OpenCore

Reference Manual (0.6~~.6~~.7)

[2021.02.18]

1 Introduction

This document provides information on 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 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`~~`multidata` — value cast to data by the implementation. Permits passing `plist string`, in which case the result is represented by a null-terminated sequence of bytes (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.

- `boot`
  Duet bootstrap loader, which initialises UEFI environment on legacy BIOS firmware and loads `OpenCore.efi` similarly to other bootstrap loaders. Modern Duet bootstrap loader will default to `OpenCore.efi` on the same partition when present.
- `ACPI`
  Directory used for storing supplemental ACPI information for `ACPI` section.
- `Drivers`
  Directory used for storing supplemental `UEFI` drivers for `UEFI` section.
- `Kexts`
  Directory used for storing supplemental kernel information for `Kernel` section.
- `Resources`
  Directory used for storing media resources, such as audio files for screen reader support. See `UEFI Audio Properties` section for more details. This directory also contains image files for graphical user interface. See OpenCanopy section for more details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  Main booter application responsible for operating system loading. The directory `OpenCore.efi` resides is called the `root directory`. By default `root directory` is set to `EFI\OC`, however, when launching `OpenCore.efi` directly or through a custom launcher, other directories containing `OpenCore.efi` can also be 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~~reflect~~, replicate the Configuration Structure described in the previous section on a EFI volume of a GPT partition. While corresponding sections of this document do provide some information 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` ~~, just like any property lists~~ file, as with any property list file, can be edited with any ~~stock textual editor (e.g. nano , vim), but~~ text editor such as 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 will have to be used. `OpenDuetPkg` is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system, `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~~, refer to the `Differences.pdf` document ~~, providing the information about the changes affecting the configuration~~ which provides information about changes to the configuration as compared to the previous release ~~, and~~ as well as to the `Changelog.md` document, ~~containing the~~ which contains a list of modifications across all published updates.

3.3 Contribution

OpenCore can be compiled as an ordinary EDK II package. Since UDK development was abandoned by TianoCore, OpenCore requires the use of EDK II Stable. ~~Currently~~ The currently supported EDK II release is hosted in acidanthera/audk. ~~The required patches for the package are present in~~ 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. ~~Contribution~~ Contributions of clean patches ~~is~~ are welcome. Please do follow EDK II C Codestyle.

To compile with `XCODE5`, besides Xcode, ~~one~~ users should also install NASM and MTOC. The latest Xcode version is recommended for use despite the toolchain name. ~~Example command sequence may look~~ 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
-IInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
-DNO_MSABI_VA_FUNCS=1
```

Listing 2: ECC Configuration

4 ACPI

4.1 Introduction

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

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

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

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

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

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

There are many places providing ACPI tables and workarounds. Commonly used ACPI tables are provided with OpenCore, VirtualSMC, VoodooPS2, and WhateverGreen releases. Besides those there are several third-party instructions commonly found on AppleLife in 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 also are several other places including daliansky's ACPI sample collection, but the quality of the suggested solutions will vary from case to case.

4.2 Properties

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

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

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

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

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

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

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

4.3 Add Properties

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

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

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

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

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

4.4 Delete Properties

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

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

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

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

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

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

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

4.5 Patch Properties

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

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

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

4. `Find`
   Type: `plist data`
   Failsafe: Empty~~data~~
   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. ~~Can be set to `0` to look through the whole ACPI table.~~

6. `Mask`
   Type: `plist data`
   Failsafe: Empty ~~data~~(Ignored)
   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~~ Must be equal to `Replace` in size ~~otherwise~~if set.

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

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

9. `ReplaceMask`
   Type: `plist data`
   Failsafe: Empty ~~data~~(Ignored)
   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~~ Must be equal to `Replace` in size ~~otherwise~~if set.

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

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

12. `TableSignature`
    Type: `plist data`, 4 bytes

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

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

- Avoid renaming devices with ACPI patches. This may fail or perform improper renaming of unrelated devices (e.g. `EC` and `EC0`), be unnecessary, or even fail to rename devices in select tables. For ACPI consistency it is much safer to rename devices at I/O Registry level, as done by WhateverGreen.
- Try to avoid patching `_OSI` to support a higher level of feature sets whenever possible. Commonly this enables a number of hacks on APTIO firmware, which result in the need to add more patches. Modern firmware generally does not need it, and those that do are fine with much smaller patches. However, laptop vendors usually 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 for enabling brightness keys. The conventional process to find these keys usually involves massive modification on DSDT and SSDTs and the debug kext is not stable on newer systems. Please switch to built-in brightness key discovery of BrightnessKeys instead.
- ~~Try to avoid hacky~~ Avoid making ad hoc changes such as renaming `_PRW` or `_DSM` whenever possible.

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

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

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

Note: Patches of different `Find` and `Replace` lengths are unsupported as they may corrupt ACPI tables and make 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 might be taken into account.

4.6 Quirks Properties

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

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

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

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

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

5.3 MmioWhitelist Properties

1. `Address`
   Type: `plist integer`
   Failsafe: `0`
   Description: Exceptional MMIO address, which memory descriptor should be left virtualised (unchanged) by `DevirtualiseMmio`. This means that the firmware will be able to directly communicate with this memory region during operating system functioning, because the region this value is in will be assigned a virtual address.

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

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

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: This address will be devirtualised unless set to `true`.

5.4 Patch Properties

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

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

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

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

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

6. `Identifier`
   Type: `plist string`
   Failsafe: ~~Empty string~~`Any` (Match any booter)
   Description: `Apple` for macOS booter (generally `boot.efi`); or a name with ~~suffix (e.g.~~ a suffix, such as `bootmgfw.efi`~~)~~ , for a specific booter~~; or `Any` / empty string (failsafe) to match any booter~~.

7. `Limit`
   Type: `plist integer`
   Failsafe: `0` (Search the entire booter)
   Description: Maximum number of bytes to search for. ~~Can be set to `0` to look through the whole booter.~~

8. `Mask`
   Type: `plist data`
   Failsafe: Empty ~~data~~(Ignored)
   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~~ Must be equal to `Find` in size ~~otherwise~~if set.

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

10. `ReplaceMask`
    Type: `plist data`
    Failsafe: Empty ~~data~~(Ignored)
    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~~ Must be equal to `Replace` in size ~~otherwise~~if set.

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

5.5 Quirks Properties

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

   Relocation block is a scratch buffer allocated in lower 4 GB to be used for loading the kernel and related structures by EfiBoot on ~~firmwares where lower memory~~ firmware where the lower memory region is otherwise occupied by ~~the (assumedto be~~(assumed) non-runtime data. Right before kernel startup, the relocation block is copied back to lower addresses. Similarly, all the other addresses pointing to relocation block are also carefully adjusted. 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 generally needs `AvoidRuntimeDefrag` 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 ~~this case one may try to use~~ 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 select 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.

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 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, which may inspect the `IODeviceTree` plane without probing (e.g. `Lilu` and its plugins such as `WhateverGreen`) it is particularly important to ensure device presence in the ACPI tables. Failing 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-BRG0.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`~~`multidata` 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.

7. `Scheme`
   Type: `plist dict`
   Description: Define kernelspace operation mode via parameters described in 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~~string~~
   Description: Kext bundle path (e.g. `Lilu.kext` or `MyKext.kext/Contents/PlugIns/MySubKext.kext`).

3. `Comment`
   Type: `plist string`
   Failsafe: Empty~~string~~
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. ~~It is implementation defined whether~~ 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~~string~~
   Description: Kext executable path relative to bundle (e.g. `Contents/MacOS/Lilu`).

6. `MaxKernel`
   Type: `plist string`
   Failsafe: Empty~~string~~
   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:

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

   Kernel version comparison is implemented as follows:

$$\alpha = \begin{cases} ParseDarwinVersion(\texttt{MinKernel}), & \text{If } \texttt{MinKernel} \text{ is valid} \\ 0 & Otherwise \end{cases}$$

$$\beta = \begin{cases} ParseDarwinVersion(\texttt{MaxKernel}), & \text{If } \texttt{MaxKernel} \text{ is valid} \\ \infty & Otherwise \end{cases}$$

$$\gamma = \begin{cases} ParseDarwinVersion(FindDarwinVersion()), & \text{If valid } \texttt{"Darwin Kernel Version"} \text{ is found} \\ \infty & Otherwise \end{cases}$$

$$f(\alpha,\beta,\gamma) = \alpha \leq \gamma \leq \beta$$

   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~~string~~
Description: Adds kernel driver on specified macOS version or newer.

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

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

7.4 Block Properties

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

2. `Comment`
Type: `plist string`
Failsafe: Empty~~string~~
Description: Arbitrary ASCII string used to provide human readable reference for the entry. ~~It is implementation defined whether~~ 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~~string~~
Description: Kext bundle identifier (e.g. `com.apple.driver.AppleTyMCEDriver`).

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

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

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

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

7.5 Emulate Properties

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

This property 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~~string~~
   Description: Emulates CPUID and applies `DummyPowerManagement` on specified macOS version or older.

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

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

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

7.6 Force Properties

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

2. `BundlePath`
   Type: `plist string`
   Failsafe: Empty~~string~~
   Description: Kext bundle path (e.g. `System\Library \Extensions \IONetworkingFamily.kext`).

3. `Comment`
   Type: `plist string`
   Failsafe: Empty~~string~~
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. ~~It is implementation defined whether~~ 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~~string~~
   Description: Kext executable path relative to bundle (e.g. `Contents/MacOS/IONetworkingFamily`).

6. `Identifier`
   Type: `plist string`
   Failsafe: Empty~~string~~
   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~~string~~
   Description: Adds kernel driver on specified macOS version or older.

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

8. `MinKernel`
   Type: `plist string`
   Failsafe: Empty~~string~~
   Description: Adds kernel driver on specified macOS version or newer.

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

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

3. `Comment`
   Type: `plist string`
   Failsafe: Empty~~string~~
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. ~~It is implementation defined whether~~ 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 ~~data~~(Immediate replacement at `Base`)
Description: Data to find. ~~Can be set to empty for immediate replacement at `Base`. Must~~ Must be equal to `Replace` in size ~~otherwise~~if set.

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

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

9. `Mask`
Type: `plist data`
Failsafe: Empty ~~data~~(Ignored)
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~~ Must be equal to `Replace` in size ~~otherwise~~if set.

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

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

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

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

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

13. `ReplaceMask`
Type: `plist data`
Failsafe: Empty ~~data~~(Ignored)
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~~ Must be equal to `Replace` in size ~~otherwise~~if set.

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

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 fragmenation 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. ~~For this one can set this~~ Set this option

- `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 ~~firmwares, Insyde in particular, but possibly others, which cannot handle~~ 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 ~~and~~ by searching as follows, and will fallback to the default icon otherwise:
     – `.VolumeIcon.icns` file at `Preboot` volume ~~directory~~ 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.

     ~~Volume icons can be set in Finder. Note, that enabling this may result in external and internal icons to be indistinguishable.~~ *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:

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, `IO80211Family` 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~~string~~
   Description: Arbitrary ASCII string used as boot arguments (load options) of the specified entry.

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

3. `Comment`
   Type: `plist string`
   Failsafe: Empty~~string~~
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. ~~It is implementation defined whether~~ 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~~string~~
   Description: Human readable entry name displayed in boot picker.

6. `Path`
   Type: `plist string`
   Failsafe: Empty~~string~~
   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 benefitial to some older tools that require text output. By default all the tools are launched in graphics mode. Read more about text modes in 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`~~`multidata` format. GUIDs must be provided in canonic string format in upper or lower case (e.g. `8BE4DF61-93CA-11D2-AA0D-00E098032B8C`).

   Created variables get `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.

- `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+)

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.

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` ~~`data`~~`multidata`, 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: ~~Not installed~~Empty (Not installed)
   Description: Sets `name` in `gEfiMiscSubClassGuid`. Value found on Macs is `platform` in ASCII.

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

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

4. `SystemUUID`
   Type: `plist string`, GUID
   Failsafe: ~~Not installed~~Empty (Not installed)

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

5. `BoardProduct`
   Type: `plist string`
   Failsafe: ~~Not installed~~Empty (Not installed)
   Description: Sets `board-id` in `gEfiMiscSubClassGuid`. Value found on Macs is equal to SMBIOS `BoardProduct` in ASCII.

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

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

   - `0x00000001` — Shutdown cause was a `PWROK` event (Same as `GEN_PMCON_2` bit 0)
   - `0x00000002` — Shutdown cause was a `SYS_PWROK` event (Same as `GEN_PMCON_2` bit 1)
   - `0x00000004` — Shutdown cause was a `THRMTRIP#` event (Same as `GEN_PMCON_2` bit 3)
   - `0x00000008` — Rebooted due to a SYS_RESET# event (Same as `GEN_PMCON_2` bit 4)
   - `0x00000010` — Power Failure (Same as `GEN_PMCON_3` bit 1 `PWR_FLR`)
   - `0x00000020` — Loss of RTC Well Power (Same as `GEN_PMCON_3` bit 2 `RTC_PWR_STS`)
   - `0x00000040` — General Reset Status (Same as `GEN_PMCON_3` bit 9 `GEN_RST_STS`)
   - `0xffffff80` — SUS Well Power Loss (Same as `GEN_PMCON_3` bit 14)
   - `0x00010000` — Wake cause was a ME Wake event (Same as PRSTS bit 0, `ME_WAKE_STS`)
   - `0x00020000` — Cold Reboot was ME Induced event (Same as `PRSTS` bit 1 `ME_HRST_COLD_STS`)
   - `0x00040000` — Warm Reboot was ME Induced event (Same as `PRSTS` bit 2 `ME_HRST_WARM_STS`)
   - `0x00080000` — Shutdown was ME Induced event (Same as `PRSTS` bit 3 `ME_HOST_PWRDN`)
   - `0x00100000` — Global reset ME 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: ~~Not installed~~`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: ~~Not installed~~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: ~~Not installed~~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: ~~Not installed~~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 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) — 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: ~~Not installed~~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: ~~Not installed~~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: ~~Not installed~~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: ~~Not installed~~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: ~~Not installed~~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: ~~Not installed~~Empty (Not installed)
   Description: Specifies the value of NVRAM variable `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:system-id` for boot services only. Value found on Macs is equal to SMBIOS `SystemUUID`.

10.6 SMBIOS Properties

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

2. `BIOSVersion`
   Type: `plist string`
   Failsafe: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~Empty (OEM specified)
   SMBIOS: System Information (Type 1) — Manufacturer
   Description: OEM manufacturer of the particular board. ~~Shall not be specified~~ Use failsafe unless strictly required. ~~Should *not*~~ 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, eficheck, 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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~Empty (OEM specified)
    SMBIOS: System Information (Type 1) — Family
    Description: Family name. May look like `iMac Pro`.

11. `BoardManufacturer`
    Type: `plist string`

Failsafe: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~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: ~~OEM specified~~Empty (OEM specified)
    SMBIOS: System Enclosure or Chassis (Type 3) — Version
    Description: Should match `BoardProduct`.

21. `ChassisSerialNumber`
    Type: `plist string`

Failsafe: ~~OEM specified~~Empty (OEM specified)
SMBIOS: System Enclosure or Chassis (Type 3) — Version
Description: Should match `SystemSerialNumber`.

22. `ChassisAssetTag`
Type: `plist string`
Failsafe: ~~OEM specified~~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.

- `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 string~~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`.

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

   Set to ~~empty string not to change console mode. Set to~~ `Max` to ~~try to use~~ attempt using the largest available console mode. ~~Currently~~ This option is currently ignored as the `Builtin` text renderer ~~supports only~~ only supports one console mode~~, so this option is ignored~~.

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

3. `Resolution`
   Type: `plist string`
   Failsafe: Empty ~~string~~(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 ~~empty string not to change screen resolution.~~
- ~~Set to~~ `Max` to ~~try to use~~ 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` ~~had to~~ 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: ~~Reinstalls~~ 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: ~~Reinstalls~~ 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: ~~Reinstalls~~ Replaces the Apple Debug Log protocol with a builtin version.
4. `AppleEvent`
   Type: `plist boolean`
   Failsafe: `false`
   Description: ~~Reinstalls~~ Replaces the Apple Event protocol with a builtin version. This may be used to ensure ~~File Vault~~ FileVault 2 compatibility on VMs or legacy Macs.
5. `AppleFramebufferInfo`
   Type: `plist boolean`
   Failsafe: `false`
   Description: ~~Reinstalls~~ 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: ~~Reinstalls~~ Replaces the Apple Image Conversion protocol with a builtin version.
7. `AppleImg4Verification`
   Type: `plist boolean`
   Failsafe: `false`
   Description: ~~Reinstalls~~ 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: ~~Reinstalls~~ Replaces Apple Key Map protocols with builtin versions.
9. `AppleRtcRam`
   Type: `plist boolean`
   Failsafe: `false`
   Description: ~~Reinstalls~~ 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: ~~Reinstalls~~ Replaces the Apple Secure Boot protocol with a builtin version.
11. `AppleSmcIo`
    Type: `plist boolean`
    Failsafe: `false`
    Description: ~~Reinstalls~~ 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: ~~Reinstalls~~ Replaces the Apple User Interface Theme protocol with a builtin version.
13. `DataHub`
    Type: `plist boolean`

Failsafe: `false`
Description: ~~Reinstalls~~ Replaces the Data Hub protocol with a builtin version. ~~This will delete all previous properties~~

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: ~~Reinstalls~~ Replaces the Device Property protocol with a builtin version. This ~~will delete all previous properties if it was already installed. 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: ~~Forcibly wraps~~ Wraps Firmware Volume protocols or installs ~~new~~ a new version to support custom cursor images for ~~File Vault~~ FileVault 2. ~~Should be set~~ Set to `true` to ensure ~~File Vault~~ FileVault 2 compatibility on ~~everything but~~ anything other than VMs and legacy Macs.

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

16. `HashServices`
Type: `plist boolean`
Failsafe: `false`
Description: ~~Forcibly reinstalls~~ Replaces Hash Services protocols with builtin versions. ~~Should be set~~ Set to `true` to ensure ~~File Vault~~ FileVault 2 compatibility on platforms ~~providing broken~~ with flawed SHA-1 ~~hashing. Can be diagnosed by~~ hash implementations. This can be determined by an invalid cursor size ~~with~~ when `UIScale` is set to `02`~~, in general platforms prior to~~ . Platforms earlier than APTIO V (Haswell and older) are typically affected.

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

18. `UnicodeCollation`
Type: `plist boolean`
Failsafe: `false`
Description: ~~Forcibly reinstalls~~ Replaces unicode collation services with builtin ~~version. Should be set~~ versions. Set to `true` to ensure UEFI Shell compatibility on platforms ~~providing broken unicode collation . In general legacy~~ with flawed unicode collation implementations. Legacy Insyde and APTIO platforms on Ivy Bridge~~and earlier are~~ , 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 ~~should be found in some future~~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 ~~only recommended when it 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 ~~that~~ 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 ~~anyhow corrupted~~ 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: ~~The reason this~~ This quirk cannot replace the kernel driver ~~is~~ because it cannot operate in ACPI S3 ~~mode~~ (sleep wake) mode and because the UEFI firmware only provides very limited multicore support ~~preventing the precise update~~ 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 ~~being unable~~ an inability to install File System protocols.

   Note: ~~The quirk is mostly relevant for select HP laptops with no drives listed~~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~~string~~
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. ~~It is implementation defined whether~~ 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~~, so to get a compatible distribution one may have to~~. For compatible distributions of such, download a device-specific image and ~~mod~~ modify it if necessary. ~~To get the~~ Visit this archived Apple Support article for a list of the bundled device-specific builds for legacy operating systems~~one can visit this archived Apple Support . Since it is not always~~. 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 RO
hdiutil mount Original.dmg -noverify -noautoopen -noautoopenrw -noautofsck -mountpoint RO
cp RO/.DS_Store DS_STORE
hdiutil detach RO -force
rm -rf RO
hdiutil convert Original.dmg -format UDRW -o ReadWrite.dmg
mkdir RW
```

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

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

8. **Why ~~do~~ must Find&Replace patches ~~must~~ be equal in ~~length~~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.