as the `MacPro5,1`, may have incompatible console output when using modern GPUs, and thus only `BuiltinGraphics` may work for them in such cases. NVIDIA GPUs may require additional firmware upgrades.

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

**Description**: Some types of firmware output text onscreen in both graphics and text mode. This is typically unexpected as random text may appear over graphical images and cause UI corruption. Setting this option to `true` will discard all text output when console control is in a different mode from `Text`.

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

9. `ReplaceTabWithSpace`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Some types of firmware do not print tab characters or everything that follows them, causing difficulties in using the UEFI Shell's builtin text editor to edit property lists and other documents. This option makes the console output spaces instead of tabs.

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

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

    macOS bootloader requires GOP or UGA (for 10.4 EfiBoot) to be present on console handle, yet the exact location of the graphics protocol is not covered by the UEFI specification. This option will ensure GOP and UGA, if present, are available on the console handle.

    *Note*: This option will also replace incompatible implementations of GOP on the console handle, as may be the case on the `MacPro5,1` when using modern GPUs.

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

    On certain firmware, the controllers that produce the console protocols (simple text out) must be reconnected when the screen resolution is changed via GOP. Otherwise, they will not produce text based on the new resolution.

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

12. `SanitiseClearScreen`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Some types of firmware reset screen resolutions to a failsafe value (such as `1024x768`) on the attempts to clear screen contents when large display (e.g. 2K or 4K) is used. This option attempts to apply a workaround.

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

13. `UgaPassThrough`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Provide UGA protocol instances on top of GOP protocol instances.

    Some types of firmware do not implement the legacy UGA protocol but this may be required for screen output by older EFI applications such as EfiBoot from 10.4.

## 11.11 ProtocolOverrides Properties

1. `AppleAudio`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces Apple audio protocols with builtin versions.

Apple audio protocols allow ~~macOS bootloader and OpenCore~~ OpenCore and the macOS bootloader to play sounds and signals for screen reading or audible error reporting. Supported protocols are beep generation and VoiceOver. The VoiceOver protocol is specific to Gibraltar machines (T2) and is not supported before macOS High Sierra (10.13). ~~Instead older~~ Older macOS versions use ~~AppleHDA protocol, which is currently not implemented~~the AppleHDA protocol (which is not currently implemented) instead.

Only one set of audio protocols can be available at a time, so this setting should be enabled in order to ~~get~~ enable audio playback in the OpenCore user interface on Mac ~~system~~ systems implementing some of these protocols~~this setting should be enabled~~.

*Note*: ~~Backend~~ The backend audio driver needs to be configured in `UEFI Audio` section for these protocols to be able to stream audio.

2. `AppleBootPolicy`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces the Apple Boot Policy protocol with a builtin version. This may be used to ensure APFS compatibility on VMs and legacy Macs.

   *Note*: This option is advisable on certain Macs, such as the `MacPro5,1`, that are APFS compatible but on which the Apple Boot Policy protocol has recovery detection issues.

3. `AppleDebugLog`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces the Apple Debug Log protocol with a builtin version.

4. `AppleEvent`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces the Apple Event protocol with a builtin version. This may be used to ensure FileVault 2 compatibility on VMs and legacy Macs.

5. `AppleFramebufferInfo`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces the Apple Framebuffer Info protocol with a builtin version. This may be used to override framebuffer information on VMs and legacy Macs to improve compatibility with legacy EfiBoot such as the one in macOS 10.4.

   *Note*: The current implementation of this property results in it only being active when GOP is available (it is always equivalent to `false` otherwise).

6. `AppleImageConversion`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces the Apple Image Conversion protocol with a builtin version.

7. `AppleImg4Verification`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces the Apple IMG4 Verification protocol with a builtin version. This protocol is used to verify `im4m` manifest files used by Apple Secure Boot.

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

9. `AppleRtcRam`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Replaces the Apple RTC RAM protocol with a builtin version.

*Note*: Builtin version of Apple RTC RAM protocol may filter out I/O attempts to certain RTC memory addresses. The list of addresses can be specified in `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:rtc-blacklist` variable as a data array.

10. `AppleSecureBoot`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Replaces the Apple Secure Boot protocol with a builtin version.

11. `AppleSmcIo`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Replaces the Apple SMC I/O protocol with a builtin version.

    This protocol replaces the legacy `VirtualSmc` UEFI driver, and is compatible with any SMC kernel extension. However, in case `FakeSMC` kernel extension is used, manual NVRAM key variable addition may be needed.

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

13. `DataHub`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Replaces the Data Hub protocol with a builtin version.

    *Note*: This will discard all previous entries if the protocol was already installed, so all properties required for the safe operation of the system must be specified in ~~your configuration~~ the configuration file.

14. `DeviceProperties`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Replaces the Device Property protocol with a builtin version. This may be used to ensure full compatibility on VMs and legacy Macs.

    *Note*: This will discard all previous entries if the protocol was already installed, so all properties required for safe operation of the system must be specified in ~~your configuration~~ the configuration file.

15. `FirmwareVolume`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Wraps Firmware Volume protocols, or installs a new version, to support custom cursor images for FileVault 2. Set to `true` to ensure FileVault 2 compatibility on anything other than on VMs and legacy Macs.

    *Note*: Several virtual machines~~including VMware~~, including VMware, may have corrupted cursor images in HiDPI mode and thus, may also require enabling this setting.

16. `HashServices`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Replaces Hash Services protocols with builtin versions. Set to `true` to ensure FileVault 2 compatibility on platforms with defective SHA-1 hash implementations. This can be determined by an invalid cursor size when `UIScale` is set to `02`. Platforms earlier than APTIO V (Haswell and older) are typically affected.

17. `OSInfo`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Replaces the OS Info protocol with a builtin version. This protocol is typically used by the firmware and other applications to receive notifications from the macOS bootloader.

18. `UnicodeCollation`
    **Type**: `plist boolean`
    **Failsafe**: `false`

# 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~~ are sometimes necessary for various reasons. While a compatible board identifier and CPUID are the obvious requirements for proper functioning of an older operating system, there are many other less obvious things to consider. This section ~~tries to cover~~ covers a common set of issues relevant to installing older macOS operating systems.

While newer operating systems can be downloaded over the internet, older operating systems did not have installation media for every minor release. For compatible distributions of such, download a device-specific image and modify it if necessary. Visit this archived Apple Support article for a list of the bundled device-specific builds for legacy operating systems. However, as this may not always be accurate, the latest versions are listed below.

### 12.1.1 macOS 10.8 and 10.9

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

### 12.1.2 macOS 10.7

- All previous issues apply.
- `SSSE3` support (not to be confused with `SSE3` support) is a hard requirement for macOS 10.7 kernel.
- Many kexts, including `Lilu` when 32-bit kernel is used and a lot of `Lilu` 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 firmware that utilise lower memory for their own purposes. Refer to acidanthera/bugtracker#1125 for tracking.

### 12.1.3 macOS 10.6

- All previous issues apply.
- `SSSE3` support is a requirement for macOS 10.6 kernel with 64-bit userspace enabled. This limitation can mostly be lifted by enabling the `LegacyCommpage` quirk.
- 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 (MEGA Mirror), assuming macOS 10.6 is legally owned. Read `DIGEST.txt` for more details. Note that these are the earliest tested versions of macOS 10.6 with OpenCore.

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
```

6. Sign all the installed drivers and tools with the private key. Do not sign tools that provide administrative access to the computer, such as UEFI Shell.
7. Vault the configuration as explained Vaulting section.
8. Sign all OpenCore binaries (`BOOTX64.efi`, `BOOTIa32.efi`, `OpenCore.efi`, custom launchers) used on this system with the same private key.
9. Sign all third-party operating system (not made by Microsoft or Apple) bootloaders if 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 firmware preferences and install the certificate with a private key. Details on how to generate a certificate can be found in various articles, such as this one, and are out of the scope of this document. If Windows is needed one will also need to add the Microsoft Windows Production CA 2011. To launch option ROMs or to use signed Linux drivers, 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 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, such as Windows 7, might work with some extra precautions. Things to 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 aware that it may be invalid on old firmware, i.e., not random. 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 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 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`. If there is a previous version of Boot Camp installed 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, 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 is typically not required).
- To access Apple filesystems such as HFS+ and APFS, separate software may need to be installed. Some of the known utilities are: Apple HFS+ driver (workaround 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 the Boot Camp Startup Disk control panel?

The Boot Camp control panel uses the GPT partition table to obtain each boot option name. After installing Windows separately~~the partition will have~~, the partition has to be relabelled manually. This can be done with many utilities including the 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~~ do I 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 disrupt certain macOS functionality, including the 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, version 7.3 or newer is needed, and Debugging the XNU Kernel with IDA Pro may also help.

To obtain the log during boot 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 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, 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 `debug=0x8` boot argument is needed.

## 12.5 Tips and Tricks

1. **How ~~to~~ do I ~~debug boot failure~~failures?**

   ~~Normally it is enough to obtain~~ Obtaining the actual error message is usually adequate. For this, ensure that:

   - 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, such as `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 workarounds in the `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~~ do I ~~debug macOS boot failure~~failures?**

   - Refer to `boot-args` values such as `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 `debug=0x108`, `serial=5`, and `msgbuf=1048576` boot arguments are needed. Refer to the patches in Sample.plist when dying before serial init.
   - Always read the logs carefully.

3. **How ~~to~~ do I 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~~ do I 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 types of firmware deleting incompatible boot options, potentially including those created by macOS, users are strongly encouraged to use the `RequestBootVarRouting` quirk, which will preserve 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 the OpenCore picker and choose the entry, it will have a `(dmg)` suffix. Custom name may be created by providing `.contentDetails` file.

   To download recovery online macrecovery.py 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.