OpenCore

Reference Manual

[2019.04.10]

Contents

1 Introduction

This document provides information on OpenCore user configuration file format used to setup the correct functioning of macOS operating system.

1.1 Known defects

For OpenCore issues please refer to Acidanthera Bugtracker. Currently this file has the following entries not completed:

- Known `UEFI` driver list is incomplete.
- Not all `NVRAM` variables are properly described (e.g. boot-args).

2 Generic Terms

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

3 Overview

3.1 Configuration Terms

- `OC config` — OpenCore Configuration file in `plist` format named `config.plist`. It has to provide extensible way to configure OpenCore and is structured to be separated into multiple named sections situated in the root `plist dictionary`. These sections are permitted to 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 `#` symbol (e.g. `#Hello`) are also considered `valid keys` and behave as comments, effectively discarding their value, which is still required to be a valid `plist object`. All other `plist keys` are not valid, and their presence yields to `undefined behaviour`.
- `valid value` — valid `plist object` of `OC config` described in this document that matches all the additional requirements in specific `plist object` description 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` description (e.g. value range), or missing from the corresponding collection. `Invalid value` is 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 value`, applying incompatible value to the host system may yield to `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. Implementation must stop the boot process from going any further until next host system boot. It is allowed but not required to perform cold reboot or show any warning message.
- `undefined behaviour` — behaviour not prescribed by this document. Implementation is allowed to take any measures including but not limited to `fatal behaviour`, assuming any states or values, or ignoring, unless these measures negatively affect system security in general.

3.2 Configuration Processing

`OC config` is guaranteed to be processed at least once if it was found. Depending on OpenCore bootstrapping mechanism multiple `OC config` files may lead to reading any of them. No `OC Config` may be present on disk, in which case all the values read follow the rules of `invalid value` and `optional value`.

`OC config` has size, nesting, and key amount limitations. `OC config` size does not exceed `16 MBs`. `OC config` has no more than `8` nesting levels. `OC config` has 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` file leads to `undefined behaviour`. Examples of malformed `OC config` 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` and continue as if no `OC config` was present. For forward compatibility it is recommended but not required for the implementation to warn about the use of `invalid values`. Recommended practice of 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>`)

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

3.3 Configuration Structure

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

- Add provides support for data addition.
- Block provides support for data removal or ignorance.
- Patch provides support for data modification.
- Quirks provides support for specific hacks.

Root configuration entries consist of the following:

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

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

3.4 Directory Structure

When directory boot is used the directory structure used should follow the description on Directory Structure figure. Available entries include:

- BOOTx64.efi
  Initial booter, which loads OpenCore.efi unless it was already started as a driver.
- ACPI
  Directory used for storing supplemental ACPI information for ACPI section.
- Drivers
  Directory used for storing supplemental UEFI drivers for UEFI section.
- Kexts
  Directory used for storing supplemental kernel information for Kernel section.
- OpenCore.efi
  Main booter driver responsible for operating system loading.
- config.hash
  Hashes for all files potentially loadable by OC Config.
- config.plist
  OC Config.
- config.sig
  Signature for config.hash.

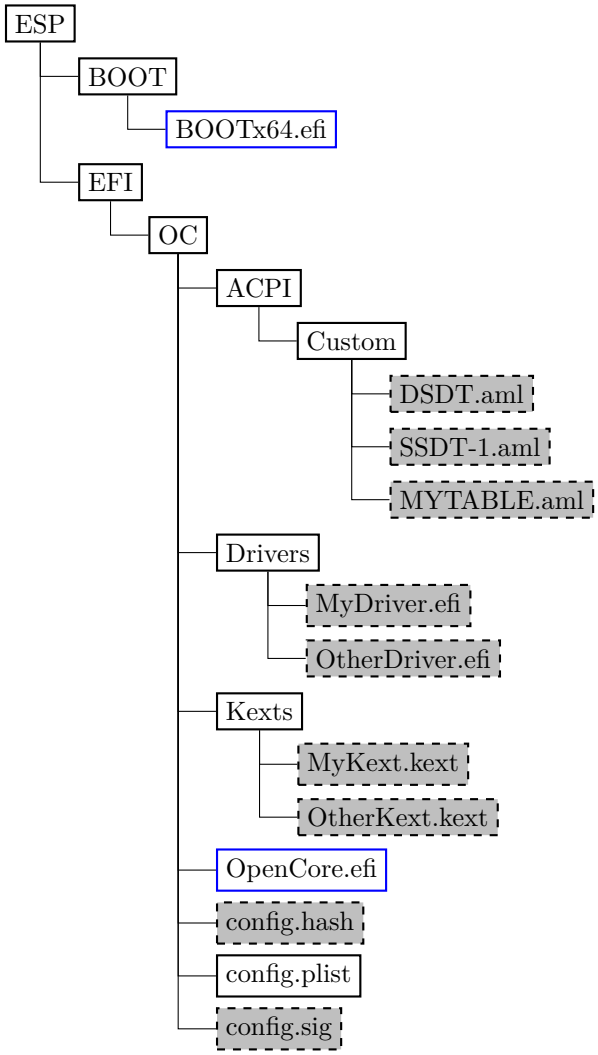

Figure 1. Directory Structure

3.5 Contribution

OpenCore can be compiled as an ordinary EDK II package with UDK 2018.

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

Required external package dependencies include EfiPkg and OcSupportPkg.

To compile with `XCODE5`, besides Xcode, one should also install NASM and MTOC. Example command sequence may look as follows:

```
git clone https://github.com/tianocore/edk2 -b UDK2018 UDK
cd UDK
git clone https://github.com/acidanthera/EfiPkg
git clone https://github.com/acidanthera/OcSupportPkg
git clone https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

4 ACPI

4.1 Introduction

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

4.2 Properties

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

   Designed to be filled with string filenames meant to be loaded as ACPI tables. Example values include `DSDT.aml`, `SSDT-8.aml`, `SSDT-USBX.aml`, etc. ACPI table load order follows the item order in the array.

   Note: all values but `DSDT.aml` insert new ables into ACPI stack. `DSDT.aml`, unlike the rest, performs replacement of DSDT table.

2. `Block`
   Type: `plist array`
   Default value: Empty
   Description: Remove selected tables from ACPI stack.

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

3. `Patch`
   Type: `plist array`
   Default value: Empty
   Description: Perform binary patches in ACPI tables after table addition and removal.

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

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

4.3 Block Properties

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

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

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

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

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

4.4 Patch Properties

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

2. `Count`
   Type: `plist integer`
   Default value: `0`
   Description: Number of patch occurrences to apply. `0` applies the patch to all occurrences found.

3. `Enabled`
   Type: `plist boolean`
   Default value: `false`
   Description: This ACPI patch will not be used unless set to `true`.

4. `Find`
   Type: `plist data`
   Default value: Empty data
   Description: Data to find. Must equal to `Replace` in size.

5. `Limit`
   Type: `plist integer`
   Default value: `0`
   Description: Maximum number of bytes to search for. Can be set to `0` to look through the whole ACPI table.

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

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

8. `Replace`
   Type: `plist data`
   Default value: Empty data
   Description: Replacement data of one or more bytes.

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

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

11. `TableLength`
    Type: `plist integer`

Default value: `0`
Description: Match table size to be equal to this value unless `0`.

12. `TableSignature`
Type:
textttplist data, 4 bytes
Default value: All zero
Description: Match table signature to be equal to this value unless all zero.

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

- Avoid renaming devices with ACPI patches. This may fail or perform improper renaming of unrelated devices (e.g. `EC` and `EC0`), be unnecessary, or even fail to rename devices in select tables. For ACPI consistency it is much safer to rename devices at I/O Registry level, as done by WhateverGreen.
- Avoid patching `_OSI` to support a higher level of feature sets unless absolutely required. Commonly this enables a number of hacks on APTIO firmwares, which result in the need to add more patches. Modern firmwares generally do not need it at all, and those that do are fine with much smaller patches.
- Try to avoid hacky changes like renaming `_PWR` or `_DSM` whenever possible.

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

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

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

4.5 Quirks Properties

1. `FadtEnableReset`
Type: `plist boolean`
Default value: `false`
Description: Provide reset register and flag in FADT table to enable reboot and shutdown on legacy hardware. Not recommended unless required.

2. `IgnoreForWindows`
Type: `plist boolean`
Default value: `false`
Description: Disable all sorts of ACPI modifications when booting Windows operating system.

   This flag implements a quick workaround for those, who made their ACPI tables incompatible with Windows, but need it right now. Not recommended, as ACPI tables must be compatible with any operating system regardless of the changes.

   Note: This option may be removed in the future.

3. `NormalizeHeaders`
Type: `plist boolean`
Default value: `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).

4. `RebaseRegions`
Type: `plist boolean`
Default value: `false`
Description: Attempt to heuristically relocate ACPI memory regions. Not recommended.

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

usually grouped in regions (e.g. `OperationRegion`). Changing firmware settings or hardware configuration, upgrading or patching the firmware inevitably leads to changes in dynamically generated ACPI code, which sometimes lead to the shift of the addresses in aforementioned `OperationRegion` constructions.

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

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

5 DeviceProperties

5.1 Introduction

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

5.2 Properties

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

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

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

3. `Quirks`
   Type: `plist dict`
   Description: Apply individual device property quirks described in Quirks Properties section below.

5.3 Quirks Properties

1. `ReinstallProtocol`
   Type: `plist boolean`
   Default value: `false`
   Description: Reinstalls device property protocol (and drops all previous properties) if it was already installed.

5.4 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.

6 Kernel

6.1 Introduction

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

6.2 Properties

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

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

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

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

3. `Patch`
   Type: `plist array`
   Default value: Empty
   Description: Perform binary patches in kernel and drivers prior to driver addition and removal (FIXME: consistency with ACPI?).

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

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

6.3 Add Properties

1. `BundleName`
   Type: `plist string`
   Default value: Empty string
   Description: Kext bundle name (e.g. `Lilu.kext`).

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

3. `Enabled`
   Type: `plist boolean`
   Default value: `false`
   Description: This kernel driver will not be added unless set to `true`.

4. `ExecutablePath`
   Type: `plist string`
   Default value: Empty string
   Description: Kext executable path relative to bundle (e.g. `Contents/MacOS/Lilu`).

5. `MatchKernel`
   Type: `plist string`

Default value: Empty string
Description: Blocks kernel driver on selected macOS version only. The selection happens based on prefix match with the kernel version, i.e. `16.7.0` will match macOS 10.12.6 and `16.` will match any macOS 10.12.x version.

6. `PlistPath`
   Type: `plist string`
   Default value: Empty string
   Description: Kext `Info.plist` path relative to bundle (e.g. `Contents/Info.plist`).

6.4 Block Properties

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

2. `Enabled`
   Type: `plist boolean`
   Default value: `false`
   Description: This kernel driver will not be blocked unless set to `true`.

3. `Identifier`
   Type: `plist string`
   Default value: Empty string
   Description: Kext bundle identifier (e.g. `com.apple.driver.AppleTyMCEDriver`).

4. `MatchKernel`
   Type: `plist string`
   Default value: Empty string
   Description: Blocks kernel driver on selected macOS version only. The selection happens based on prefix match with the kernel version, i.e. `16.7.0` will match macOS 10.12.6 and `16.` will match any macOS 10.12.x version.

6.5 Patch Properties

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

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

3. `Count`
   Type: `plist integer`
   Default value: `0`
   Description: Number of patch occurrences to apply. `0` applies the patch to all occurrences found.

4. `Enabled`
   Type: `plist boolean`
   Default value: `false`
   Description: This kernel patch will not be used unless set to `true`.

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

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

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

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

9. `MatchKernel`
   Type: `plist string`
   Default value: Empty string
   Description: Adds kernel driver to selected macOS version only. The selection happens based on prefix match with the kernel version, i.e. `16.7.0` will match macOS 10.12.6 and `16.` will match any macOS 10.12.x version.

10. `Replace`
    Type: `plist data`
    Default value: Empty data
    Description: Replacement data of one or more bytes.

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

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

6.6 Quirks Properties

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

   Note: This option should avoided whenever possible. Modern firmwares provide `CFG Lock` setting, disabling which is much cleaner. More details about the issue can be found in VerifyMsrE2 notes.

2. `ExternalDiskIcons`
   Type: `plist boolean`
   Default value: `false`
   Description: Apply icon type patches to IOAHCIPort.kext to force internal disk icons for all AHCI disks.

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

3. `ThirdPartyTrim`
   Type: `plist boolean`
   Default value: `false`
   Description: Patch IOAHCIFamily.kext to force TRIM command support on AHCI SSDs.

   Note: This option should avoided whenever possible. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called `trimforce`.

4. `XhciPortLimit`
   Type: `plist boolean`
   Default value: `false`
   Description: Patch various kexts (AppleUSBXHCI.kext, AppleUSBXHCIPCI.kext, IOUSBHostFamily.kext) to remove USB port count limit of 15 ports.

   Note: This option should 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 Misc

7.1 Introduction

This section contains miscellaneous configuration entries for OpenCore behaviour that does not go to any other sections

7.2 Properties

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

7.3 Debug Properties

1. `Delay`
   Type: `plist integer`
   Default value: `0`
   Description: Delay in microseconds performed after every printed line of visible logging output like console, Data Hub, or serial port.

2. `Target`
   Type: `plist integer`
   Default value: `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:

   - `1` — Enable logging, otherwise all log is discarded.
   - `2` — Enable basic console (onscreen) logging.
   - `4` — Enable logging to Data Hub.
   - `8` — Enable serial port logging.
   - `16` — Enable UEFI variable logging.
   - `32` — Enable non-volatile UEFI variable logging.
   - `64` — Enable logging to file.

   Note: 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).

   Note: To obtain Data Hub log use the following command in macOS:

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

   Note: UEFI variable log may get truncated on some firmwares. Using non-volatile flag will write the log to NVRAM flash after every printed line. To obtain UEFI variable log use the following command in macOS:

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

   Note: File logging is currently not implemented.

8 NVRAM

8.1 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`)

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.

8.2 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 and not blocked. To overwrite a variable add it to `Block` section. This approach enables to provide default values till the operating system takes the lead.

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

2. `Block`
   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.

To read NVRAM variable value from macOS one could use `nvram` by concatenating variable GUID and name separated by `:` symbol. For example, `nvram 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args`.

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

8.3 Mandatory 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.

8.4 Recommended Variables

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

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:csr-active-config`
  32-bit System Integrity Protection bitmask. Declared in XNU source code in csr.h.

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeatures`
  Combined `FirmwareFeatures` and `ExtendedFirmwareFeatures`. Present on newer Macs to avoid extra parsing of SMBIOS tables
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask`
  Combined `FirmwareFeaturesMask` and `ExtendedFirmwareFeaturesMask`. Present on newer Macs to avoid extra parsing of SMBIOS tables.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID`
  Hardware `BoardProduct` (e.g. `Mac-35C1E88140C3E6CF`). Not present on real Macs, but used to avoid extra parsing of SMBIOS tables, especially in boot.efi.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB`
  Hardware `BoardSerialNumber`. Override for MLB. Present on newer Macs (2013+ at least).
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM`
  Hardware ROM. Override for ROM. Present on newer Macs (2013+ at least).
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:prev-lang:kbd`
  ASCII string defining default keyboard layout. Format is `lang-COUNTRY:keyboard`, e.g. `ru-RU:19456` for Mac keyboard. Also accepts short forms: `ru:19456` or `ru:0`. Full decoded list of keyboards in `AppleKeyboardLayouts-L.dat` can be found on AppleLife.
- `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`
  8-bit integer defining boot.efi user interface scaling. Should be 1 for normal screens and 2 for HDPI screens.

8.5 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.
  - FIXME: document several known values! debug, keepsyms, slide, -v, -s, -x, cpus=x, io=x, kextlog=x, -nehalem_error_disable -no_compat_check nvda_drv=1, etc?
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg`
  Booter arguments, similar to `boot-args` but for boot.efi. Accepts a set of arguments, which are hexadecimal 64-bit values with or without 0x prefix primarily for logging control:
  - `log=VALUE`
    - `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`
    - `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` — Verbosity level of DEBUG output. Everything but `0x80000000` is stripped from the binary, and this is the default value.
  - `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.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg-once`
  Booter arguments override removed after first launch. Otherwise equivalent to `bootercfg`.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:fmm-computer-name`
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:nvda_drv`

9 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 `MacInfoPkg` package, which itself generates a set of interfaces based on a database in YAML format. These fields are written to three select destinations:

- SMBIOS
- Data Hub
- NVRAM

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

9.1 Properties

1. `Automatic`
   Type: `plist boolean`
   Default value: `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` and `DataHub` data is unused.

   FIXME: Currently unsupported.

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

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

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

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

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

5. `UpdateSMBIOSMode`
   Type: `plist string`
   Default value: `Auto`
   Description: Update SMBIOS fields approach:

   - `Auto` — `Overwrite` if new size is <= than the page-aligned original and there are no issues with legacy region unlock. `Create` otherwise.
   - `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 first SMBIOS table (`gEfiSmbiosTableGuid`) to `gOcCustomSmbiosTableGuid` to workaround firmwares overwriting SMBIOS contents at ExitBootServices. Otherwise equivalent to `Create`. Requires patching AppleSmbios.kext and AppleACPIPlatform.kext to read from another GUID: `"EB9D2D31"` -> `"EB9D2D35"` (in ASCII).

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

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

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

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

9.2 Generic Properties

1. `SystemProductName`
   Type: `plist string`
   Default value: `MacPro6,1`
   Description: Refer to SMBIOS `SystemProductName`.

2. `SystemSerialNumber`
   Type: `plist string`
   Default value: `OPENCORE_SN1`
   Description: Refer to SMBIOS `SystemSerialNumber`.

3. `SystemUUID`
   Type: `plist string`, GUID
   Default value: OEM specified
   Description: Refer to SMBIOS `SystemUUID`.

4. `MLB`
   Type: `plist string`
   Default value: `OPENCORE_MLB_SN11`
   Description: Refer to SMBIOS `BoardSerialNumber`.

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

9.3 DataHub Properties

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

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

3. `SystemSerialNumber`
   Type: `plist string`

Default value: Not installed
Description: Sets `SystemSerialNumber` in `gEfiMiscSubClassGuid`. Value found on Macs is equal to SMBIOS `SystemSerialNumber` in Unicode.

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

5. `BoardProduct`
   Type: `plist string`
   Default value: Not installed
   Description: Sets `board-id` in `gEfiMiscSubClassGuid`. Value found on Macs is equal to SMBIOS `BoardProduct` in ASCII.

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

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

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

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

9. `FSBFrequency`
   Type: `plist integer`, 64-bit
   Default value: Automatic
   Description: Sets `FSBFrequency` in `gEfiProcessorSubClassGuid`. Sets CPU FSB frequency.

10. `ARTFrequency`
    Type: `plist integer`, 64-bit
    Default value: Not installed
    Description: Sets `ARTFrequency` in `gEfiProcessorSubClassGuid`. Sets CPU ART frequency, Skylake and newer.

11. `DevicePathsSupported`
    Type: `plist data`, 1 byte
    Default value: Not installed

Description: Sets DevicePathsSupported in gEfiMiscSubClassGuid. Value found on Macs is 01. Read by AppleACPIPlatform.kext.

12. SmcRevision
Type: plist data, 6 bytes
Default value: 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
Default value: 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
Default value: Not installed
Description: Sets RPlt in gEfiMiscSubClassGuid. Custom property read by VirtualSMC or FakeSMC to generate SMC RPlt key.

9.4 PlatformNVRAM Properties

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

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

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

9.5 SMBIOS Properties

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

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

```
Apple ROM Version
 BIOS ID:      MBP151.88Z.F000.B00.1811142212
 Model:        MBP151
```

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

3. `BIOSReleaseDate`
   Type: `plist string`
   Default value: 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`
   Default value: OEM specified
   SMBIOS: System Information (Type 1) — Manufacturer
   Description: OEM manufacturer of the particular board. Shall not be specified unless strictly required. Should *not* contain `Apple Inc.`, as this confuses numerous services present in the operating system, such as firmware updates, eficheck, as well as kernel extensions developed in Acidanthera, such as Lilu and its plugins.

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

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

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

7. `SystemSerialNumber`
   Type: `plist string`
   Default value: 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
   Default value: 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`
   Default value: 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`
    Default value: OEM specified
    SMBIOS: System Information (Type 1) — Family
    Description: Family name. May look like `iMac Pro`.

11. `BoardManufacturer`
    Type: `plist string`
    Default value: OEM specified
    SMBIOS: Baseboard (or Module) Information (Type 2) - Manufacturer
    Description: Board manufacturer. All rules of `SystemManufacturer` do apply.

12. `BoardProduct`
    Type: `plist string`
    Default value: 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`
    Default value: 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`
    Default value: 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`
    Default value: 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`
    Default value: 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`
    Default value: 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`
    Default value: OEM specified
    SMBIOS: System Enclosure or Chassis (Type 3) — Manufacturer
    Description: Board manufacturer. All rules of `SystemManufacturer` do apply.

19. `ChassisType`
    Type: `plist integer`
    Default value: 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`
    Default value: OEM specified
    SMBIOS: System Enclosure or Chassis (Type 3) — Version
    Description: Should match `BoardProduct`.

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

22. `ChassisAssetTag`
    Type: `plist string`
    Default value: 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`
    Default value: `0`
    SMBIOS: `APPLE_SMBIOS_TABLE_TYPE133` - `PlatformFeature`
    Description: Platform features bitmask. Refer to AppleFeatures.h for more details.

24. `FirmwareFeatures`
    Type: `plist integer`, 64-bit
    Default value: `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`.

25. `FirmwareFeaturesMask`
    Type: `plist integer`, 64-bit
    Default value: `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`.

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

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

10 UEFI

10.1 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.

10.2 Properties

1. `ConnectDrivers`
   Type: `plist boolean`
   Default value: `NO`
   Description: Perform UEFI controller connection after driver loading. This option is useful for loading filesystem drivers, which usually follow UEFI driver model, and may not start by themselves. While effective, this option is not necessary with e.g. APFS loader driver, and may slightly slowdown the boot.

2. `Drivers`
   Type: `plist array`
   Default value: None
   Description: Load selected drivers from `OC/Drivers` directory.

   Designed to be filled with string filenames meant to be loaded as UEFI drivers. Depending on the firmware a different set of drivers may be required. Loading an incompatible driver may lead your system to unbootable state or even cause permanent firmware damage. Some of the known drivers include:

   FIXME: Write

3. `Quirks`
   Type: `plist dict`
   Default value: None
   Description: Apply individual firmware quirks described in Quirks Properties section below.

10.3 Quirks Properties

1. `DisableWatchDog`
   Type: `plist boolean`
   Default value: `NO`
   Description: Select firmwares may not succeed in quickly booting the operating system, which results in watch dog timer aborting the process. This option turns off watch dog timer.

   Note: This option is believed to be unnecessary on modern firmwares, yet may be safer to turn on as system performance across the boots is not constant.

2. `IgnoreInvalidFlexRatio`
   Type: `plist boolean`
   Default value: `NO`
   Description: Select firmwares, namely APTIO IV, may contain invalid values in `MSR_FLEX_RATIO` (`0x194`) MSR register. These values may cause macOS boot failure on Intel platforms.

   Note: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required.

3. `ProvideConsoleGop`
   Type: `plist boolean`
   Default value: `NO`
   Description: macOS bootloader requires GOP (Graphics Output Protocol) to be present on console handle. This option will install it if missing.

   Note: Some drivers, like AptioMemoryFix, may provide equivalent functionality. These drivers are not guaranteed to adhere to the same logic, and if a quirk is necessary, this option is preferred.

4. `ReleaseUsbOwnership`
   Type: `plist boolean`
   Default value: `false`
   Description: Attempt to detach USB controller ownership from the firmware driver. While most firmwares manage to properly do that, or at least have an option for, select firmwares do not. As a result, operating system may freeze upon boot. Not recommended unless required.

11 Troubleshooting

Good luck.