



OpenCore

Reference Manual (0.6.7)

[2021.02.16]

Contents

1	Introduction	3
1.1	Generic Terms	3
2	Configuration	4
2.1	Configuration Terms	4
2.2	Configuration Processing	4
2.3	Configuration Structure	5
3	Setup	6
3.1	Directory Structure	6
3.2	Installation and Upgrade	7
3.3	Contribution	8
3.4	Coding conventions	9
3.5	Debugging	10
4	ACPI	11
4.1	Introduction	11
4.2	Properties	11
4.3	Add Properties	12
4.4	Delete Properties	12
4.5	Patch Properties	13
4.6	Quirks Properties	14
5	Booter	16
5.1	Introduction	16
5.2	Properties	16
5.3	MmioWhitelist Properties	17
5.4	Patch Properties	17
5.5	Quirks Properties	18
6	DeviceProperties	23
6.1	Introduction	23
6.2	Properties	23
6.3	Common Properties	23
7	Kernel	25
7.1	Introduction	25
7.2	Properties	25
7.3	Add Properties	26
7.4	Block Properties	27
7.5	Emulate Properties	27
7.6	Force Properties	28
7.7	Patch Properties	29
7.8	Quirks Properties	31
7.9	Scheme Properties	34
8	Misc	37
8.1	Introduction	37
8.2	Properties	38
8.3	Boot Properties	38
8.4	Debug Properties	42
8.5	Security Properties	45
8.6	Entry Properties	51
9	NVRAM	52
9.1	Introduction	52

9.2	Properties	52
9.3	Mandatory Variables	53
9.4	Recommended Variables	53
9.5	Other Variables	54
10	PlatformInfo	57
10.1	Properties	57
10.2	Generic Properties	59
10.3	DataHub Properties	60
10.4	Memory Properties	62
10.5	PlatformNVRAM Properties	64
10.6	SMBIOS Properties	65
11	UEFI	69
11.1	Introduction	69
11.2	Drivers	69
11.3	Tools and Applications	71
11.4	OpenCanopy	71
11.5	OpenRuntime	72
11.6	Properties	72
11.7	APFS Properties	74
11.8	Audio Properties	75
11.9	Input Properties	76
11.10	Output Properties	78
11.11	ProtocolOverrides Properties	80
11.12	Quirks Properties	82
11.13	ReservedMemory Properties	83
12	Troubleshooting	85
12.1	Legacy Apple OS	85
12.2	UEFI Secure Boot	86
12.3	Windows support	87
12.4	Debugging	88
12.5	Tips and Tricks	89

1 Introduction

This document provides information on 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 bugs 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.

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

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 result 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 must prevent the boot process from proceeding until the next host system boot. 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 are permitted to take any measures including, but not limited to, **fatal behaviour**, assumptions of any state or value, or disregarding the state or value. 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 OpenCore bootstrapping mechanism, the presence of multiple **OC config** files may lead to the reading any of them. It is permissible that no **OC Config** file is present on disk. In such cases, all the values read shall follow the rules of **invalid values** and **optional values**.

The **OC config** file has size, nesting, and key amount limitations. The **OC config** file size shall not exceed 16 MBs. The **OC config** file shall not have more than 8 nesting levels. The **OC config** file shall have up to 16384 XML nodes (i.e. one **plist dictionary** item is counted as a pair of nodes) within each **plist object**.

Reading malformed **OC config** files result in **undefined behaviour**. Examples of malformed **OC config** files cover at least the following cases:

- files non-conformant to **plist DTD**
- files with unsupported or non-conformant **plist objects** found in this document
- files violating size, nesting, and key amount limitations

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**. The recommended practice for interpreting **invalid values** is to conform to the following convention where applicable:

Type	Value
plist string	Empty string (<string></string>)
plist data	Empty data (<data></data>)
plist integer	0 (<integer>0</integer>)
plist boolean	False (<false/>)
plist tristate	False (<false/>)

2.3 Configuration Structure

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

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

Root configuration entries consist of the following:

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

Basic validation of the `OC config` file is possible using the `ocvalidate` utility. Please note that `ocvalidate` must match the used OpenCore release and may not be able to detect all configuration issues present in the file.

Note: For system integrity, most properties typically have predefined values even if not specified in the `OC config` file. However, this behaviour should not be relied upon and all fields must be properly specified in the `OC config` file.

3 Setup

3.1 Directory Structure

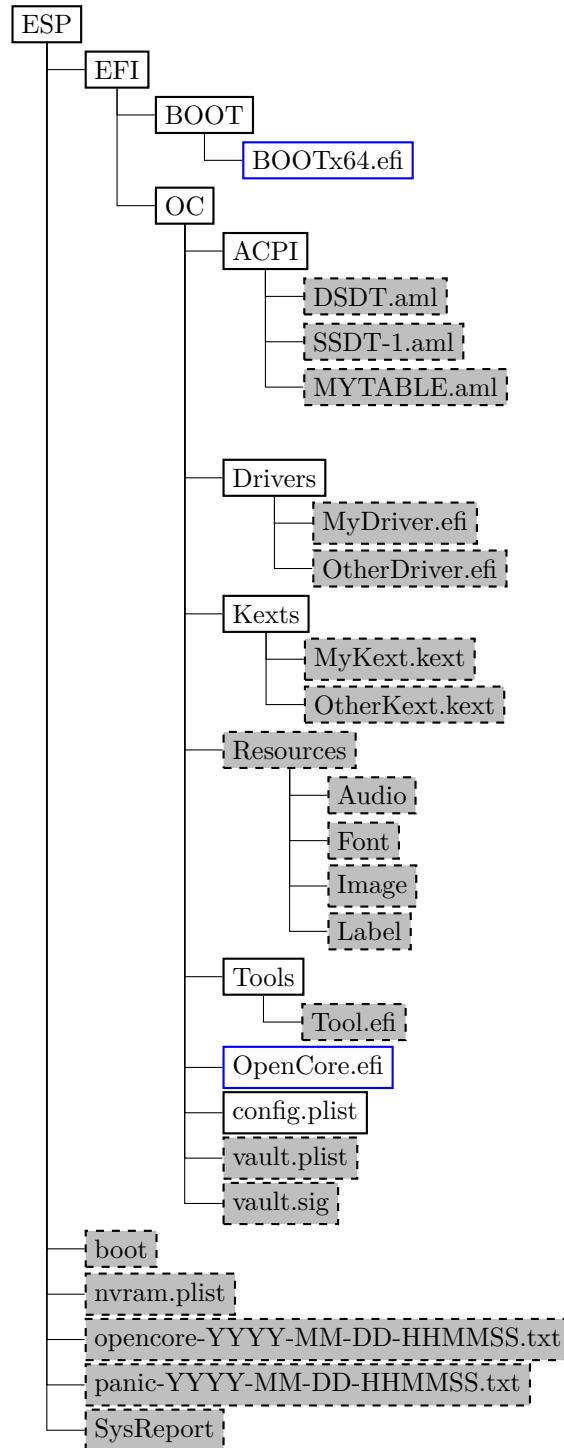


Figure 1. Directory Structure

When directory boot is used, the directory structure used should follow the descriptions in the Directory Structure figure. Available entries include:

- **BOOTx64.efi** or **BOOTIa32.efi**

Initial bootstrap loaders, which load **OpenCore.efi**. **BOOTx64.efi** is loaded by the firmware by default according to UEFI specifications. However, it can also be renamed and put in a custom location to allow OpenCore to coexist with operating systems using **BOOTx64.efi** as their own loaders (e.g. Windows). See the **LauncherOption**

property for more details.

- **boot**
Duet bootstrap loader, which initialises the UEFI environment on legacy BIOS firmware and loads `OpenCore.efi` similar to other bootstrap loaders. A modern Duet bootstrap loader will default to `OpenCore.efi` on the same partition when present.
- **ACPI**
Directory used for storing supplemental ACPI information for the `ACPI` section.
- **Drivers**
Directory used for storing supplemental UEFI drivers for UEFI section.
- **Kexts**
Directory used for storing supplemental kernel information for the `Kernel` section.
- **Resources**
Directory used for storing media resources such as audio files for screen reader support. See the `UEFI Audio Properties` section for more details. This directory also contains image files for graphical user interface. See the `OpenCanopy` section for more details.
- **Tools**
Directory used for storing supplemental tools.
- **OpenCore.efi**
Main booter application responsible for operating system loading. The directory `OpenCore.efi` resides in is called the `root` directory which is set to `EFI\OC` by default. When launching `OpenCore.efi` directly or through a custom launcher however, other directories containing `OpenCore.efi` files are also supported.
- **config.plist**
OC Config.
- **vault.plist**
Hashes for all files potentially loadable by OC Config.
- **vault.sig**
Signature for `vault.plist`.
- **SysReport**
Directory containing system reports generated by `SysReport` option.
- **nvram.plist**
OpenCore variable import file.
- **opencore-YYYY-MM-DD-HHMMSS.txt**
OpenCore log file.
- **panic-YYYY-MM-DD-HHMMSS.txt**
Kernel panic log file.

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

3.2 Installation and Upgrade

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

The `OC config` file, as with any property list file, can be edited with any text editor including nano and vim. However, specialised software may provide a better experience. On macOS, the preferred GUI application is Xcode. For a lightweight cross-platform and open-source alternative, the ProperTree editor can be utilised.

For BIOS booting, a third-party UEFI environment provider is required. `OpenDuetPkg` is a well known UEFI environment provider for legacy systems. To run OpenCore on such a legacy system, `OpenDuetPkg` can be installed with a dedicated tool — `BootInstall` (bundled with OpenCore). Third-party utilities can be used to perform this on systems other than macOS.

For upgrade purposes, refer to the `Differences.pdf` document which provides information about changes to the configuration (as compared to the previous release) as well as to the `Changelog.md` document, which contains a list of modifications across all published updates.

3.3 Contribution

OpenCore can be compiled as an ordinary EDK II package. As UDK development has been abandoned by TianoCore however, OpenCore requires the use of EDK II Stable. The currently supported EDK II release is hosted in acidanthera/audk. Required patches for this package can be found in the **Patches** directory.

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

To compile with XCODE5, besides Xcode, users should also install NASM and MTOC. The latest Xcode version is recommended for use despite the toolchain name. An example command sequence is as follows:

```
git clone --depth=1 https://github.com/acidanthera/audk UDK
cd UDK
git submodule update --init --recommend-shallow
git clone --depth=1 https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

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

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

Listing 2: ECC Configuration

Warning: Tool developers modifying `config.plist` or any other OpenCore files must ensure that their tool checks the `opencore-version` NVRAM variable (see the Debug Properties section below) and warns users if the version listed is unsupported or prerelease. The OpenCore configuration may change across releases and such, tools shall ensure they carefully follow this document. Failure to do so may result in such tools being considered to be malware and blocked by any means.

3.4 Coding conventions

As with any other project, we have conventions that we follow during development. All third-party contributors are advised to adhere to the conventions listed below before submitting patches. To minimise abortive work and the potential rejection of submissions, third-party contributors should initially raise issues to the Acidanthera Bugtracker for feedback before submitting patches.

Organisation. The codebase is contained in the `OpenCorePkg` repository, which is the primary EDK II package.

- Whenever changes are required in multiple repositories, separate pull requests should be sent to each.
- Committing the changes should happen firstly to dependent repositories, secondly to primary repositories to avoid automatic build errors.
- Each unique commit should compile with `XCORE5` and preferably with other toolchains. In the majority of the cases it can be checked by accessing the CI interface. Ensuring that static analysis finds no warnings is preferred.
- External pull requests and tagged commits must be validated. That said, commits in master may build but may not necessarily work.
- Internal branches should be named as follows: `author-name-date`, e.g. `vit9696-ballooning-20191026`.
- Commit messages should be prefixed with the primary module (e.g. library or code module) the changes were made in. For example, `OcGuardLib: Add OC_ALIGNED macro`. For non-library changes `Docs` or `Build` prefixes are used.

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

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

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

- Write inline documentation for the functions and variables only once: in headers, where a header prototype is available, and inline for **static** variables and functions.
- Use line length of 120 characters or less, preferably 100 characters.
- Use spaces after casts, e.g. `(VOID *) (UINTN) Variable`.
- Use two spaces to indent function arguments when splitting lines.
- Prefix public functions with either `Oc` or another distinct name.
- Do not prefix private **static** functions, but prefix private **non-static** functions with **Internal**.
- Use SPDX license headers as shown in [acidanthera/bugtracker#483](#).

3.5 Debugging

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

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

When trying to find the problematic change it is useful to rely on `git-bisect` functionality. There also are some unofficial resources that provide per-commit binary builds of OpenCore, such as [Dortania](#).

4 ACPI

4.1 Introduction

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

To compile and disassemble ACPI tables, the iASL compiler can be used developed by ACPICA. 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, which is not possible before the operating system boots according to the ACPI specification, operating system chainloading, and harder ACPI debugging. For this reason it may be required to carefully use `_OSI` method when writing the changes.

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

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 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.

4.3 Add Properties

1. **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.
2. **Enabled**
Type: plist boolean
Failsafe: false (Do not add this ACPI table)
Description: Set to true to add this ACPI table.
3. **Path**
Type: plist string
Failsafe: Empty
Description: File paths meant to be loaded as ACPI tables. Example values include `DSDT.aml`, `SubDir/SSDT-8.aml`, `SSDT-USBX.aml`, etc.

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

Note: All tables apart from tables with a DSDT table identifier (determined by parsing data, not by filename) insert new tables into the ACPI stack. DSDT tables perform a replacement of DSDT tables instead.

4.4 Delete Properties

1. **All**
Type: plist boolean
Failsafe: false (Only delete the first matched table)
Description: Set to true to delete all ACPI tables matching the condition.
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 (Do not remove this ACPI table)
Description: Set to true to remove this ACPI table.
4. **OemTableId**
Type: plist data, 8 bytes
Failsafe: All zero (Match any table OEM ID)
Description: Match table OEM ID equal to this value.
5. **TableLength**
Type: plist integer
Failsafe: 0 (Match any table size)
Description: Match table size equal to this value.
6. **TableSignature**
Type: plist data, 4 bytes
Failsafe: All zero (Match any table signature)
Description: Match table signature equal to this value.

Note: Do not specify table signatures when the sequence must be replaced in multiple places. This is particularly relevant when performing different types of renames.

4.5 Patch Properties

1. 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.
2. Count
Type: plist integer
Failsafe: 0 (Apply patch to all occurrences found)
Description: Number of occurrences to patch.
3. Enabled
Type: plist boolean
Failsafe: false (Do not apply this ACPI patch)
Description: Set to true to apply this ACPI patch.
4. Find
Type: plist data
Failsafe: Empty
Description: Data to find. Must be equal to **Replace** in size if set.
5. Limit
Type: plist integer
Failsafe: 0 (Search entire ACPI table)
Description: Maximum number of bytes to search for.
6. Mask
Type: plist data
Failsafe: Empty (Ignored)
Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Must be equal to **Replace** in size if set.
7. OemTableId
Type: plist data, 8 bytes
Failsafe: All zero (Match any table OEM ID)
Description: Match table OEM ID equal to this value.
8. Replace
Type: plist data
Failsafe: Empty
Description: Replacement data of one or more bytes.
9. ReplaceMask
Type: plist data
Failsafe: Empty (Ignored)
Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Must be equal to **Replace** in size if set.
10. Skip
Type: plist integer
Failsafe: 0 (Do not skip any occurrences)
Description: Number of found occurrences to skip before replacements are applied.
11. TableLength
Type: plist integer
Failsafe: 0 (Match any table size)
Description: Match table size equal to this value.
12. TableSignature
Type: plist data, 4 bytes

Failsafe: All zero (Match any table signature)

Description: Match table signature equal to this value.

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

- Avoid renaming devices with ACPI patches. This may fail or perform improper renaming of unrelated devices (e.g. EC and ECO), 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 hacks on APTIO firmware, it typically results in a need for additional patches. Modern firmware generally do not need this and those that do are fine with much smaller patches. 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 usually 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`.
- To provide a custom method implementation within an SSDT, for instance, to inject shutdown fix on certain computers, 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. Can also fix power-button shortcuts. Not recommended unless required.

2. NormalizeHeaders

Type: plist boolean

Failsafe: false

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

3. RebaseRegions

Type: plist boolean

Failsafe: false

Description: Attempt to heuristically relocate ACPI memory regions. Not recommended.

ACPI tables are often generated dynamically by the underlying firmware implementation. Among the position-independent code, ACPI tables may contain the physical addresses of MMIO areas used for device configuration, usually 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.

For this reason, Applying any kind of modifications to ACPI tables is extremely risky. The best approach is to make as few as possible changes to ACPI and to avoid replacing any tables, especially DSDT. When this is not possible, Attempt to ensure that any custom DSDTs are based on the most recent DSDT or to remove reads and writes for the affected areas.

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

4. **ResetHwSig**

Type: plist boolean

Failsafe: false

Description: Reset FACS table **HardwareSignature** value to 0.

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

5. **ResetLogoStatus**

Type: plist boolean

Failsafe: false

Description: Reset BGRT table **Displayed** status field to **false**.

This works around firmware that provide a BGRT table but fail to handle screen updates afterwards.

5 Booter

5.1 Introduction

This section allows the application of different types of UEFI modifications to 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 a 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 be looked up first. Here is an example of a bug found in some **Z68** motherboards. To turn **Power Nap** and the others off, run the following commands in Terminal:

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

Note: These settings may be reset at hardware change 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 (Devirtualise address)

Description: Virtualise address.

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 (Do not use this booter patch)

Description: Set to `true` to implement 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 `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.

8. Mask

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

Type: plist data
Failsafe: Empty
Description: Replacement data of one or more bytes.
10. ReplaceMask

Type: plist data
Failsafe: Empty (Ignored)
Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Must be equal to `Replace` in size if set.
11. Skip

Type: plist integer
Failsafe: 0 (Do not skip any occurrences)
Description: Number of found occurrences to skip before replacements are applied.

5.5 Quirks Properties

1. AllowRelocationBlock

Type: plist boolean
Failsafe: false
Description: Allows booting macOS through a relocation block.

The relocation block is a scratch buffer allocated in the lower 4 GB used for loading the kernel and related structures by EfiBoot on firmware where the lower memory region is otherwise occupied by (assumed) non-runtime data. Right before kernel startup, the relocation block is copied back to lower addresses. Similarly, all the other addresses pointing to the relocation block are also carefully adjusted. The relocation block can be used when:

- No better slide exists (all the memory is used)
- `slide=0` is forced (by an argument or safe mode)
- KASLR (slide) is unsupported (this is macOS 10.7 or older)

This quirk requires `ProvideCustomSlide` to also be enabled and typically needs `AvoidRuntimeDefrag` to be set for this to work correctly. Hibernation is not supported when booting with a relocation block (but relocation block is not always used when the quirk is enabled).

Note: While this quirk is required to run older macOS versions on platforms with used lower memory, it is not compatible with some hardware and macOS 11. In such cases, consider using `EnableSafeModeSlide` instead.

2. AvoidRuntimeDefrag

Type: plist boolean
Failsafe: false
Description: Protect from boot.efi runtime memory defragmentation.

This option fixes UEFI runtime services (date, time, NVRAM, power control, etc.) support on firmware that uses SMM backing for some services such as variable storage. SMM may try to access physical addresses but they get moved by boot.efi.

Note: Most types of firmware, apart from Apple and VMware, need this quirk.

3. DevirtualiseMmio

Type: plist boolean
Failsafe: false
Description: Remove runtime attribute from select MMIO regions.

This option reduces stolen memory footprint from the memory map by removing runtime bit for known memory regions. This quirk may result in the increase of KASLR slides available, but is not necessarily compatible with the target board without additional measures. This typically frees between 64 and 256 megabytes of memory,

present in the debug log, and on some platforms, is the only way to boot macOS, which otherwise fails with allocation errors at the bootloader stage.

This option is generally useful on all types of firmware, apart from some very old ones such as Sandy Bridge. On some types of firmware, a list of addresses that need virtual addresses for proper NVRAM and hibernation functionality may be required. Use the `MmioWhitelist` section for this.

4. `DisableSingleUser`

Type: plist boolean

Failsafe: false

Description: Disable single user mode.

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

5. `DisableVariableWrite`

Type: plist boolean

Failsafe: false

Description: Protect from macOS NVRAM write access.

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

Note: This quirk can also be used as an ad-hoc workaround for flawed UEFI runtime services implementations that are unable to write variables to NVRAM and result in operating system failures.

6. `DiscardHibernateMap`

Type: plist boolean

Failsafe: false

Description: Reuse original hibernate memory map.

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

Note: This may be used to workaround problematic memory maps on older hardware which are now considered rare legacy. Examples of such hardware are Ivy Bridge laptops with Insyde firmware such as the Acer V3-571G. Do not use this option without a full understanding of the implications.

7. `EnableSafeModeSlide`

Type: plist boolean

Failsafe: false

Description: Patch bootloader to have KASLR enabled in safe mode.

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

Note: The need for this option is determined by the availability of safe mode and can be enabled when booting to safe mode fails.

8. `EnableWriteUnprotector`

Type: plist boolean

Failsafe: false

Description: Permit write access to UEFI runtime services code.

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

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

9. `ForceExitBootServices`

Type: plist boolean

Failsafe: false

Description: Retry `ExitBootServices` with new memory map on failure.

Ensure that `ExitBootServices` call succeeds, even with an outdated `MemoryMap` key argument if required, by obtaining the current memory map and retrying the `ExitBootServices` call.

Note: The need for this quirk is determined by early boot crashes of the firmware. Do not use this option without a full understanding of the implications.

10. `ProtectMemoryRegions`

Type: plist boolean

Failsafe: false

Description: Protect memory regions from incorrect access.

Some types of firmware incorrectly map select memory regions:

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

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

Note: The need for this quirk is determined by artifacts, sleep wake issues, and boot failures. Only very old firmware typically need this quirk.

11. `ProtectSecureBoot`

Type: plist boolean

Failsafe: false

Description: Protect UEFI Secure Boot variables from being written.

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

Note: This quirk attempts to mitigate problems on NVRAM implementations with fragmentation issues such as the `MacPro5,1` and some Insyde firmware.

12. `ProtectUefiServices`

Type: plist boolean

Failsafe: false

Description: Protect UEFI services from being overridden by the firmware.

Some modern firmware, including on virtual machines such as VMware, may update pointers to UEFI services during driver loading and related actions. Consequently, this directly breaks other quirks that affect memory management, such as `DevirtualiseMmio`, `ProtectMemoryRegions`, or `RebuildAppleMemoryMap`, and may also break other quirks depending on the scope of such.

Note: On VMware, the need for this quirk may be determined by the appearance of the “Your Mac OS guest might run unreliably with more than one virtual core.” message.

13. `ProvideCustomSlide`

Type: plist boolean

Failsafe: false

Description: Provide custom KASLR slide on low memory.

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

Note: The need for this quirk is determined by the OCABC: `Only N/256 slide values are usable!` message in the debug log.

14. **ProvideMaxSlide**

Type: plist integer

Failsafe: 0

Description: Provide maximum KASLR slide when higher ones are unavailable.

This option overrides the maximum slide of 255 by a user specified value between 1 and 254 (inclusive) when **ProvideCustomSlide** is enabled. It is assumed that modern firmware allocates pool memory from top to bottom, effectively resulting in free memory when slide scanning is used later as temporary memory during kernel loading. When such memory is not available, this option stops the evaluation of higher slides.

Note: The need for this quirk is determined by random boot failures when **ProvideCustomSlide** is enabled and the randomized slide falls into the unavailable range. When **AppleDebug** is enabled, usually the debug log may contain messages such as AAPL: [EB|'LD:LKC] } Err(0x9). To find the optimal value, append **slide=X**, where X is the slide value, to the **boot-args** and select the largest one that does not result in boot failures.

15. **RebuildAppleMemoryMap**

Type: plist boolean

Failsafe: false

Description: Generate macOS compatible Memory Map.

The Apple kernel has several limitations on parsing the UEFI memory map:

- The Memory map size must not exceed 4096 bytes as the Apple kernel maps it as a single 4K page. Since some types of firmware can have very large memory maps, potentially over 100 entries, the Apple kernel will crash on boot.
- The Memory attributes table is ignored. **EfiRuntimeServicesCode** memory statically gets RX permissions while all other memory types get RW permissions. Since some firmware drivers may write to global variables at runtime, the Apple kernel will crash at calling UEFI runtime services unless the driver **.data** section has a **EfiRuntimeServicesData** type.

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

Note 1: Since several types of firmware come with incorrect memory protection tables, this quirk often comes paired with **SyncRuntimePermissions**.

Note 2: The need for this quirk is determined by early boot failures. This quirk replaces **EnableWriteUnprotector** on firmware supporting Memory Attribute Tables (MAT). This quirk is usually unnecessary when using **OpenDuetPkg** but may be required to boot macOS 10.6, and earlier, for reasons that are as yet unclear.

16. **SetupVirtualMap**

Type: plist boolean

Failsafe: false

Description: Setup virtual memory at **SetVirtualAddresses**.

Some types of firmware access memory by virtual addresses after a **SetVirtualAddresses** call, resulting in early boot crashes. This quirk workarounds the problem by performing early boot identity mapping of assigned virtual addresses to physical memory.

Note: The need for this quirk is determined by early boot failures. However, new firmware with memory protection support (such as OVMF) do not currently support this quirk. See [acidanthera/bugtracker#719](#).

17. **SignalAppleOS**

Type: plist boolean

Failsafe: false

Description: Report macOS being loaded through OS Info for any OS.

This quirk is useful on Mac firmware, which behaves differently for different OS. For example, it is supposed to enable Intel GPU in Windows and Linux in some dual-GPU MacBook models.

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 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.

6 DeviceProperties

6.1 Introduction

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

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

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

Device properties are part of the `IODeviceTree` (`gIODT`) plane of the macOS I/O Registry. This plane has several construction stages relevant for the platform initialisation. While the early construction stage is performed by the XNU kernel in the `IODeviceTreeAlloc` method, the majority of the construction is performed by the platform expert, implemented in `AppleACPIPlatformExpert.kext`.

`AppleACPIPlatformExpert` incorporates two stages of `IODeviceTree` construction implemented by calling `AppleACPIPlatformExpert::mergeDeviceProperties`:

1. During ACPI table initialisation through the recursive ACPI namespace scanning by the calls to `AppleACPIPlatformExpert::createDTNubs`.
2. During IOService registration (`IOServices::registerService`) callbacks implemented as a part of `AppleACPIPlatformExpert::platformAdjustService` function and its private worker method `AppleACPIPlatformExpert::platformAdjustPCIDevice` specific to the PCI devices.

The application of the stages depends on the device presence in ACPI tables. The first stage applies very early but exclusively to the devices present in ACPI tables. The second stage applies to all devices much later after the PCI configuration and may repeat the first stage if the device was not present in ACPI.

For all kernel drivers that may inspect the `IODeviceTree` plane without probing, such as `Lilu` and its plugins such as `WhateverGreen`, it is particularly important to ensure device presence in the ACPI tables. A failure to do so may result **in all kinds of erratic behaviour** caused by ignoring the injected device properties as they were not constructed at the first stage. See `SSDT-IMEI.dsl` and `SSDT-BRGO.dsl` for an example.

6.2 Properties

1. Add

Type: `plist dict`

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

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

2. Delete

Type: `plist dict`

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

6.3 Common Properties

Some known properties include:

- `device-id`
User-specified device identifier used for I/O Kit matching. Has 4 byte data type.
- `vendor-id`
User-specified vendor identifier used for I/O Kit matching. Has 4 byte data type.

- **AAPL,ig-platform-id**
Intel GPU framebuffer identifier used for framebuffer selection on Ivy Bridge and newer. Has 4 byte data type.
- **AAPL,snb-platform-id**
Intel GPU framebuffer identifier used for framebuffer selection on Sandy Bridge. Has 4 byte data type.
- **layout-id**
Audio layout used for AppleHDA layout selection. Has 4 byte data type.

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 select 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 affects older operating systems, where various dependency kexts, such as `IOAudioFamily` or `IONetworkingFamily` may not be present in the kernel cache by default. Kernel driver load order follows the item order in the array, thus the dependencies should be written prior to their consumers. **Force** happens before **Add**.

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

5. Patch

Type: `plist array`

Failsafe: Empty

Description: Perform binary patches in kernel and drivers prior to driver addition and removal.

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

6. Quirks

Type: `plist dict`

Description: Apply individual kernel and driver quirks described in the Quirks Properties section below.

7. Scheme

Type: plist dict

Description: Define kernelspace operation mode via parameters described in the Scheme Properties section below.

7.3 Add Properties

1. Arch

Type: plist string

Failsafe: Any

Description: Kext architecture (Any, i386, x86_64).

2. BundlePath

Type: plist string

Failsafe: Empty

Description: Kext bundle path (e.g. Lilu.kext or MyKext.kext/Contents/PlugIns/MySubKext.kext).

3. 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.

4. Enabled

Type: plist boolean

Failsafe: false

Description: This kernel driver will not be added unless set to true.

5. ExecutablePath

Type: plist string

Failsafe: Empty

Description: Kext executable path relative to bundle (e.g. Contents/MacOS/Lilu).

6. MaxKernel

Type: plist string

Failsafe: Empty

Description: Adds kernel driver on specified macOS version or older.

Kernel version can be obtained with `uname -r` command, and should look like 3 numbers separated by dots, for example 18.7.0 is the kernel version for 10.14.6. Kernel version interpretation is implemented as follows:

$$\begin{aligned} \text{ParseDarwinVersion}(\kappa, \lambda, \mu) &= \kappa \cdot 10000 && \text{Where } \kappa \in (0, 99) \text{ is kernel version major} \\ &+ \lambda \cdot 100 && \text{Where } \lambda \in (0, 99) \text{ is kernel version minor} \\ &+ \mu && \text{Where } \mu \in (0, 99) \text{ is kernel version patch} \end{aligned}$$

Kernel version comparison is implemented as follows:

$$\begin{aligned} \alpha &= \begin{cases} \text{ParseDarwinVersion}(\text{MinKernel}), & \text{If MinKernel is valid} \\ 0 & \text{Otherwise} \end{cases} \\ \beta &= \begin{cases} \text{ParseDarwinVersion}(\text{MaxKernel}), & \text{If MaxKernel is valid} \\ \infty & \text{Otherwise} \end{cases} \\ \gamma &= \begin{cases} \text{ParseDarwinVersion}(\text{FindDarwinVersion}()), & \text{If valid "Darwin Kernel Version" is found} \\ \infty & \text{Otherwise} \end{cases} \\ f(\alpha, \beta, \gamma) &= \alpha \leq \gamma \leq \beta \end{aligned}$$

Here *ParseDarwinVersion* argument is assumed to be 3 integers obtained by splitting Darwin kernel version string from left to right by the . symbol. *FindDarwinVersion* function looks up Darwin kernel version by locating "Darwin Kernel Version $\kappa.\lambda.\mu$ " string in the kernel image.

7. `MinKernel`
Type: `plist string`
Failsafe: `Empty`
Description: Adds kernel driver on specified macOS version or newer.
Note: Refer to the `Add MaxKernel` description for matching logic.
8. `PlistPath`
Type: `plist string`
Failsafe: `Empty`
Description: Kext `Info.plist` path relative to bundle (e.g. `Contents/Info.plist`).

7.4 Block Properties

1. `Arch`
Type: `plist string`
Failsafe: `Any`
Description: Kext block architecture (`Any`, `i386`, `x86_64`).
2. `Comment`
Type: `plist string`
Failsafe: `Empty`
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: This kernel driver will not be blocked unless set to `true`.
4. `Identifier`
Type: `plist string`
Failsafe: `Empty`
Description: Kext bundle identifier (e.g. `com.apple.driver.AppleTyMCEDriver`).
5. `MaxKernel`
Type: `plist string`
Failsafe: `Empty`
Description: Blocks kernel driver on specified macOS version or older.
Note: Refer to the `Add MaxKernel` description for matching logic.
6. `MinKernel`
Type: `plist string`
Failsafe: `Empty`
Description: Blocks kernel driver on specified macOS version or newer.
Note: Refer to the `Add MaxKernel` description for matching logic.

7.5 Emulate Properties

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

This property primarily serves for three needs:

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

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

Note 2: Normally it is only the value of `EAX` that needs to be taken care of, since it represents the full `CPUID`. The remaining bytes are to be left as zeroes. Byte order is Little Endian, so for example, `C3 06 03 00` stands for `CPUID 0x0306C3` (Haswell).

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

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

Note 4: Note that the following configurations are unsupported by XCPM (at least out of the box):

- Consumer Ivy Bridge (0x0306A9) as Apple disabled XCPM for Ivy Bridge and recommends legacy power management for these CPUs. `_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 hacks for older models can be found in the `Special NOTES` section of [acidanthera/bugtracker#365](#).

2. `Cpuid1Mask`

Type: plist data, 16 bytes

Failsafe: All zero

Description: Bit mask of active bits in `Cpuid1Data`.

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

3. `DummyPowerManagement`

Type: plist boolean

Failsafe: false

Requirement: 10.4

Description: Disables `AppleIntelCpuPowerManagement`.

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

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

4. `MaxKernel`

Type: plist string

Failsafe: Empty

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

Description: Kext architecture (Any, i386, x86_64).

2. **BundlePath**
Type: plist string
Failsafe: Empty
Description: Kext bundle path (e.g. `System/Library/Extensions/IONetworkingFamily.kext`).
3. **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.
4. **Enabled**
Type: plist boolean
Failsafe: false
Description: This kernel driver will not be added when not present unless set to `true`.
5. **ExecutablePath**
Type: plist string
Failsafe: Empty
Description: Kext executable path relative to bundle (e.g. `Contents/MacOS/IONetworkingFamily`).
6. **Identifier**
Type: plist string
Failsafe: Empty
Description: Kext identifier to perform presence checking before adding (e.g. `com.apple.iokit.IONetworkingFamily`). Only drivers which identifiers are not be found in the cache will be added.
7. **MaxKernel**
Type: plist string
Failsafe: Empty
Description: Adds kernel driver on specified macOS version or older.
Note: Refer to Add MaxKernel description for matching logic.
8. **MinKernel**
Type: plist string
Failsafe: Empty
Description: Adds kernel driver on specified macOS version or newer.
Note: Refer to Add MaxKernel description for matching logic.
9. **PlistPath**
Type: plist string
Failsafe: Empty
Description: Kext Info.plist path relative to bundle (e.g. `Contents/Info.plist`).

7.7 Patch Properties

1. **Arch**
Type: plist string
Failsafe: Any
Description: Kext patch architecture (`Any`, `i386`, `x86_64`).
2. **Base**
Type: plist string
Failsafe: Empty (Ignored)
Description: Selects symbol-matched base for patch lookup (or immediate replacement) by obtaining the address of the provided symbol name.
3. **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.

4. Count
Type: plist integer
Failsafe: 0
Description: Number of patch occurrences to apply. 0 applies the patch to all occurrences found.
5. Enabled
Type: plist boolean
Failsafe: false
Description: This kernel patch will not be used unless set to **true**.
6. Find
Type: plist data
Failsafe: Empty (Immediate replacement at Base)
Description: Data to find. Must be equal to **Replace** in size if set.
7. Identifier
Type: plist string
Failsafe: Empty
Description: Kext bundle identifier (e.g. `com.apple.driver.AppleHDA`) or `kernel` for kernel patch.
8. Limit
Type: plist integer
Failsafe: 0 (Search entire kext or kernel)
Description: Maximum number of bytes to search for.
9. Mask
Type: plist data
Failsafe: Empty (Ignored)
Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Must be equal to **Replace** in size if set.
10. MaxKernel
Type: plist string
Failsafe: Empty
Description: Patches data on specified macOS version or older.
Note: Refer to the **Add MaxKernel** description for matching logic.
11. MinKernel
Type: plist string
Failsafe: Empty
Description: Patches data on specified macOS version or newer.
Note: Refer to the **Add MaxKernel** description for matching logic.
12. Replace
Type: plist data
Failsafe: Empty
Description: Replacement data of one or more bytes.
13. ReplaceMask
Type: plist data
Failsafe: Empty (Ignored)
Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Must be equal to **Replace** in size if set.
14. Skip
Type: plist integer
Failsafe: 0 (Do not skip any occurrences)
Description: Number of found occurrences to skip before replacements are applied.

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. `./ifreextract 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 select CPUs, which have no native XCPM support.

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

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

4. AppleXcpmForceBoost

Type: plist boolean

Failsafe: false

Requirement: 10.8 (not required for older)

Description: Forces maximum performance in XCPM mode.

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

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

5. CustomSMBIOSGuid
Type: plist boolean
Failsafe: false
Requirement: 10.4
Description: Performs GUID patching for UpdateSMBIOSMode Custom mode. Usually relevant for Dell laptops.
6. DisableIoMapper
Type: plist boolean
Failsafe: false
Requirement: 10.8 (not required for older)
Description: Disables IOMapper support in XNU (VT-d), which may conflict with the firmware implementation.
Note: This option is a preferred alternative to deleting DMAR ACPI table and disabling VT-d in firmware preferences, which does not break VT-d support in other systems in case they need it.
7. DisableLinkeditJettison
Type: plist boolean
Failsafe: false
Requirement: 11
Description: Disables __LINKEDIT jettison code.

This option lets Lilu.kext and possibly some others function in macOS Big Sur with best performance without keepsyms=1 boot argument.
8. DisableRtcChecksum
Type: plist boolean
Failsafe: false
Requirement: 10.4
Description: Disables primary checksum (0x58-0x59) writing in AppleRTC.

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

Note 2: This option will not protect areas from being overwritten at firmware stage (e.g. macOS bootloader), see AppleRtcRam protocol description if this is desired.
9. ExtendBTFeatureFlags
Type: plist boolean
Failsafe: false
Requirement: 10.8
Description: Set FeatureFlags to 0x0F for full functionality of Bluetooth, including Continuity.

Note: This option is a substitution for BT4LEContinuityFixup.kext, which does not function properly due to late patching progress.
10. ExternalDiskIcons
Type: plist boolean
Failsafe: false
Requirement: 10.4
Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

Note: This option should be avoided whenever possible. Modern firmware usually have compatible AHCI controllers.
11. ForceSecureBootScheme
Type: plist boolean
Failsafe: false
Requirement: 11
Description: Force x86 scheme for IMG4 verification.

Note: This option is required on virtual machines when using SecureBootModel different from x86legacy.
12. IncreasePciBarSize
Type: plist boolean

Failsafe: false
Requirement: 10.10
Description: Increases 32-bit PCI bar size in IOPCIFamily from 1 to 4 GBs.

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

13. **LapicKernelPanic**
Type: plist boolean
Failsafe: false
Requirement: 10.6 (64-bit)
Description: Disables kernel panic on LAPIC interrupts.
14. **LegacyCommpage**
Type: plist boolean
Failsafe: false
Requirement: 10.4 - 10.6
Description: Replaces the default 64-bit commpage bcopy implementation with one that does not require SSSE3, useful for legacy platforms. This prevents a `commpage no match for last panic` due to no available 64-bit bcopy functions that do not require SSSE3.
15. **PanicNoKextDump**
Type: plist boolean
Failsafe: false
Requirement: 10.13 (not required for older)
Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.
16. **PowerTimeoutKernelPanic**
Type: plist boolean
Failsafe: false
Requirement: 10.15 (not required for older)
Description: Disables kernel panic on `setPowerState` timeout.

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

17. **SetApfsTrimTimeout**
Type: plist integer
Failsafe: -1
Requirement: 10.14 (not required for older)
Description: Set trim timeout in microseconds for APFS filesystems on SSDs.

APFS filesystem is designed in a way that the space controlled via spaceman structure is either used or free. This may be different in other filesystems where the areas can be marked as used, free, and *unmapped*. All free space is trimmed (unmapped/deallocated) at macOS startup. The trimming procedure for NVMe drives happens in LBA ranges due to the nature of DSM command with up to 256 ranges per command. The more fragmented the memory on the drive is, the more commands are necessary to trim all the free space.

Depending on the SSD controller and the drive fragmentation trim procedure may take considerable amount of time, causing noticeable boot slowdown APFS driver explicitly ignores previously unmapped areas and trims them on boot again and again. To workaround boot slowdown macOS driver introduced a timeout (9.999999 seconds) that stops trim operation when it did not manage to complete in time. On many controllers, such as Samsung, where the deallocation is not very fast, the timeout is reached very quickly. Essentially it means that macOS will try to trim all the same lower blocks that have already been deallocated, but will never have enough time to deallocate higher blocks once the fragmentation increases. This means that trimming on these SSDs will be broken soon after the installation, causing extra wear to the flash.

One way to workaround the problem is to increase the timeout to a very high value, which at the cost of slow boot times (extra minutes) will ensure that all the blocks are trimmed. Set this option to a high value, such as

4294967295, to ensure all the blocks are trimmed.

Another way is to utilise over-provisioning if it is supported or create a dedicated unmapped partition where the reserve blocks can be found by the controller. In this case the trim operation can also be disabled by setting a very low timeout. e.g. 999. See more details in this article.

18. ThirdPartyDrives

Type: plist boolean

Failsafe: false

Requirement: 10.6 (not required for older)

Description: Apply vendor patches to IOAHCIBlockStorage.kext to enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

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

19. XhciPortLimit

Type: plist boolean

Failsafe: false

Requirement: 10.11 (not required for older)

Description: Patch various kexts (AppleUSBXHCI.kext, AppleUSBXHCIPCI.kext, IOUSBHostFamily.kext) to remove USB port count limit of 15 ports.

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

7.9 Scheme Properties

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

1. FuzzyMatch

Type: plist boolean

Failsafe: false

Description: Use `kernelcache` with different checksums when available.

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

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

2. KernelArch

Type: plist string

Failsafe: Auto

Description: Prefer specified kernel architecture (`Auto`, `i386`, `i386-user32`, `x86_64`) when available.

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

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

Below is the algorithm determining the kernel architecture.

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

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

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

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

3. KernelCache

Type: plist string

Failsafe: Auto

Description: Prefer specified kernel cache type (**Auto**, **Cacheless**, **Mkext**, **Prelinked**) when available.

Different variants of macOS support different kernel caching variants designed to improve boot performance. This setting prevents the use of faster kernel caching variants if slower variants are available for debugging and stability reasons. I.e., 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

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

frames broken on these systems.

8 Misc

8.1 Introduction

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

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

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

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

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

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

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

The boot process is as follows:

- Try looking up first valid primary option through `BootNext` UEFI variable.
- On failure looking up first valid primary option through `BootOrder` UEFI variable.

- Mark the option as the default option to boot.
- Boot option through the picker or without it depending on the `ShowPicker` option.
- Show picker on failure otherwise.

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

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

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

8.2 Properties

1. Boot

Type: plist dict

Description: Apply boot configuration described in the Boot Properties section below.

2. BlessOverride

Type: plist array

Description: Add custom scanning paths through bless model.

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

3. Debug

Type: plist dict

Description: Apply debug configuration described in the Debug Properties section below.

4. Entries

Type: plist array

Description: Add boot entries to boot picker.

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

5. Security

Type: plist dict

Description: Apply security configuration described in the Security Properties section below.

6. Tools

Type: plist array

Description: Add tool entries to boot picker.

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

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

8.3 Boot Properties

1. ConsoleAttributes

Type: plist integer

Failsafe: 0

Description: Sets specific attributes for console.

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

- 0x00 — EFI_BLACK

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

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

2. HibernateMode

Type: plist string

Failsafe: None

Description: Hibernation detection mode. The following modes are supported:

- **None** — Avoid hibernation (Recommended).
- **Auto** — Use RTC and NVRAM detection.
- **RTC** — Use RTC detection.
- **NVRAM** — Use NVRAM detection.

3. HideAuxiliary

Type: plist boolean

Failsafe: false

Description: Hides auxiliary entries from picker menu by default.

An entry is considered auxiliary when at least one of the following applies:

- Entry is macOS recovery.
- Entry is macOS Time Machine.
- Entry is explicitly marked as **Auxiliary**.
- Entry is system (e.g. **Reset NVRAM**).

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

4. LauncherOption

Type: plist string

Failsafe: Disabled

Description: Register launcher option in firmware preferences for persistence.

Valid values:

- **Disabled** — do nothing.
- **Full** — create or update 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 provides integration with third-party operating system installation and upgrade at the times they overwrite `\EFI\BOOT\BOOTx64.efi` file. By creating a custom option in this file path becomes no longer used for bootstrapping OpenCore. The path used for bootstrapping is specified in **LauncherPath** option.

Note 1: Some types of firmware may have faulty NVRAM, no boot option support, or other incompatibilities. While unlikely, the use of this option may even cause boot failures. This option should be used without any warranty exclusively on the boards known to be compatible. Check [acidanthera/bugtracker#1222](#) for some known issues with Haswell and other boards.

Note 2: Be aware that while NVRAM reset executed from OpenCore should not erase the boot option created in **Bootstrap**, executing NVRAM reset prior to loading OpenCore will remove it. For significant implementation updates (e.g. in OpenCore 0.6.4) make sure to perform NVRAM reset with **Bootstrap** disabled before reenabling.

5. LauncherPath

Type: plist string

Failsafe: Default

Description: Launch path for **LauncherOption**.

Default stays for launched `OpenCore.efi`, any other path, e.g. `\EFI\Launcher.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 picker.

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

Current OpenCore values include:

- 0x0001 — `OC_ATTR_USE_VOLUME_ICON`, provides custom icons for boot entries:
For Tools OpenCore will try to load a custom icon and fallback to the default icon:
 - `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 try to load a custom icon and fallback to the volume icon or the default icon:

- `<ENTRY_PATH>.icns` — icon near the entry file with appended `.icns` extension.

For all other entries OpenCore will try to load a volume icon by searching as follows, and will fallback to the default icon otherwise:

- `.VolumeIcon.icns` file at **Preboot** volume in per-volume directory (`/System/Volumes/Preboot/{GUID}/` when mounted at default location within macOS) for APFS (if present).
- `.VolumeIcon.icns` file at **Preboot** root (`/System/Volumes/Preboot/` when mounted at default location within macOS) for APFS (otherwise).
- `.VolumeIcon.icns` file at volume root for other filesystems.

Note 1: Apple's boot picker partially supports placing a volume icon file at the operating system's **Data** volume root (`/System/Volumes/Data/` when mounted at default location within macOS). This approach is broken: that file is not accessible either by OpenCanopy or by Apple's own boot picker when FileVault 2 is enabled, which should be most people's default choice. Therefore OpenCanopy does not try to support it. You may place a volume icon file at **Preboot** root for compatibility with both the Apple and OpenCanopy boot pickers, or use the **Preboot** per-volume location as above with OpenCanopy as a preferred alternative to Apple's existing approach.

Note 2: Be aware that using a volume icon on any drive overrides the normal boot 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>.1b1` (`<TOOL_NAME>.12x`) file near tool for Tools.

Prerendered labels can be generated via `disklabel` utility or `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. May 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`, enable pointer control in the 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 boot picker.

For macOS bootloader screen reader preference is set in `preferences.efi` archive in `isV0Enabled.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 OpenCore boot picker and macOS bootloader FileVault 2 login window can also be done with `Command + F5` key combination.

Note: screen reader requires working audio support, see `UEFI Audio Properties` section for more details.

8. PollAppleHotKeys

Type: plist boolean

Failsafe: false

Description: Enable modifier hotkey handling in boot picker.

In addition to `action hotkeys`, which are partially described in `PickerMode` section and are normally handled by Apple BDS, there exist modifier keys, which are handled by operating system bootloader, namely `boot.efi`. These keys allow to change operating system behaviour by providing different boot modes.

On some types of firmware, it may be problematic to use modifier keys due to driver incompatibilities. To workaround this problem this option allows registering select hotkeys in a more permissive manner from within boot picker. Such extensions include the support of tapping on keys in addition to holding and pressing `Shift` along with other keys instead of just `Shift` alone, 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. ShowPicker

Type: plist boolean

Failsafe: false

Description: Show simple boot picker to allow boot entry selection.

10. TakeoffDelay

Type: plist integer, 32 bit

Failsafe: 0

Description: Delay in microseconds performed before handling picker startup and `action hotkeys`.

Introducing a delay may give extra time to hold the right `action hotkey` sequence to e.g. boot to recovery mode. On some platforms setting this option to at least 5000–10000 microseconds may be necessary to access `action hotkeys` at all due to the nature of the keyboard driver.

11. Timeout

Type: plist integer, 32 bit

Failsafe: 0

Description: Timeout in seconds in boot picker before automatic booting of the default boot entry. Use 0 to disable timer.

12. PickerMode

Type: plist string

Failsafe: Builtin

Description: Choose boot picker used for boot management.

Picker describes underlying boot management with an optional user interface responsible for handling boot options. The following values are supported:

- **Builtin** — boot management is handled by OpenCore, a simple text only user interface is used.
- **External** — an external boot management protocol is used if available. Otherwise **Builtin** mode is used.
- **Apple** — Apple boot management is used if available. Otherwise **Builtin** mode is used.

Upon success **External** mode will entirely disable all boot management in OpenCore except policy enforcement. In **Apple** mode it may additionally bypass policy enforcement. See OpenCanopy plugin for an example of a custom user interface.

OpenCore built-in boot picker contains a set of actions chosen during the boot process. The list of supported actions is similar to Apple BDS and in general can be accessed by holding **action hotkeys** during boot process. Currently the following actions are considered:

- **Default** — this is the default option, and it lets OpenCore built-in boot picker to loads the default boot option as specified in Startup Disk preference pane.
- **ShowPicker** — this option forces picker to show. Normally it can be achieved by holding **OPT** key during boot. Setting **ShowPicker** to **true** will make **ShowPicker** the default option.
- **ResetNvram** — this option performs select UEFI variable erase and is normally achieved by holding **CMD+OPT+P+R** key combination during boot. Another way to erase UEFI variables is to choose **Reset NVRAM** in the picker. This option requires **AllowNvramReset** to be set to **true**.
- **BootApple** — this options performs booting to the first found Apple operating system unless the default chosen operating system is already made by Apple. Hold **X** key to choose this option.
- **BootAppleRecovery** — this option performs booting to Apple operating system recovery. Either the one related to the default chosen operating system, or first found in case default chosen operating system is not made by Apple or has no recovery. Hold **CMD+R** key combination to choose this option.

Note 1: Activated **KeySupport**, **OpenUsbKbDxe**, or similar driver is required for key handling to work. On several types of firmware, it is not possible to get all the key functions.

Note 2: In addition to **OPT** OpenCore supports **Escape** key to display picker when **ShowPicker** is disabled. This key exists for the **Apple** picker mode and for firmware with PS/2 keyboards that fail to report held **OPT** keys and requiring continual presses of the **Escape** key to access the boot menu.

Note 3: On Macs with problematic GOP, it may be difficult to access the Apple BootPicker. The **BootKicker** utility can be blessed to workaround this problem even without loading OpenCore. On some Macs however, the **BootKicker** utility cannot be run from OpenCore.

13. PickerVariant

Type: plist string

Failsafe: Auto

Description: Choose specific icon set used for boot management.

The following values are supported:

- **Auto** — Automatically select one set of icons based on **DefaultBackground** colour.
- **Default** — Normal icon set (without prefix).
- **Old** — Vintage icon set (**Old** filename prefix).
- **Modern** — Nouveau icon set (**Modern** filename prefix).
- Other value — Custom icon set if supported by the resources.

8.4 Debug Properties

1. AppleDebug

Type: plist boolean

Failsafe: false

Description: Enable **boot.efi** debug log saving to OpenCore log.

Note: This option only applies to 10.15.4 and newer.

2. ApplePanic

Type: plist boolean

Failsafe: false

Description: Save macOS kernel panic to OpenCore root partition.

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

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

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

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

3. DisableWatchDog

Type: plist boolean

Failsafe: false

Description: Some types of firmware may not succeed in booting the operating system quickly, especially in debug mode, which results in the watchdog timer aborting the process. This option turns off the watchdog timer.

4. DisplayDelay

Type: plist integer

Failsafe: 0

Description: Delay in microseconds performed after every printed line visible onscreen (i.e. console).

5. DisplayLevel

Type: plist integer, 64 bit

Failsafe: 0

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

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

6. SerialInit

Type: plist boolean

Failsafe: false

Description: Perform serial port initialisation.

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

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

See more details in [Debugging](#) section.

7. SysReport

Type: plist boolean

Failsafe: false

Description: Produce system report on ESP folder.

This option will create a **SysReport** directory on ESP partition unless it is already present. The directory will contain ACPI, SMBIOS, and audio codec dumps. Audio codec dumps require an audio backend driver to be loaded.

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

8. Target

Type: plist integer

Failsafe: 0

Description: A bitmask (sum) of enabled logging targets. By default all the logging output is hidden, so this option is required to be set when debugging is necessary.

The following logging targets are supported:

- 0x01 (bit 0) — Enable logging, otherwise all log is discarded.
- 0x02 (bit 1) — Enable basic console (onscreen) logging.
- 0x04 (bit 2) — Enable logging to Data Hub.
- 0x08 (bit 3) — Enable serial port logging.
- 0x10 (bit 4) — Enable UEFI variable logging.
- 0x20 (bit 5) — Enable non-volatile UEFI variable logging.
- 0x40 (bit 6) — Enable logging to file.

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

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

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

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

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

Warning: Some types of firmware appear to have flawed NVRAM garbage collection. This means that they may not be able to always free space after variable deletion. Do not use non-volatile NVRAM logging without extra need on such devices.

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

File logging will create a file named **opencore-YYYY-MM-DD-HHMMSS.txt** at EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmware are not reliable and may corrupt data when writing files through UEFI. Log writing is attempted in the safest manner and thus, is very slow. Ensure that **DisableWatchDog** is set to **true** when a slow drive is used. Try to avoid frequent use of this option when dealing with flash drives as large I/O amounts may speedup memory wear and render the flash drive unusable quicker.

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

Drivers and tools:

- BMF — OpenCanopy, bitmap font
- BS — Bootstrap
- GSTT — GoptStop
- HDA — AudioDxe
- KKT — KeyTester

- MMDD — MmapDump
- OCPAVP — PavpProvision
- OCRST — ResetSystem
- OCUI — OpenCanopy
- OC — OpenCore main, also OcMainLib
- VMOPT — VerifyMemOpt

Libraries:

- AAPL — OcDebugLogLib, Apple EfiBoot logging
- OCABC — OcAfterBootCompatLib
- OCAE — OcAppleEventLib
- OCAK — OcAppleKernelLib
- OCAU — OcAudioLib
- OCA — OcAcpiLib
- OCBP — OcAppleBootPolicyLib
- OCB — OcBootManagementLib
- OCCL — OcAppleChunkListLib
- OCCPU — OcCpuLib
- OCC — OcConsoleLib
- OCDC — OcDriverConnectionLib
- OCDH — OcDataHubLib
- OCDI — OcAppleDiskImageLib
- OCFS — OcFileLib
- OCFV — OcFirmwareVolumeLib
- OCHS — OcHashServicesLib
- OCIA4 — OcAppleImg4Lib
- OCIC — OcImageConversionLib
- OCII — OcInputLib
- OCJS — OcApfsLib
- OCKM — OcAppleKeyMapLib
- OCL — OcDebugLogLib
- OCMCO — OcMachoLib
- OCME — OcHeciLib
- OCMM — OcMemoryLib
- OCPE — OcPeCoffLib, OcPeCoffExtLib
- OCPI — OcFileLib, partition info
- OCPNG — OcPngLib
- OCRAM — OcAppleRamDiskLib
- OCRTC — OcRtcLib
- OCSB — OcAppleSecureBootLib
- OCSMB — OcSmbiosLib
- OCSMC — OcSmcLib
- OCST — OcStorageLib
- OCS — OcSerializedLib
- OCTPL — OcTemplateLib
- OCUC — OcUnicodeCollationLib
- OCUT — OcAppleUserInterfaceThemeLib
- OCXML — OcXmlLib

8.5 Security Properties

1. AllowNvramReset

Type: plist boolean

Failsafe: false

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

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

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

2. `AllowSetDefault`

Type: plist boolean

Failsafe: false

Description: Allow `CTRL+Enter` and `CTRL+Index` handling to set the default boot option in boot picker.

3. `ApECID`

Type: plist integer, 64 bit

Failsafe: 0

Description: Apple Enclave Identifier.

Setting this value to any non-zero 64-bit integer will allow using personalised Apple Secure Boot identifiers. To use this setting, make sure to generate a random 64-bit number with a cryptographically secure random number generator. As an alternative, first 8 bytes of `SystemUUID` can be used for `ApECID`, this is found in macOS 11 for Macs without the T2 chip.

With this value set and `SecureBootModel` valid and not `Disabled` it is possible to achieve **Full Security** of Apple Secure Boot.

To start using personalised Apple Secure Boot, the operating system will have to be reinstalled or personalised. Unless the operating system is personalised, macOS DMG recovery cannot be loaded. If DMG recovery is missing, it can be downloaded with `macrecovery` utility and put to `com.apple.recovery.boot` as explained in Tips and Tricks section. Note that DMG loading needs to be set to `Signed` to use any DMG with Apple Secure Boot.

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

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

Before macOS 11, which introduced a dedicated `x86legacy` model for models without the T2 chip, personalised Apple Secure Boot may not work as expected. When reinstalling the operating system, macOS Installer from macOS 10.15 and older, will usually run out of free memory on the `/var/tmp` partition when trying to install macOS with the personalised Apple Secure Boot. Soon after downloading the macOS installer image an **Unable to verify macOS** error message will appear. To workaround this issue allocate a dedicated RAM disk of 2 MBs for macOS personalisation by entering the following commands in macOS recovery terminal before starting the installation:

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

4. `AuthRestart`

Type: plist boolean

Failsafe: false

Description: Enable `VirtualSMC`-compatible authenticated restart.

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

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

5. `BlacklistAppleUpdate`

Type: plist boolean

Failsafe: false

Description: Ignore boot options trying to update Apple peripheral firmware (e.g. `MultiUpdater.efi`).

Note: This option exists due to some operating systems, namely macOS Big Sur, being incapable of disabling firmware updates with the NVRAM variable (`run-efi-updater`).

6. DmgLoading

Type: plist string

Failsafe: Signed

Description: Define Disk Image (DMG) loading policy used for macOS Recovery.

Valid values:

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

7. EnablePassword

Type: plist boolean

Failsafe: false

Description: Enable password protection to allow sensitive operations.

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

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

8. ExposeSensitiveData

Type: plist integer

Failsafe: 0x6

Description: Sensitive data exposure bitmask (sum) to operating system.

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

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

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

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

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

To obtain OpenCore version use the following command in macOS:

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

To obtain OEM information use the following commands in macOS:

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

9. **HaltLevel**
Type: plist integer, 64 bit
Failsafe: 0x80000000 (DEBUG_ERROR)
Description: EDK II debug level bitmask (sum) causing CPU to halt (stop execution) after obtaining a message of **HaltLevel**. Possible values match **DisplayLevel** values.
10. **PasswordHash**
Type: plist data 64 bytes
Failsafe: all zero
Description: Password hash used when **EnabledPassword** is set.
11. **PasswordSalt**
Type: plist data
Failsafe: empty
Description: Password salt used when **EnabledPassword** is set.
12. **Vault**
Type: plist string
Failsafe: Secure
Description: Enables vaulting mechanism in **OpenCore**.

Valid values:

- **Optional** — require nothing, no vault is enforced, insecure.
- **Basic** — require **vault.plist** file present in **OC** directory. This provides basic filesystem integrity verification and may protect from unintentional filesystem corruption.
- **Secure** — require **vault.sig** signature file for **vault.plist** in **OC** directory. This includes **Basic** integrity checking but also attempts to build a trusted bootchain.

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

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

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

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

The complete set of commands to:

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

Can look as follows:

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

Note 1: While it may appear obvious, an external method is required to verify **OpenCore.efi** and **BOOTx64.efi** for secure boot path. For this, it is recommended to enable UEFI SecureBoot using a custom certificate and to

sign `OpenCore.efi` and `BOOTx64.efi` with a custom key. More details on customising secure boot on modern firmware can be found in Taming UEFI SecureBoot paper (in Russian).

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

13. ScanPolicy

Type: plist integer, 32 bit

Failsafe: 0x10F0103

Description: Define operating system detection policy.

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

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

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

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

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

14. SecureBootModel

Type: plist string

Failsafe: Default

Description: Apple Secure Boot hardware model.

Sets Apple Secure Boot hardware model and policy. Specifying this value defines which operating systems will be

bootable. Operating systems shipped before the specified model was released will not boot. Valid values:

- **Default** — Recent available model, currently set to j137.
- **Disabled** — No model, Secure Boot will be disabled.
- j137 — iMacPro1,1 (December 2017). Minimum macOS 10.13.2 (17C2111)
- j680 — MacBookPro15,1 (July 2018). Minimum macOS 10.13.6 (17G2112)
- j132 — MacBookPro15,2 (July 2018). Minimum macOS 10.13.6 (17G2112)
- j174 — Macmini8,1 (October 2018). Minimum macOS 10.14 (18A2063)
- j140k — MacBookAir8,1 (October 2018). Minimum macOS 10.14.1 (18B2084)
- j780 — MacBookPro15,3 (May 2019). Minimum macOS 10.14.5 (18F132)
- j213 — MacBookPro15,4 (July 2019). Minimum macOS 10.14.5 (18F2058)
- j140a — MacBookAir8,2 (July 2019). Minimum macOS 10.14.5 (18F2058)
- j152f — MacBookPro16,1 (November 2019). Minimum macOS 10.15.1 (19B2093)
- j160 — MacPro7,1 (December 2019). Minimum macOS 10.15.1 (19B88)
- j230k — MacBookAir9,1 (March 2020). Minimum macOS 10.15.3 (19D2064)
- j214k — MacBookPro16,2 (May 2020). Minimum macOS 10.15.4 (19E2269)
- j223 — MacBookPro16,3 (May 2020). Minimum macOS 10.15.4 (19E2265)
- j215 — MacBookPro16,4 (June 2020). Minimum macOS 10.15.5 (19F96)
- j185 — iMac20,1 (August 2020). Minimum macOS 10.15.6 (19G2005)
- j185f — iMac20,2 (August 2020). Minimum macOS 10.15.6 (19G2005)
- x86legacy — Macs without T2 chip and VMs. Minimum macOS 11.0.1 (20B29)

Apple Secure Boot appeared in macOS 10.13 on models with T2 chips. Since **PlatformInfo** and **SecureBootModel** are independent, Apple Secure Boot can be used with any SMBIOS with and without T2. Setting **SecureBootModel** to any valid value but **Disabled** is equivalent to **Medium Security** of Apple Secure Boot. The **ApECID** value must also be specified to achieve **Full Security**. Check **ForceSecureBootScheme** when using Apple Secure Boot on a virtual machine.

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

- (a) As with T2 Macs, unsigned kernel drivers and several signed kernel drivers, including NVIDIA Web Drivers, cannot be installed.
- (b) The list of cached drivers may be different, resulting in the need to change the list of **Added** or **Forced** kernel drivers. For example, **I080211Family** cannot be injected in this case.
- (c) System volume alterations on operating systems with sealing, such as macOS 11, may result in the operating system being unbootable. Do not try to disable system volume encryption unless Apple Secure Boot is disabled.
- (d) If the platform requires certain settings, but they were not enabled, because the obvious issues did not trigger before, boot failure might occur. Be extra careful with **IgnoreInvalidFlexRatio** or **HashServices**.
- (e) Operating systems released before Apple Secure Boot landed (e.g. macOS 10.12 or earlier) will still boot until UEFI Secure Boot is enabled. This is so, because from Apple Secure Boot point they are treated as incompatible and are assumed to be handled by the firmware as Microsoft Windows is.
- (f) On older CPUs (e.g. before Sandy Bridge) enabling Apple Secure Boot might cause slightly slower loading by up to 1 second.
- (g) Since **Default** value will increase with time to support the latest major release operating system, it is not recommended to use **ApECID** and **Default** value together.
- (h) Installing macOS with Apple Secure Boot enabled is not possible while using HFS+ target volume. This may include HFS+ formatted drives when no spare APFS drive is available.

Sometimes the already installed operating system may have outdated Apple Secure Boot manifests on the **Preboot** partition causing boot failure. If there is “OCB: Apple Secure Boot prohibits this boot entry, enforcing!” message, it is likely the case. When this happens, either reinstall the operating system or copy the manifests (files with **.im4m** extension, such as **boot.efi.j137.im4m**) from **/usr/standalone/i386** to **/Volumes/Preboot/<UUID>/System/Library/CoreServices**. Here **<UUID>** is the system volume identifier. On HFS+ installations the manifests should be copied to **/System/Library/CoreServices** on the system volume.

For more details on how to configure Apple Secure Boot with UEFI Secure Boot refer to UEFI Secure Boot section.

8.6 Entry Properties

1. Arguments
Type: plist string
Failsafe: Empty
Description: Arbitrary ASCII string used as boot arguments (load options) of the specified entry.
2. Auxiliary
Type: plist boolean
Failsafe: false
Description: This entry will not be listed by default when `HideAuxiliary` is set to `true`.
3. Comment
Type: plist string
Failsafe: Empty
Description: Arbitrary ASCII string used to provide human readable reference for the entry. Whether this value is used is implementation defined.
4. Enabled
Type: plist boolean
Failsafe: false
Description: This entry will not be listed unless set to `true`.
5. Name
Type: plist string
Failsafe: Empty
Description: Human readable entry name displayed in boot picker.
6. Path
Type: plist string
Failsafe: Empty
Description: Entry location depending on entry type.
 - **Entries** specify external boot options, and therefore take device paths in `Path` key. These values are not checked, thus be extremely careful. Example: `PciRoot(0x0)/Pci(0x1,0x1)/.../EFI/COOL.EFI`
 - **Tools** specify internal boot options, which are part of bootloader vault, and therefore take file paths relative to `OC/Tools` directory. Example: `OpenShell.efi`.
7. RealPath
Type: plist boolean
Failsafe: false
Description: Pass full path to the tool when launching.

Passing tool directory may be unsafe for tool accidentally trying to access files without checking their integrity and thus should generally be disabled. Reason to enable this property may include cases where tools cannot work without external files or may need them for better function (e.g. `memtest86` for logging and configuration or `Shell` for automatic script execution).

Note: This property is only valid for **Tools**. For **Entries** this property cannot be specified and is always `true`.
8. TextMode
Type: plist boolean
Failsafe: false
Description: Run the entry in text mode instead of graphics mode.

This setting may be beneficial to some older tools that require text output. By default all the tools are launched in graphics mode. Read more about text modes in the Output Properties section below.

9 NVRAM

9.1 Introduction

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

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

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

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

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

1. Not all tools may be aware of protected namespaces.
When `RequestBootVarRouting` is used `Boot-`prefixed variable access is restricted and protected in a separate namespace. To access the original variables tools have to be aware of `OC_FIRMWARE_RUNTIME` logic.

9.2 Properties

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

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

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

2. Delete
Type: `plist dict`
Description: Removes NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.
3. LegacyEnable
Type: `plist boolean`
Failsafe: `false`
Description: Enables loading of NVRAM variable file named `nvram.plist` from EFI volume root.

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

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

Variable loading happens prior to `Delete` (and `Add`) phases. Unless `LegacyOverwrite` is enabled, it will not overwrite any existing variable. Variables allowed to be set must be specified in `LegacySchema`. Third-party scripts may be used to create `nvram.plist` file. An example of such script can be found in `Utilities`. The use of third-party scripts may require `ExposeSensitiveData` set to `0x3` to provide `boot-path` variable with OpenCore EFI partition UUID.

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

4. LegacyOverwrite

Type: plist boolean

Failsafe: false

Description: Permits overwriting firmware variables from `nvr.plist`.

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

5. LegacySchema

Type: plist dict

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

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

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

6. WriteFlash

Type: plist boolean

Failsafe: false

Description: Enables writing to flash memory for all added variables.

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

To read NVRAM variable value from macOS, `nvr` could be used by concatenating GUID and name variables separated by a `:` symbol. For example, `nvr 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 PlatformNVRAM or Generic subsections of 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:

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:csr-active-config`
32-bit System Integrity Protection bitmask. Declared in XNU source code in `csr.h`.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeatures`
Combined `FirmwareFeatures` and `ExtendedFirmwareFeatures`. Present on newer Macs to avoid extra parsing of SMBIOS tables.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask`
Combined `FirmwareFeaturesMask` and `ExtendedFirmwareFeaturesMask`. Present on newer Macs to avoid extra parsing of SMBIOS tables.

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID`
Hardware BoardProduct (e.g. `Mac-35C1E88140C3E6CF`). Not present on real Macs, but used to avoid extra parsing of SMBIOS tables, especially in `boot.efi`.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB`
Hardware BoardSerialNumber. Override for MLB. Present on newer Macs (2013+ at least).
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM`
Hardware ROM. Override for ROM. Present on newer Macs (2013+ at least).
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:SSN`
Serial number. Present on newer Macs (2013+ at least).
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:prev-lang:kbd`
ASCII string defining default keyboard layout. Format is `lang-COUNTRY:keyboard`, e.g. `ru-RU:252` for Russian locale and ABC keyboard. Also accepts short forms: `ru:252` or `ru:0` (U.S. keyboard, compatible with 10.9). Full decoded keyboard list from `AppleKeyboardLayouts-L.dat` can be found here. Using non-latin keyboard on 10.14 will not enable ABC keyboard, unlike previous and subsequent macOS versions, and is thus not recommended in case 10.14 is needed.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:security-mode`
ASCII string defining FireWire security mode. Legacy, can be found in `IOFireWireFamily` source code in `IOFireWireController.cpp`. It is recommended not to set this variable, which may speedup system startup. Setting to `full` is equivalent to not setting the variable and `none` disables FireWire security.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:UIScale`
One-byte data defining `boot.efi` user interface scaling. Should be `01` for normal screens and `02` for HiDPI screens.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:DefaultBackgroundColor`
Four-byte BGRA data defining `boot.efi` user interface background colour. Standard colours include `BF BF BF 00` (Light Gray) and `00 00 00 00` (Syrah Black). Other colours may be set at user's preference.

9.5 Other Variables

The following variables may be useful for certain configurations or troubleshooting:

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args`
Kernel arguments, used to pass configuration to Apple kernel and drivers. There are many arguments, which may be found by looking for the use of `PE_parse_boot_argn` function in the kernel or driver code. Some of the known boot arguments include:
 - `acpi_layer=0xFFFFFFFF`
 - `acpi_level=0xFFFF5F` (implies `ACPI_ALL_COMPONENTS`)
 - `arch=i386` (force kernel architecture to `i386`, see `KernelArch`)
 - `batman=VALUE` (`AppleSmartBatteryManager` debug mask)
 - `batman-nosmc=1` (disable `AppleSmartBatteryManager` SMC interface)
 - `cpus=VALUE` (maximum number of CPUs used)
 - `debug=VALUE` (debug mask)
 - `io=VALUE` (IOKit debug mask)
 - `ioaccel_debug=VALUE` (`IOAccelerator` debug mask)
 - `keepsyms=1` (show panic log debug symbols)
 - `kextlog=VALUE` (kernel extension loading debug mask)
 - `nvram-log=1` (enables `AppleEFINVRAM` logs)
 - `nv_disable=1` (disables NVIDIA GPU acceleration)
 - `nvda_drv=1` (legacy way to enable NVIDIA web driver, removed in 10.12)
 - `npci=0x2000` (legacy, disables `kIOPCIConfiguratorPFM64`)
 - `lapic_dont_panic=1` (disable lapic spurious interrupt panic on AP cores)
 - `panic_on_display_hang=1` (trigger panic on display hang)
 - `panic_on_gpu_hang=1` (trigger panic on GPU hang)
 - `slide=VALUE` (manually set KASLR slide)
 - `smcdebug=VALUE` (`AppleSMC` debug mask)
 - `spin_wait_for_gpu=1` (reduces GPU timeout on high load)
 - `-amd_no_dgpu_accel` (alternative to WhateverGreen's `-radvesa` for new GPUs)
 - `-nehalem_error_disable`
 - `-no_compat_check` (disable model checking on 10.7+)

- **-s** (single mode)
- **-v** (verbose mode)
- **-x** (safe mode)

There are multiple external places summarising macOS argument lists: example 1, example 2.

- **7C436110-AB2A-4BBB-A880-FE41995C9F82:booterconfig**

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

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

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

- `boot-save-log=VALUE` — debug log save mode for normal boot.
 - * `0`
 - * `1`
 - * `2` — (default).
 - * `3`
 - * `4` — (save to file).
- `wake-save-log=VALUE` — debug log save mode for hibernation wake.
 - * `0` — disabled.
 - * `1`
 - * `2` — (default).
 - * `3` — (unavailable).
 - * `4` — (save to file, unavailable).
- `breakpoint=VALUE` — enables debug breaks (missing in production `boot.efi`).
 - * `0` — disables debug breaks on errors (default).
 - * `1` — enables debug breaks on errors.
- `console=VALUE` — enables console logging.
 - * `0` — disables console logging.
 - * `1` — enables console logging when debug protocol is missing (default).
 - * `2` — enables console logging unconditionally (unavailable).
- `embed-log-dt=VALUE` — enables DeviceTree logging.
 - * `0` — disables DeviceTree logging (default).
 - * `1` — enables DeviceTree logging.
- `kc-read-size=VALUE` — Chunk size used for buffered I/O from network or disk for prelinkedkernel reading and related. Set to 1MB (0x100000) by default, can be tuned for faster booting.
- `log-level=VALUE` — log level bitmask.
 - * `0x01` — enables trace logging (default).
- `serial=VALUE` — enables serial logging.
 - * `0` — disables serial logging (default).
 - * `1` — enables serial logging for `EXITBS:END` onwards.
 - * `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 verbose output from `boot.efi` on modern macOS versions enable `AppleDebug` option. This will save the log to general OpenCore log. For versions before 10.15.4 set `bootercfg` to `log=1`. This will print verbose output onscreen.

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg-once`
Booter arguments override removed after first launch. Otherwise equivalent to `bootercfg`.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:efiboot-perf-record`
Enable performance log saving in `boot.efi`. Performance log is saved to physical memory and is pointed by `efiboot-perf-record-data` and `efiboot-perf-record-size` variables. Starting from 10.15.4 it can also be saved to OpenCore log by `AppleDebug` option.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:fmm-computer-name`
Current saved host name. ASCII string.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:nvda_drv`
NVIDIA Web Driver control variable. Takes ASCII digit 1 or 0 to enable or disable installed driver.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:run-efi-updater`
Override EFI firmware updating support in macOS (`MultiUpdater`, `ThorUtil`, and so on). Setting this to `No` or alternative boolean-castable value will prevent any firmware updates in macOS starting with 10.10 at least.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:StartupMute`
Mute startup chime sound in firmware audio support. 8-bit integer. The value of `0x00` means unmuted. Missing variable or any other value means muted.
- `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 of several identification fields generated or filled manually to be compatible with macOS services. The base part of the configuration may be obtained from `AppleModels`, which itself generates a set of interfaces based on a database in YAML format. These fields are written to three select destinations:

- SMBIOS
- Data Hub
- NVRAM

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

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

10.1 Properties

1. Automatic

Type: plist boolean

Failsafe: false

Description: Generate PlatformInfo based on `Generic` section instead of using values from `DataHub`, `NVRAM`, and `SMBIOS` sections.

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

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

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

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 `Generic` or `DataHub` sections depending on `Automatic` value.

Note: The implementation of the Data Hub protocol in EFI firmware on essentially all systems, including Apple hardware, means that existing Data Hub entries cannot be overridden, while new entries are added to the end with macOS ignoring them. You can work around this by reinstalling 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 `Generic` or `PlatformNVRAM` sections depending on `Automatic` value. All the other fields are to be specified with `NVRAM` section.

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

5. UpdateSMBIOS

Type: plist boolean

Failsafe: false

Description: Update SMBIOS fields. These fields are read from **Generic** or **SMBIOS** sections depending on **Automatic** value.

6. UpdateSMBIOSMode

Type: plist string

Failsafe: Create

Description: Update SMBIOS fields approach:

- **TryOverwrite** — **Overwrite** if new size is \leq 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 **Custom** approach is making SMBIOS updates exclusive to macOS, avoiding a collision with existing Windows activation and custom OEM software but potentially breaking Apple-specific tools.

7. UseRawUuidEncoding

Type: plist boolean

Failsafe: false

Description: Use raw encoding for SMBIOS UUIDs.

Each UUID AABBCDD-EEFF-GGHH-IIJJ-KKLLMMNNOO PP 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}).

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 **Big Endian** format everywhere but it ignores SMBIOS UUID within macOS.
- `dmidecode` uses **Big Endian** format for SMBIOS 2.5.x or lower and **Little Endian** for 2.6 and newer. Acidanthera `dmidecode` prints all the three.
- Windows uses **Little Endian** format everywhere, but it 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 **Big Endian** format is used to store the `SystemUUID` data. Otherwise **Little Endian** is used.

Note: Since UUIDs used in DataHub and NVRAM 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 dangerous to use Apple in SMBIOS vendor fields for reasons given in `SystemManufacturer` description. However, certain firmware may not provide valid values otherwise, which could break some software.

2. AdviseWindows

Type: plist boolean

Failsafe: false

Description: Forces Windows support in `FirmwareFeatures`.

Added bits to `FirmwareFeatures`:

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

3. 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.
- `Upgradable` — explicitly unset `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` (0x2) in `PlatformFeature`.
- `Soldered` — explicitly set `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` (0x2) in `PlatformFeature`.

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

5. ProcessorType

Type: plist integer

Failsafe: 0 (Automatic)
Description: Refer to SMBIOS `ProcessorType`.

6. `SystemProductName`
Type: `plist string`
Failsafe: Empty (OEM specified or not installed)
Description: Refer to SMBIOS `SystemProductName`.

7. `SystemSerialNumber`
Type: `plist string`
Failsafe: Empty (OEM specified or not installed)
Description: Refer to SMBIOS `SystemSerialNumber`.

Specify special string value `OEM` to extract current value from NVRAM (`SSN` variable) or SMBIOS and use it throughout the sections. This feature can only be used on Mac-compatible firmware.

8. `SystemUUID`
Type: `plist string`, GUID
Failsafe: Empty (OEM specified or not installed)
Description: Refer to SMBIOS `SystemUUID`.

Specify special string value `OEM` to extract current value from NVRAM (`system-id` variable) or SMBIOS and use it throughout the sections. Since not every firmware implementation has valid (and unique) values, this feature is not applicable to some setups, and may provide unexpected results. It is highly recommended to specify the UUID explicitly. Refer to `UseRawUuidEncoding` to determine how SMBIOS value is parsed.

9. `MLB`
Type: `plist string`
Failsafe: Empty (OEM specified or not installed)
Description: Refer to SMBIOS `BoardSerialNumber`.

Specify special string value `OEM` to extract current value from NVRAM (`MLB` variable) or SMBIOS and use it throughout the sections. This feature can only be used on Mac-compatible firmware.

10. `ROM`
Type: `plist metadata`, 6 bytes
Failsafe: Empty (OEM specified or not installed)
Description: Refer to `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM`.

Specify special string value `OEM` to extract current value from NVRAM (`ROM` variable) and use it throughout the sections. This feature can only be used on Mac-compatible firmware.

10.3 DataHub Properties

1. `PlatformName`
Type: `plist string`
Failsafe: Empty (Not installed)
Description: Sets `name` in `gEfiMiscSubClassGuid`. The value found on Macs is `platform` in ASCII.
2. `SystemProductName`
Type: `plist string`
Failsafe: Empty (Not installed)
Description: Sets `Model` in `gEfiMiscSubClassGuid`. The value found on Macs is equal to SMBIOS `SystemProductName` in Unicode.
3. `SystemSerialNumber`
Type: `plist string`
Failsafe: Empty (Not installed)
Description: Sets `SystemSerialNumber` in `gEfiMiscSubClassGuid`. The value found on Macs is equal to SMBIOS `SystemSerialNumber` in Unicode.
4. `SystemUUID`
Type: `plist string`, GUID
Failsafe: Empty (Not installed)

Description: Sets system-id in gEfiMiscSubClassGuid. The value found on Macs is equal to SMBIOS SystemUUID (with swapped byte order).

5. BoardProduct

Type: plist string

Failsafe: Empty (Not installed)

Description: Sets board-id in gEfiMiscSubClassGuid. The value found on Macs is equal to SMBIOS BoardProduct in ASCII.

6. BoardRevision

Type: plist data, 1 byte

Failsafe: 0

Description: Sets board-rev in gEfiMiscSubClassGuid. The value found on Macs seems to correspond to internal board revision (e.g. 01).

7. StartupPowerEvents

Type: plist integer, 64-bit

Failsafe: 0

Description: Sets StartupPowerEvents in gEfiMiscSubClassGuid. The value found on Macs is power management state bitmask, normally 0. Known bits read by X86PlatformPlugin.kext:

- 0x00000001 — Shutdown cause was a PWROK event (Same as GEN_PMCN_2 bit 0)
- 0x00000002 — Shutdown cause was a SYS_PWROK event (Same as GEN_PMCN_2 bit 1)
- 0x00000004 — Shutdown cause was a THRMTRIP# event (Same as GEN_PMCN_2 bit 3)
- 0x00000008 — Rebooted due to a SYS_RESET# event (Same as GEN_PMCN_2 bit 4)
- 0x00000010 — Power Failure (Same as GEN_PMCN_3 bit 1 PWR_FLR)
- 0x00000020 — Loss of RTC Well Power (Same as GEN_PMCN_3 bit 2 RTC_PWR_STS)
- 0x00000040 — General Reset Status (Same as GEN_PMCN_3 bit 9 GEN_RST_STS)
- 0xffffffff80 — SUS Well Power Loss (Same as GEN_PMCN_3 bit 14)
- 0x00010000 — Wake cause was a ME Wake event (Same as PRSTS bit 0, ME_WAKE_STS)
- 0x00020000 — Cold Reboot was ME Induced event (Same as PRSTS bit 1 ME_HRST_COLD_STS)
- 0x00040000 — Warm Reboot was ME Induced event (Same as PRSTS bit 2 ME_HRST_WARM_STS)
- 0x00080000 — Shutdown was ME Induced event (Same as PRSTS bit 3 ME_HOST_PWRDN)
- 0x00100000 — Global reset ME Watchdog Timer event (Same as PRSTS bit 6)
- 0x00200000 — Global reset PowerManagement Watchdog Timer event (Same as PRSTS bit 15)

8. InitialTSC

Type: plist integer, 64-bit

Failsafe: 0

Description: Sets InitialTSC in gEfiProcessorSubClassGuid. Sets initial TSC value, normally 0.

9. FSBFrequency

Type: plist integer, 64-bit

Failsafe: 0 (Automatic)

Description: Sets FSBFrequency in gEfiProcessorSubClassGuid.

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

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

10. ARTFrequency

Type: plist integer, 64-bit

Failsafe: 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 client Intel segment, 25 MHz for server Intel segment, and 19.2 MHz for Intel Atom CPUs. macOS till 10.15 inclusive assumes 24 MHz by default.

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

11. **DevicePathsSupported**
Type: plist integer, 32-bit
Failsafe: 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 usually 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.

- 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.

- Bit 0 — Reserved, set to 0
- Bit 1 — Other
- Bit 2 — Unknown
- Bit 7 — Synchronous
- Bit 13 — Registered (buffered)
- Bit 14 — Unbuffered (unregistered)

10.4.1 Memory Device Properties

1. `AssetTag`

Type: plist string

Failsafe: Unknown

SMBIOS: Memory Device (Type 17) — Asset Tag

Description: Specifies the asset tag of this memory device.

2. BankLocator

Type: plist string

Failsafe: Unknown

SMBIOS: Memory Device (Type 17) — Bank Locator

Description: Specifies the physically labeled bank where the memory device is located.

3. DeviceLocator

Type: plist string

Failsafe: Unknown

SMBIOS: Memory Device (Type 17) — Device Locator

Description: Specifies the physically-labeled socket or board position where the memory device is located.

4. Manufacturer

Type: plist string

Failsafe: Unknown

SMBIOS: Memory Device (Type 17) — Manufacturer

Description: Specifies the manufacturer of this memory device.

5. PartNumber

Type: plist string

Failsafe: Unknown

SMBIOS: Memory Device (Type 17) — Part Number

Description: Specifies the part number of this memory device.

6. SerialNumber

Type: plist string

Failsafe: Unknown

SMBIOS: Memory Device (Type 17) — Serial Number

Description: Specifies the serial number of this memory device.

7. Size

Type: plist integer, 32-bit

Failsafe: 0

SMBIOS: Memory Device (Type 17) — Size

Description: Specifies the size of the memory device, in megabytes. 0 indicates this slot is not populated.

8. Speed

Type: plist integer, 16-bit

Failsafe: 0

SMBIOS: Memory Device (Type 17) — Speed

Description: Specifies the maximum capable speed of the device, in megatransfers per second (MT/s). 0 indicates an unknown speed.

10.5 PlatformNVRAM Properties

1. BID

Type: plist string

Failsafe: Empty (Not installed)

Description: Specifies the value of NVRAM variable 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID.

2. ROM

Type: plist data, 6 bytes

Failsafe: Empty (Not installed)

Description: Specifies the values of NVRAM variables 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM and 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM.

3. MLB

Type: plist string

Failsafe: Empty (Not installed)

Description: Specifies the values of NVRAM variables 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB and 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:MLB.

4. FirmwareFeatures

Type: plist data, 8 bytes

Failsafe: Empty (Not installed)

Description: This variable comes in pair with FirmwareFeaturesMask. Specifies the values of NVRAM variables:

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

5. FirmwareFeaturesMask

Type: plist data, 8 bytes

Failsafe: Empty (Not installed)

Description: This variable comes in pair with FirmwareFeatures. Specifies the values of NVRAM variables:

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

6. SystemSerialNumber

Type: plist string

Failsafe: Empty (Not installed)

Description: Specifies the values of NVRAM variables 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_SSN and 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:SSN.

7. SystemUUID

Type: plist string

Failsafe: Empty (Not installed)

Description: Specifies the value of NVRAM variable 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:system-id for boot services only. The value found on Macs is equal to SMBIOS SystemUUID.

10.6 SMBIOS Properties

1. BIOSVendor

Type: plist string

Failsafe: Empty (OEM specified)

SMBIOS: BIOS Information (Type 0) — Vendor

Description: BIOS Vendor. All rules of SystemManufacturer do apply.

2. BIOSVersion

Type: plist string

Failsafe: Empty (OEM specified)

SMBIOS: BIOS Information (Type 0) — BIOS Version

Description: Firmware version. This value gets updated and takes part in update delivery configuration and macOS version compatibility. This value could look like MM71.88Z.0234.B00.1809171422 in older firmware and is described in BiosId.h. In newer firmware, it should look like 236.0.0.0.0 or 220.230.16.0.0 (iBridge: 16.16.2542.0.0,0). iBridge version is read from BridgeOSVersion variable, and is only present on macs with T2.

Apple ROM Version

```
BIOS ID:      MBP151.88Z.F000.B00.1811142212
Model:        MBP151
EFI Version:   220.230.16.0.0
Built by:      root@quinoa
Date:          Wed Nov 14 22:12:53 2018
Revision:      220.230.16 (B&I)
ROM Version:   F000_B00
Build Type:    Official Build, RELEASE
Compiler:      Apple LLVM version 10.0.0 (clang-1000.2.42)
UUID:          E5D1475B-29FF-32BA-8552-682622BA42E1
UUID:          151B0907-10F9-3271-87CD-4BF5DBECACF5
```

3. **BIOSReleaseDate**
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: BIOS Information (Type 0) — BIOS Release Date
Description: Firmware release date. Similar to **BIOSVersion**. May look like 12/08/2017.
4. **SystemManufacturer**
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — Manufacturer
Description: OEM manufacturer of the particular board. Use failsafe unless strictly required. Do not override to contain **Apple Inc.** on non-Apple hardware, as this confuses numerous services present in the operating system, such as firmware updates, efichk, as well as kernel extensions developed in Acidanthera, such as Lilu and its plugins. In addition it will also make some operating systems such as Linux unbootable.
5. **SystemProductName**
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1), Product Name
Description: Preferred Mac model used to mark the device as supported by the operating system. This value must be specified by any configuration for later automatic generation of the related values in this and other SMBIOS tables and related configuration parameters. If **SystemProductName** is not compatible with the target operating system, **-no_compat_check** boot argument may be used as an override.

Note: If **SystemProductName** is unknown, and related fields are unspecified, default values should be assumed as being set to MacPro6,1 data. The list of known products can be found in **AppleModels**.
6. **SystemVersion**
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — Version
Description: Product iteration version number. May look like 1.1.
7. **SystemSerialNumber**
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — Serial Number
Description: Product serial number in defined format. Known formats are described in **macserial**.
8. **SystemUUID**
Type: plist string, GUID
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — UUID
Description: A UUID is an identifier that is designed to be unique across both time and space. It requires no central registration process.
9. **SystemSKUNumber**
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — SKU Number
Description: Mac Board ID (board-id). May look like Mac-7BA5B2D9E42DDD94 or Mac-F221BEC8 in older models. Sometimes it can be just empty.
10. **SystemFamily**
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — Family
Description: Family name. May look like iMac Pro.
11. **BoardManufacturer**
Type: plist string
Failsafe: Empty (OEM specified)

SMBIOS: Baseboard (or Module) Information (Type 2) - Manufacturer
Description: Board manufacturer. All rules of `SystemManufacturer` do apply.

12. `BoardProduct`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) - Product
Description: Mac Board ID (board-id). May look like Mac-7BA5B2D9E42DDD94 or Mac-F221BEC8 in older models.
13. `BoardVersion`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) - Version
Description: Board version number. Varies, may match `SystemProductName` or `SystemProductVersion`.
14. `BoardSerialNumber`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) — Serial Number
Description: Board serial number in defined format. Known formats are described in `macserial`.
15. `BoardAssetTag`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) — Asset Tag
Description: Asset tag number. Varies, may be empty or `Type2 - Board Asset Tag`.
16. `BoardType`
Type: `plist integer`
Failsafe: 0 (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) — Board Type
Description: Either 0xA (Motherboard (includes processor, memory, and I/O) or 0xB (Processor/Memory Module), refer to Table 15 – Baseboard: Board Type for more details.
17. `BoardLocationInChassis`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) — Location in Chassis
Description: Varies, may be empty or `Part Component`.
18. `ChassisManufacturer`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: System Enclosure or Chassis (Type 3) — Manufacturer
Description: Board manufacturer. All rules of `SystemManufacturer` do apply.
19. `ChassisType`
Type: `plist integer`
Failsafe: 0 (OEM specified)
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`

SMBIOS: `APPLE_SMBIOS_TABLE_TYPE133 - PlatformFeature`

Description: Platform features bitmask. Refer to `AppleFeatures.h` for more details. Use `0xFFFFFFFF` value to not provide this table.

24. `SmcVersion`

Type: plist data, 16 bytes

Failsafe: All zero

SMBIOS: `APPLE_SMBIOS_TABLE_TYPE134 - Version`

Description: ASCII string containing SMC version in upper case. Missing on T2 based Macs. Ignored when zero.

25. `FirmwareFeatures`

Type: plist data, 8 bytes

Failsafe: 0

SMBIOS: `APPLE_SMBIOS_TABLE_TYPE128 - FirmwareFeatures` and `ExtendedFirmwareFeatures`

Description: 64-bit firmware features bitmask. Refer to `AppleFeatures.h` for more details. Lower 32 bits match `FirmwareFeatures`. Upper 64 bits match `ExtendedFirmwareFeatures`.

26. `FirmwareFeaturesMask`

Type: plist data, 8 bytes

Failsafe: 0

SMBIOS: `APPLE_SMBIOS_TABLE_TYPE128 - FirmwareFeaturesMask` and `ExtendedFirmwareFeaturesMask`

Description: Supported bits of extended firmware features bitmask. Refer to `AppleFeatures.h` for more details. Lower 32 bits match `FirmwareFeaturesMask`. Upper 64 bits match `ExtendedFirmwareFeaturesMask`.

27. `ProcessorType`

Type: plist integer, 16-bit

Failsafe: 0 (Automatic)

SMBIOS: `APPLE_SMBIOS_TABLE_TYPE131 - ProcessorType`

Description: Combined of Processor Major and Minor types.

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

11 UEFI

11.1 Introduction

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

11.2 Drivers

Depending on the firmware a different set of drivers may be required. Loading an incompatible driver may lead 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 ExFatDxeLegacy driver should be used due to the lack of RDRAND instruction support.
HfsPlus	Proprietary HFS file system driver with bless support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs HfsPlusLegacy driver should be used due to the lack of RDRAND instruction support.
HiiDatabase*	HII services support driver from MdeModulePkg. This driver is included in most 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 firmware have a flawed 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.
OpenUsbKbdDxe*	USB keyboard driver adding the support of AppleKeyMapAggregator protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin KeySupport, which may work better or worse depending on the firmware.
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 OpenUsbKbdDxe 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
```

11.3 Tools and Applications

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

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

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

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

Listing 3: Blessing tool

Note 1: **/System/Library/CoreServices/BridgeVersion.bin** should be copied to **/Volumes/VOLNAME/DIR**.

Note 2: To be able to use **bless** disabling System Integrity Protection is necessary.

Note 3: To be able to boot Secure Boot might be disabled if present.

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

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

11.4 OpenCanopy

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

OpenCanopy requires graphical resources located in **Resources** directory to run. Sample resources (fonts and images) can be found in **OcBinaryData** repository. Customised icons can be found over the internet (e.g. [here](#) or [there](#)).

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

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

Note: In the following all dimensions are normative for the 1x scaling level and shall be scaled accordingly for other levels.

- **Cursor** — Mouse cursor (mandatory, up to 144x144).
- **Selected** — Selected item (mandatory, 144x144).
- **Selector** — Selecting item (mandatory, up to 144x40).
- **Left** — Scrolling left (mandatory, 40x40).
- **Right** — Scrolling right (mandatory, 40x40).
- **HardDrive** — Generic OS (mandatory, 128x128).

- **Background** — Centred background image.
- **Apple** — Apple OS (128x128).
- **AppleRecv** — Apple Recovery OS (128x128).
- **AppleTM** — Apple Time Machine (128x128).
- **Windows** — Windows (128x128).
- **Other** — Custom entry (see **Entries**, 128x128).
- **ResetNVRAM** — Reset NVRAM system action or tool (128x128).
- **Shell** — Entry with UEFI Shell name for e.g. **OpenShell** (128x128).
- **Tool** — Any other tool (128x128).

Predefined labels are put to `\EFI\OC\Resources\Label` directory. Each label has `.1b1` or `.12x` suffix to represent the scaling level. Full list of labels is provided below. All labels are mandatory.

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

Note: All labels must have a height of exactly 12 px. There is no limit for their width.

Label and icon generation can be performed with bundled utilities: `disklabel` and `icnspack`. Font is Helvetica 12 pt times scale factor.

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

11.5 OpenRuntime

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

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

11.6 Properties

1. APFS

Type: plist dict

Failsafe: None

Description: Provide APFS support as configured in the APFS Properties section below.

2. Audio

Type: plist dict

Failsafe: None

Description: Configure audio backend support described in the Audio Properties section below.

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

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.efi` 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 select 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 firmware and hardware functioning, which should not be used by the operating system. An example of such memory region could be second 256 MB corrupted by 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.

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

2. GlobalConnect

Type: plist boolean

Failsafe: false

Description: Perform full device connection during APFS loading.

Instead of partition handle connection normally used for APFS driver loading every handle is connected recursively. This may take more time than usual but can be the only way to access APFS partitions on some types of 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 at OpenCore startup but also during boot picker. This permits APFS USB hot plug. Disable if not required.

5. MinDate

Type: plist integer

Failsafe: 0

Description: Minimal allowed APFS driver date.

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

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

6. MinVersion

Type: plist integer

Failsafe: 0

Description: Minimal allowed APFS driver version.

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

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

11.8 Audio Properties

1. AudioCodec

Type: plist integer

Failsafe: 0

Description: Codec address on the specified audio controller for audio support.

Normally this contains first audio codec address on the builtin analog audio controller (HDEF). Audio codec addresses, e.g. 2, can be found in the debug log (marked in bold-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 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 empty device path will result in the first available audio controller to be used.

3. AudioOut

Type: plist integer

Failsafe: 0

Description: Index of the output port of the specified codec starting from 0.

Normally this contains the index of the green out of the builtin analog audio controller (HDEF). The number of output nodes (N) in the debug log (marked in bold-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 reader will use this volume level, when the calculated volume level is less than `MinimumVolume`. Boot chime sound will not play if the calculated volume level is less than `MinimumVolume`.

6. `PlayChime`

Type: plist string

Failsafe: Auto

Description: Play chime sound at startup.

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

7. `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. In general the necessary delay may be as long as 0.5 seconds.

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

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

3. KeyMergeThreshold

Type: plist integer

Failsafe: 0

Description: Assume simultaneous combination for keys submitted within this timeout in milliseconds.

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

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

4. KeySupport

Type: plist boolean

Failsafe: false

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

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

5. KeySupportMode

Type: plist string

Failsafe: 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.

6. KeySwap

Type: plist boolean

Failsafe: false

Description: Swap `Command` and `Option` keys during submission.

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

7. PointerSupport

Type: plist boolean

Failsafe: false

Description: Enable internal pointer driver.

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

8. PointerSupportMode

Type: plist string

Failsafe: Empty

Description: Set OEM protocol used for internal pointer driver.

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

9. TimerResolution

Type: plist integer

Failsafe: 0

Description: Set architecture timer resolution.

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

The recommended value is 50000 (5 milliseconds) or slightly higher. Select ASUS Z87 boards use 60000 for the interface. Apple boards use 100000. 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 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**, set **Resolution** to **Max**. **BuiltinText** variant is an alternative **BuiltinGraphics** for some very old and buggy laptop firmware, which can only draw in **Text** mode.

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

Note: Some Macs, namely **MacPro5,1**, may have broken console output with newer GPUs, and thus only **BuiltinGraphics** may work for them.

2. ConsoleMode

Type: plist string

Failsafe: Empty (Maintain current console mode)

Description: Sets console output mode as specified with the **WxH** (e.g. 80x24) formatted string.

Set to **Max** to attempt using the largest available console mode. This option is currently ignored as the **Builtin** text renderer only supports one console mode.

Note: This field is best left empty on most types of firmware.

3. Resolution

Type: plist string

Failsafe: Empty (Maintain current screen resolution)

Description: Sets console output screen resolution.

- Set to **WxH@Bpp** (e.g. 1920x1080@32) or **WxH** (e.g. 1920x1080) formatted string to request custom resolution from GOP if available.

- Set to **Max** to attempt using the largest available screen resolution.

On HiDPI screens `APPLE_VENDOR_VARIABLE_GUID UIscale` NVRAM variable may need to be set to `02` to enable HiDPI scaling in `Builtin` text renderer, FileVault 2 UEFI password interface, and boot screen logo. Refer to Recommended Variables section for more details.

Note: This will fail when console handle has no GOP protocol. When the firmware does not provide it, it can be added with `ProvideConsoleGop` set to `true`.

4. ForceResolution

Type: plist boolean

Failsafe: false

Description: Forces `Resolution` to be set in cases where the desired resolution is not available by default, such as on legacy Intel GMA and first generation Intel HD Graphics (Ironlake/Arrandale). Setting `Resolution` to **Max** will try to pull the largest available resolution from the connected display's EDID.

Note: This option depends on the `OC_FORCE_RESOLUTION_PROTOCOL` protocol being present. This protocol is currently only supported by `OpenDuetPkg`. The `OpenDuetPkg` implementation currently only supports Intel iGPUs.

5. ClearScreenOnModeSwitch

Type: plist boolean

Failsafe: false

Description: Some types of firmware only clear part of the screen when switching from graphics to text mode, leaving a fragment of previously drawn images visible. This option fills the entire graphics screen with black colour before switching to text mode.

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

6. DirectGopRendering

Type: plist boolean

Failsafe: false

Description: Use builtin graphics output protocol renderer for console.

On some types of firmware, such as on the `MacPro5,1`, this may provide better performance or fix rendering issues. However, this option is not recommended unless there is an obvious benefit as it may result in issues such as slower scrolling.

7. IgnoreTextInGraphics

Type: plist boolean

Failsafe: false

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.

8. 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.

9. 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 broken GOP protocol on console handle, which may be the case on MacPro5,1 with newer GPUs.

10. **ReconnectOnResChange**

Type: plist boolean

Failsafe: false

Description: Reconnect console controllers after changing screen resolution.

On some types of 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.

11. **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.

12. **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 to play sounds and signals for screen reading or audible error reporting. Supported protocols are beep generation and VoiceOver. VoiceOver protocol is specific to Gibraltar machines (T2) and is not supported before macOS High Sierra (10.13). Instead older macOS versions use AppleHDA protocol, which is currently not implemented.

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

Note: 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 or legacy Macs.

Note: Some Macs, namely MacPro5,1, do have APFS compatibility, but their Apple Boot Policy protocol contains recovery detection issues, thus using this option is advised on them as well.

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 or 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 or legacy Macs to improve compatibility with legacy EfiBoot such as the one in macOS 10.4.
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 select 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 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 safe operation of the system must be specified in your configuration.

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 or 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.
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 VMs and legacy Macs.
Note: Several virtual machines 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 flawed 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
Description: Replaces unicode collation services with builtin versions. Set to **true** to ensure UEFI Shell compatibility on platforms with flawed unicode collation implementations. Legacy Insyde and APTIO platforms on Ivy Bridge, and earlier, are typically affected.

11.12 Quirks Properties

1. **DisableSecurityPolicy**
Type: plist boolean
Failsafe: false
Description: Disable platform security policy.
Note: This setting disables various security features of the firmware, defeating the purpose of any kind of Secure Boot. Do NOT enable if you use UEFI Secure Boot.
2. **ExitBootServicesDelay**
Type: plist integer
Failsafe: 0
Description: Adds delay in microseconds after **EXIT_BOOT_SERVICES** event.

This is a very rough workaround to circumvent the **Still waiting for root device** message on some APTIO IV firmware (ASUS Z87-Pro) particularly when using FileVault 2. It appears that for some reason, they execute code in parallel to **EXIT_BOOT_SERVICES**, which results in the SATA controller being inaccessible from macOS. A better approach is required and Acidanthera is open to suggestions. Expect 3 to 5 seconds to be adequate when this quirk is needed.
3. **IgnoreInvalidFlexRatio**
Type: plist boolean
Failsafe: false

Description: Some types of firmware (such as APTIO IV) may contain invalid values in the MSR_FLEX_RATIO (0x194) MSR register. These values may cause macOS boot failures on Intel platforms.

Note: While the option is not expected to harm unaffected firmware, its use is recommended only when specifically required.

4. ReleaseUsbOwnership

Type: plist boolean

Failsafe: false

Description: Attempt to detach USB controller ownership from the firmware driver. While most types of firmware manage to do this properly, or at least have an option for this, some do not. As a result, the operating system may freeze upon boot. Not recommended unless specifically required.

5. RequestBootVarRouting

Type: plist boolean

Failsafe: false

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

This quirk requires OC_FIRMWARE_RUNTIME protocol implemented in `OpenRuntime.efi`. The quirk lets default boot entry preservation at times when the firmware deletes incompatible boot entries. In summary, this quirk is required to reliably use the Startup Disk preference pane in firmware that is not compatible with macOS boot entries by design.

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

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

6. TscSyncTimeout

Type: plist integer

Failsafe: 0

Description: Attempts to perform TSC synchronisation with a specified timeout.

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

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

Note: This quirk cannot replace the kernel driver because it cannot operate in ACPI S3 (sleep wake) mode and because the UEFI firmware only provides very limited multicore support which prevents precise updates of the MSR registers.

7. UnblockFsConnect

Type: plist boolean

Failsafe: false

Description: Some types of firmware block partition handles by opening them in By Driver mode, resulting in an inability to install File System protocols.

Note: This quirk is useful in cases where unsuccessful drive detection results in an absence of boot entries.

11.13 ReservedMemory Properties

1. Address

Type: plist integer

Failsafe: 0

Description: Start address of the reserved memory region, which should be allocated as reserved effectively marking the memory of this type inaccessible to the operating system.

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

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

2. **Comment**

Type: `plist string`

Failsafe: Empty

Description: Arbitrary ASCII string used to provide human readable reference for the entry. Whether this value is used is implementation defined.

3. **Size**

Type: `plist integer`

Failsafe: 0

Description: Size of the reserved memory region, must be page-aligned (4 KBs).

4. **Type**

Type: `plist string`

Failsafe: `Reserved`

Description: Memory region type matching the UEFI specification memory descriptor types. Mapping:

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

5. **Enabled**

Type: `plist boolean`

Failsafe: `false`

Description: This region will not be reserved unless set to `true`.

12 Troubleshooting

12.1 Legacy Apple OS

Older operating systems may be more complicated to install, but sometimes can be necessary to use for all kinds of reasons. While a compatible board identifier and CPUID are the obvious requirements for proper functioning of an older operating system, there are many other less obvious things to consider. This section tries to cover 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 Apple Partitioning Scheme and require `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 R0
hdiutil mount Original.dmg -noverify -noautoopen -noautoopenrw -noautofsck -mountpoint R0
cp R0/.DS_Store DS_STORE
hdiutil detach R0 -force
rm -rf R0
hdiutil convert Original.dmg -format UDRW -o ReadWrite.dmg
mkdir RW
xattr -c OSInstall.mpkg
```

```
hdiutil mount ReadWrite.dmg -noverify -noautoopen -noautoopenrw -noautofsck -mountpoint RW
cp OSInstall.mpkg RW/System/Installation/Packages/OSInstall.mpkg
killall Finder fsevents
rm -rf RW/.fsevents
cp DS_STORE RW/.DS_Store
hdiutil detach RW -force
rm -rf DS_STORE RW
hdiutil convert ReadWrite.dmg -format UDZO -o ReadOnly.dmg
```

12.1.4 macOS 10.5

- All previous issues apply.
- This macOS version does not support `x86_64` kernel and requires `i386` kernel extensions and patches.
- This macOS version uses the first (V1) version of `prelinkedkernel`, which has kext symbol tables corrupted by the kext tools. This nuance renders `prelinkedkernel` kext injection impossible in OpenCore. `Mkext` kext injection will still work without noticeable performance drain and will be chosen automatically when `KernelCache` is set to `Auto`.
- Last released installer image for macOS 10.5 is macOS 10.5.7 build 9J3050 (for `MacBookPro5,3`). Unlike the others, this image is not limited to the target model identifiers and can be used as is. The original 9J3050 image can be found here (MEGA Mirror), assuming macOS 10.5 is legally owned. Read `DIGEST.txt` for more details. Note that this is the earliest tested version of macOS 10.5 with OpenCore.

12.1.5 macOS 10.4

- All previous issues apply.
- This macOS version has a hard requirement to access all the optional packages on the second DVD disk installation media, requiring either two disks or USB media installation.
- Last released installer images for macOS 10.4 are macOS 10.4.10 builds `8R4061a` (for `MacBookPro3,1`) and `8R4088` (for `iMac7,1`). These images are limited to their target model identifiers as on newer macOS versions. Modified `8R4088` images (with `ACDT` suffix) without model restrictions can be found here (MEGA Mirror), assuming macOS 10.4 is legally owned. Read `DIGEST.txt` for more details. Note that these are the earliest tested versions of macOS 10.4 with OpenCore.

12.2 UEFI Secure Boot

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

1. Enable Apple Secure Boot by setting `SecureBootModel` to run macOS. Note, that not every macOS is compatible with Apple Secure Boot and there are several other restrictions as explained in Apple Secure Boot section.
2. Disable DMG loading by setting `DmgLoading` to `Disabled` if users have concerns of loading old vulnerable DMG recoveries. This is **not** required, but recommended. For the actual tradeoffs see the details in DMG loading section.
3. Make sure that APFS JumpStart functionality restricts the loading of old vulnerable drivers by setting `MinDate` and `MinVersion` to 0. More details are provided in APFS JumpStart section. An alternative is to install `apfs.efi` driver manually.
4. Make sure that `Force` driver loading is not needed and all the operating systems are still bootable.
5. Make sure that `ScanPolicy` restricts loading from undesired devices. It is a good idea to prohibit all removable drivers or unknown filesystems.

6. Sign all the installed drivers and tools with 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 usually not needed).
- 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 (hack for Windows 10), HFSExplorer, MacDrive, Paragon APFS, Paragon HFS+, TransMac, etc. Remember to never ever attempt to modify Apple file systems from Windows as this often leads to irrecoverable data loss.

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

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

```
PS C:\gdisk> .\gdisk64.exe \\.\\physicaldrive0
```

```
GPT fdisk (gdisk) version 1.0.4
```

```
Command (? for help): p
```

```
Disk \\.\\physicaldrive0: 419430400 sectors, 200.0 GiB
```

```
Sector size (logical): 512 bytes
```

```
Disk identifier (GUID): DEC57EB1-B3B5-49B2-95F5-3B8C4D3E4E12
```

```
Partition table holds up to 128 entries
```

```
Main partition table begins at sector 2 and ends at sector 33
```

```
First usable sector is 34, last usable sector is 419430366
```

```
Partitions will be aligned on 2048-sector boundaries
```

```
Total free space is 4029 sectors (2.0 MiB)
```

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

```
Command (? for help): c
```

```
Partition number (1-4): 4
```

```
Enter name: BOOTCAMP
```

```
Command (? for help): w
```

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

```
Do you want to proceed? (Y/N): Y
```

```
OK; writing new GUID partition table (GPT) to \\.\\physicaldrive0.
```

```
Disk synchronization succeeded! The computer should now use the new partition table.
```

```
The operation has completed successfully.
```

Listing 4: Relabeling Windows volume

How to choose Windows BOOTCAMP with custom NTFS drivers?

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

12.4 Debugging

Similar to other projects working with hardware OpenCore supports auditing and debugging. The use of NOOPT or DEBUG build modes instead of RELEASE can produce a lot more debug output. With NOOPT source level debugging with GDB or IDA Pro is also available. For GDB check OpenCore Debug page. For IDA Pro, 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 debug boot failure?

Normally it is enough to obtain the actual error message. 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 hacks in Quirks sections one by one. For early boot troubleshooting, for instance, when OpenCore menu does not appear, using UEFI Shell (bundled with OpenCore) may help to see early debug messages.

2. How to debug macOS boot failure?

- Refer to `boot-args` values such as `debug=0x100`, `keepyms=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 customise boot entries?

OpenCore follows standard Apple Bless model and extracts the entry name from `.contentDetails` and `.disk_label.contentDetails` files in the booter directory if present. These files contain an ASCII string with an entry title, which may then be customised by the user.

4. How to choose the default boot entry?

OpenCore uses the primary UEFI boot option to select the default entry. This choice can be altered from UEFI Setup, with the macOS Startup Disk preference, or the Windows Boot Camp Control Panel. Since choosing OpenCore’s `BOOTx64.EFI` as a primary boot option limits this functionality in addition to several 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 OpenCore Boot Picker and choose the entry, it will have a `(dmg)` suffix. Custom name may be created by providing `.contentDetails` file.

To download recovery online `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.

7. Can I use this on Apple hardware or virtual machines?

Sure, most relatively modern Mac models including MacPro5,1 and virtual machines are fully supported. Even though there are little to none specific details relevant to Mac hardware, some ongoing instructions can be found on MacRumors.com.

8. Why must Find&Replace patches be equal in size?

For machine code (x86 code) it is not possible to do differently sized replacements due to relative addressing. For ACPI code this is risky, and is technically equivalent to ACPI table replacement, thus not implemented. More detailed explanation can be found on AppleLife.ru or in the ACPI section of this document.

9. How can I decide which Booter quirks to use?

These quirks originate from AptioMemoryFix driver but provide a wider set of changes specific to modern systems. Note, that OpenRuntime driver is required for most configurations. To get a configuration similar to AptioMemoryFix the following set of quirks should be enabled:

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

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

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