OpenCore

Reference Manual (~~0.9.9~~1.0.0)

[2024.05.03]

Contents

1 Introduction

This document provides information on the format of the OpenCore user configuration file used to set up the correct functioning of the macOS operating system. It is to be read as the official clarification of expected OpenCore behaviour. All deviations, if found in published OpenCore releases, shall be considered to be documentation or implementation issues which should be reported via the Acidanthera Bugtracker. An errata sheet is available in OpenCorePkg repository.

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

Third-party articles, utilities, books, and similar, may be more useful for a wider audience as they could provide guide-like material. However, they are subject to their authors' preferences, 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 implications.

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

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

1.1 Generic Terms

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

2 Configuration

2.1 Configuration Terms

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

2.2 Configuration Processing

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

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

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

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

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

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

4 ACPI

4.1 Introduction

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

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

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

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

Note: `RebaseRegions` and `SyncTableIds` quirks are special and are processed after all other ACPI changes since they can only be applied on the final ACPI configuration including all the patches and added tables.

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

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

There are several sources of ACPI tables and workarounds. Commonly used ACPI tables are provided with OpenCore, VirtualSMC, VoodooPS2, and WhateverGreen releases. Besides those, several third-party instructions may be found on the AppleLife Laboratory and DSDT subforums (e.g. Battery register splitting guide). A slightly more user-friendly explanation of some tables included with OpenCore can also be found in Dortania's Getting started with ACPI guide. For more exotic cases, there are several alternatives such as daliansky's ACPI sample collection (English Translation by 5T33Z0 et al). Please note however, that suggested solutions from third parties may be outdated or may contain errors.

4.2 Properties

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

   To be filled with `plist dict` values, describing each add entry. Refer to the Add Properties section below for details.

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

   To be filled with `plist dict` values, describing each delete entry. Refer to the Delete Properties section below for details.

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

   To be filled with `plist dictionary` values describing each patch entry. Refer to the Patch Properties section below for details.

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

4.3 Add Properties

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

2. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set to `true` to add this ACPI table.

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

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

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

4.4 Delete Properties

1. `All`
   Type: `plist boolean`
   Failsafe: `false` (Only delete the first matched table)
   Description: Set to `true` to delete all ACPI tables matching the condition.

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

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set to `true` to remove this ACPI table.

4. `OemTableId`
   Type: `plist data`, 8 bytes
   Failsafe: All zero (Match any table OEM ID)
   Description: Match table OEM ID equal to this value.

5. `TableLength`
   Type: `plist integer`
   Failsafe: `0` (Match any table size)
   Description: Match table size equal to this value.

6. `TableSignature`
   Type: `plist data`, 4 bytes
   Failsafe: All zero (Match any table signature)
   Description: Match table signature equal to this value.

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

To be filled with `plist dictionary` values, describing each patch. Refer to the Patch Properties section below for details.

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

5.3 MmioWhitelist Properties

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

   The addresses written here must be part of the memory map, have `EfiMemoryMappedIO` type and `EFI_MEMORY_RUNTIME` attribute (highest bit) set. The debug log can be used to find the list of the candidates.
2. `Comment`
   Type: `plist string`
   Failsafe: Empty
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. Whether this value is used is implementation defined.
3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Exclude MMIO address from the devirtualisation procedure.

5.4 Patch Properties

1. `Arch`
   Type: `plist string`
   Failsafe: `Any` (Apply to any supported architecture)
   Description: Booter patch architecture (`i386`, `x86_64`).
2. `Comment`
   Type: `plist string`
   Failsafe: Empty
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. Whether this value is used is implementation defined.
3. `Count`
   Type: `plist integer`
   Failsafe: `0` (Apply to all occurrences found)
   Description: Number of patch occurrences to apply.
4. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set to `true` to activate this booter patch.
5. `Find`
   Type: `plist data`
   Failsafe: Empty
   Description: Data to find. Must be equal to `Replace` in size if set.
6. `Identifier`
   Type: `plist string`
   Failsafe: `Any` (Match any booter)

Description: `Apple` for macOS booter (typically `boot.efi`); or a name with a suffix, such as `bootmgfw.efi`, for a specific booter.

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

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

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

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

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

5.5 Quirks Properties

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

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

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

   This quirk typically requires `ProvideCustomSlide` and `AvoidRuntimeDefrag` to be enabled ~~and typically also requires enabling `AvoidRuntimeDefrag`~~ to function correctly. Hibernation is not supported when booting with a relocation block, which will only be used if required when the quirk is enabled.

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

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

   This option fixes UEFI runtime services (date, time, NVRAM, power control, etc.) support on firmware that uses SMM backing for certain services such as variable storage. SMM may try to access memory by physical addresses in non-SMM areas but this may sometimes have been moved by boot.efi. This option prevents boot.efi from moving such data.

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

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

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

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

9. `FixupAppleEfiImages`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Fix errors in early Mac OS X boot.efi images.

   Modern secure PE loaders will refuse to load `boot.efi` images from ~~macOS~~ Mac OS X 10.4 to macOS 10.12 due to these files containing `W^X` errors (in all versions) and illegal overlapping sections (in 10.4 and 10.5 32-bit versions only).

   This quirk detects these issues and pre-processes such images in memory, so that a modern loader will accept them.

   Pre-processing in memory is incompatible with secure boot, as the image loaded is not the image on disk, so you cannot sign files which are loaded in this way based on their original disk image contents. Certain firmware will offer to register the hash of new, unknown images - this would still work. On the other hand, it is not particularly realistic to want to start these early, insecure images with secure boot anyway.

   Note 1: The quirk is never applied during the Apple secure boot path for newer macOS. The Apple secure boot path includes its own separate mitigations for `boot.efi` `W^X` issues.

   Note 2: When enabled, and when not processing for Apple secure boot, this quirk is applied to:

   - All images from Apple Fat binaries (32-bit and 64-bit versions in one image).
   - All Apple-signed images.
   - All images at `\System\Library\CoreServices\boot.efi` within their filesystem.

   Note 3: This quirk is needed for ~~macOS~~ Mac OS X 10.4 to macOS 10.12 (and higher, if Apple secure boot is not enabled), but only when the firmware itself includes a modern, more secure PE COFF image loader. This applies to current builds of OpenDuet, and to OVMF if built from audk source code.

10. `ForceBooterSignature`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Set macOS `boot-signature` to OpenCore launcher.

    Booter signature, essentially a SHA-1 hash of the loaded image, is used by Mac EFI to verify the authenticity of the bootloader when waking from hibernation. This option forces macOS to use OpenCore launcher SHA-1 hash as a booter signature to let OpenCore shim hibernation wake on Mac EFI firmware.

    Note: OpenCore launcher path is determined from `LauncherPath` property.

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

    Try to ensure that the `ExitBootServices` call succeeds. If required, an outdated `MemoryMap` key argument can be used by obtaining the current memory map and retrying the `ExitBootServices` call.

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

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

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

17. `RebuildAppleMemoryMap`
Type: `plist boolean`
Failsafe: `false`
Description: Generate macOS compatible Memory Map.

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

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

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

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

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

18. `ResizeAppleGpuBars`
Type: `plist integer`
Failsafe: `-1`
Description: Reduce GPU PCI BAR sizes for compatibility with macOS.

This quirk reduces GPU PCI BAR sizes for Apple macOS up to the specified value or lower if it is unsupported. The specified value follows PCI Resizable BAR spec. While Apple macOS supports a theoretical 1 GB maximum, in practice all non-default values may not work correctly. For this reason the only supported value for this quirk is the minimal supported BAR size, i.e. `0`. Use `-1` to disable this quirk.

For development purposes one may take risks and try other values. Consider a GPU with 2 BARs:

- `BAR0` supports sizes from 256 MB to 8 GB. Its value is 4 GB.
- `BAR1` supports sizes from 2 MB to 256 MB. Its value is 256 MB.

Example 1: Setting `ResizeAppleGpuBars` to 1 GB will change `BAR0` to 1 GB and leave `BAR1` unchanged.
Example 2: Setting `ResizeAppleGpuBars` to 1 MB will change `BAR0` to 256 MB and `BAR0` to 2 MB.
Example 3: Setting `ResizeAppleGpuBars` to 16 GB will make no changes.

Note: See `ResizeGpuBars` quirk for general GPU PCI BAR size configuration and more details about the technology.

19. `SetupVirtualMap`
Type: `plist boolean`
Failsafe: `false`
Description: Setup virtual memory at `SetVirtualAddresses`.

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

6 DeviceProperties

6.1 Introduction

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

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

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

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

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

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

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

For all kernel extensions that may inspect the `IODeviceTree` plane without probing, such as `Lilu` and its plugins (e.g. `WhateverGreen`), it is especially important to ensure device presence in the ACPI tables. A failure to do so may result **in 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 multidata` format.

   Note 1: Device paths must be provided in canonic string format (e.g. `PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x0)`).

   Note 2: Existing properties will not be changed unless deleted in the `DeviceProperties Delete` section.

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.

   Note: Currently, existing properties may only exist on firmware with DeviceProperties drivers (e.g. Apple). Hence, there is typically no reason to delete variables unless a new driver has been installed.

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.

7 Kernel

7.1 Introduction

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

Kernel and kext changes apply with the following effective order:

- `Block` is processed.
- `Add` and `Force` are processed.
- `Emulate` and `Quirks` are processed.
- `Patch` is processed.

7.2 Properties

1. `Add`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected kernel extensions (kexts) from the `OC/Kexts` directory.

   To be filled with `plist dict` values, describing each kext. Refer to the Add Properties section below for details.

   Note 1: The load order is based on the order in which the kexts appear in the array. Hence, dependencies must appear before kexts that depend on them.

   Note 2: To track the dependency order, inspect the `OSBundleLibraries` key in the `Info.plist` file of the kext being added. Any kext included under the key is a dependency that must appear before the kext being added.

   Note 3: Kexts may have inner kexts (`Plugins`) included in the bundle. Such `Plugins` must be added separately and follow the same global ordering rules as other kexts.

2. `Block`
   Type: `plist array`
   Failsafe: Empty
   Description: Remove selected kernel extensions (kexts) from the prelinked kernel.

   To be filled with `plist dictionary` values, describing each blocked kext. Refer to the Block Properties section below for details.

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

4. `Force`
   Type: `plist array`
   Failsafe: Empty
   Description: Load kernel extensions (kexts) from the system volume if they are not cached.

   To be filled with `plist dict` values, describing each kext. Refer to the Force Properties section below for details. This section resolves the problem of injecting kexts that depend on other kexts, which are not otherwise cached. The issue typically affects older operating systems, where various dependency kexts, such as `IOAudioFamily` or `IONetworkingFamily` may not be present in the kernel cache by default.

   Note 1: The load order is based on the order in which the kexts appear in the array. Hence, dependencies must appear before kexts that depend on them.

   Note 2: `Force` happens before `Add`.

   Note 3: The signature of the “forced” kext is not checked in any way. This makes using this feature extremely dangerous and undesirable for secure boot.

   Note 4: This feature may not work on encrypted partitions in newer operating systems.

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

   To be filled with `plist dictionary` values, describing each patch. Refer to the Patch Properties section below for details.

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

7. `Scheme`
   Type: `plist dict`
   Description: Define kernelspace operation mode via parameters described in the Scheme Properties section below.

7.3 Add Properties

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

2. `BundlePath`
   Type: `plist string`
   Failsafe: Empty
   Description: Kext bundle path (e.g. `Lilu.kext` or `MyKext.kext/Contents/PlugIns/MySubKext.kext`).

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

4. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set to `true` to add this kernel extension.

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

6. `MaxKernel`
   Type: `plist string`
   Failsafe: Empty
   Description: Adds kernel extension 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:

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

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

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

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

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
   Description: Adds kernel extension on specified macOS version or newer.

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

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

7.4 Block Properties

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

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

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set to `true` to block this kernel extension.

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

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

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

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

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

7. `Strategy`
   Type: `plist string`
   Failsafe: `Disable` (Forcibly make the kernel driver kmod startup code return failure)
   Description: Determines the behaviour of kernel driver blocking.

   Valid values:

   - `Disable` — Forcibly make the kernel driver kmod startup code return failure.
   - `Exclude` — Remove the kernel driver from the kernel cache by dropping plist entry and filling in zeroes.

   Note: It is risky to `Exclude` a kext that is a dependency of others.

   Note 2: At this moment `Exclude` is only applied to `prelinkedkernel` and newer mechanisms.

   Note 3: In most cases strategy `Exclude` requires the new kext to be injected as a replacement.

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 meets three requirements:

   - Enabling support for an unsupported CPU model (e.g. Intel Pentium).
   - Enabling support for a CPU model not yet supported by a specific version of macOS (typically old versions).
   - 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: Only the value of `EAX`, which represents the full CPUID, typically needs to be accounted for and remaining bytes should be left as zeroes. The byte order is Little Endian. For example, `C3 06 03 00` stands for CPUID `0x0306C3` (Haswell).

   Note 3: For XCPM support it is recommended to use the following combinations. Be warned that one is required to set the correct frequency vectors matching the installed CPU.

   - 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`
   - Comet Lake U62 (`0x0A0660`) to Comet Lake U42 (`0x0806EC`):
     `Cpuid1Data`: `EC 06 08 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`
   - Rocket Lake (`0x0A0670`) to Comet Lake (`0x0A0655`):
     `Cpuid1Data`: `55 06 0A 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`
   - Alder Lake (`0x090672`) to Comet Lake (`0x0A0655`):
     `Cpuid1Data`: `55 06 0A 00 00 00 00 00 00 00 00 00 00 00 00 00`
     `Cpuid1Mask`: `FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00`

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

   - Consumer Ivy Bridge (`0x0306A9`) as Apple disabled XCPM for Ivy Bridge and recommends legacy power management for these CPUs. `_xcpm_bootstrap` should manually be patched to enforce XCPM on these CPUs instead of this option.
   - Low-end CPUs (e.g. Haswell+ Pentium) as they are not supported properly by macOS. Legacy workarounds for older models can be found in the `Special NOTES` section of acidanthera/bugtracker#365.

2. `Cpuid1Mask`
   Type: `plist data`, 16 bytes
   Failsafe: All zero
   Description: Bit mask of active bits in `Cpuid1Data`.

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

3. `DummyPowerManagement`
   Type: `plist boolean`
   Failsafe: `false`
   Requirement: 10.4-12
   Description: Disables `AppleIntelCpuPowerManagement`.

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

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

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

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

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

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

7.6 Force Properties

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

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

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

4. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set to `true` to load this kernel extension from the system volume when not present in the kernel cache.

5. `ExecutablePath`
   Type: `plist string`
   Failsafe: Empty
   Description: Kext executable path relative to bundle (e.g. `Contents/MacOS/IONetworkingFamily`).

6. `Identifier`
   Type: `plist string`

Failsafe: Empty
Description: Kext identifier to perform presence checking before adding (e.g. `com.apple.iokit.IONetworkingFamily`). Only drivers which identifiers are not be found in the cache will be added.

7. `MaxKernel`
   Type: `plist string`
   Failsafe: Empty
   Description: Adds kernel extension on specified macOS version or older.

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

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

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

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

7.7 Patch Properties

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

2. `Base`
   Type: `plist string`
   Failsafe: Empty (Ignored)
   Description: Selects symbol-matched base for patch lookup (or immediate replacement) by obtaining the address of the provided symbol name.

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

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

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

6. `Find`
   Type: `plist data`
   Failsafe: Empty (Immediate replacement at `Base`)
   Description: Data to find. Must be equal to `Replace` in size if set.

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

8. `Limit`
   Type: `plist integer`

trim the same lower blocks that have previously been deallocated, but never have enough time to deallocate higher blocks. The outcome is that trimming on such SSDs will be non-functional soon after installation, resulting in additional wear on the flash.

One way to workaround the problem is to increase the timeout to an extremely high value, which at the cost of slow boot times (extra minutes) will ensure that all the blocks are trimmed. Setting this option to a high value, such as `4294967295` ensures that all blocks are trimmed. Alternatively, use over-provisioning, if supported, or create a dedicated unmapped partition where the reserve blocks can be found by the controller. Conversely, the trim operation can be mostly disabled by setting a very low timeout value, while `0` entirely disables it. Refer to this article for details.

Note: The failsafe value `-1` indicates that this patch will not be applied, such that `apfs.kext` will remain untouched.

Note 2: On macOS 12.0 and above, it is no longer possible to specify trim timeout. However, trim can be disabled by setting `0`.

Note 3: Trim operations are *only* affected at booting phase when the startup volume is mounted. Either specifying timeout, or completely disabling trim with `0`, will not affect normal macOS running.

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

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

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

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

7.9 Scheme Properties

These properties are particularly relevant for older macOS operating systems. Refer to the Legacy Apple OS section for details on how to install and troubleshoot such macOS installations.

1. `CustomKernel`
Type: `plist boolean`
Failsafe: `false`
Description: Use customised kernel cache from the `Kernels` directory located at the root of the ESP partition.

    Unsupported platforms including `Atom` and `AMD` require modified versions of XNU kernel in order to boot. This option provides the possibility to using a customised kernel cache which contains such modifications from ESP partition.

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

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

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

3. `KernelArch`
   Type: `plist string`
   Failsafe: `Auto` (Choose the preferred architecture automatically)
   Description: Prefer specified kernel architecture (`i386`, `i386-user32`, `x86_64`) when available.

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

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

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

   (a) `arch` argument in image arguments (e.g. when launched via UEFI Shell) or in `boot-args` variable overrides any compatibility checks and forces the specified architecture, completing this algorithm.
   (b) OpenCore build architecture restricts capabilities to `i386` and `i386-user32` mode for the 32-bit firmware variant.
   (c) Determined EfiBoot version restricts architecture choice:
   - 10.4-10.5 — `i386` or `i386-user32` (only on 32-bit firmware)
   - 10.6 — `i386`, `i386-user32`, or `x86_64`
   - 10.7 — `i386` or `x86_64`
   - 10.8 or newer — `x86_64`

   (d) If `KernelArch` is set to `Auto` and `SSSE3` is not supported by the CPU, capabilities are restricted to `i386-user32` if supported by EfiBoot.
   (e) Board identifier (from SMBIOS) based on EfiBoot version disables `x86_64` support on an unsupported model if any `i386` variant is supported. `Auto` is not consulted here as the list is not overridable in EfiBoot.
   (f) `KernelArch` restricts the support to the explicitly specified architecture (when not set to `Auto`) if the architecture remains present in the capabilities.
   (g) The best supported architecture is chosen in this order: `x86_64`, `i386`, `i386-user32`.

   Unlike ~~macOS~~Mac OS X 10.7 (where certain board identifiers are treated as `i386` only machines), and ~~macOS~~Mac OS X 10.5 or earlier (where `x86_64` is not supported by the macOS kernel), ~~macOS~~Mac OS X 10.6 is very special. The architecture choice on ~~macOS~~Mac OS X 10.6 depends on many factors including not only the board identifier, but also the macOS product type (client vs server), macOS point release, and amount of RAM. The detection of all these is complicated and impractical, as several point releases had implementation flaws resulting in a failure to properly execute the server detection in the first place. For this reason when `Auto` is set, OpenCore on ~~macOS~~Mac OS X 10.6 falls back to the `x86_64` architecture when it is supported by the board, as on ~~macOS~~Mac OS X 10.7. The 32-bit `KernelArch` options can still be configured explicitly however.

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

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

A `.contentVisibility` file may be placed next to the bootloader (such as `boot.efi`), or in the boot folder (for DMG folder based boot items). Example locations, as seen from within macOS, are:

- `/System/Volumes/Preboot/{GUID}/System/Library/CoreServices/.contentVisibility`
- `/Volumes/{ESP}/EFI/BOOT/.contentVisibility`

In addition a `.contentVisibility` file may be placed in the instance-specific (for macOS) or absolute root folders related to a boot entry, for example:

- `/System/Volumes/Preboot/{GUID}/.contentVisibility`
- `/System/Volumes/Preboot/.contentVisibility`
- `/Volumes/{ESP}/.contentVisibility` (*not recommended*)

These root folder locations are supported specifically for macOS, because non-Apple files next to the Apple bootloader are removed by macOS updates. It is supported but not recommended to place a `.contentVisibility` file in a non-macOS root location (such as the last location shown above), because it will hide all entries on the drive.

The `.contentVisibility` file, when present, may optionally target only specific instances of OpenCore. Its contents are `[{Instance-List}:](Disabled|Auxiliary)`. If a colon (`:`) is present, the preceding `Instance-List` it is a comma separated list of `InstanceIdentifier` values (example: `OCA,OCB:Disabled`). When this list is present, the specified visibility is only applied if the `InstanceIdentifier` of the current instance of OpenCore is present in the list. When the list is not present, the specified visibility is applied for all instances of OpenCore.

Note 1: For any instance of OpenCore with no `InstanceIdentifier` value, the specified visibility from a `.contentVisibility` file with an `Instance-List` will never be applied.

Note 2: Visibilities with a visibility list will be treated as invalid, and so ignored, in earlier versions of OpenCore - which may be useful when comparing behaviour of older and newer versions.

Note 3: Avoid extraneous spaces in the `.contentVisibility` file: these will not be treated as whitespace, but as part of the adjacent token.

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

The display order is as follows:

- Alternate options follow corresponding primary options. That is, Apple recovery options will follow the relevant macOS option whenever possible.
- Options will be listed in file system handle firmware order to maintain an established order across reboots regardless of the operating system chosen for loading.
- Custom entries, tools, and system entries will be added after all other options.
- Auxiliary options will only be displayed upon entering "Extended Mode" in the OpenCore picker (typically by pressing the `Space` key).

The boot process is as follows:

- Look up the first valid primary option in the `BootNext` UEFI variable.
- On failure, look up the first valid primary option in the `BootOrder` UEFI variable.
- Mark the option as the default option to boot.
- Boot option through the picker or without it depending on the `ShowPicker` option.
- Show picker on failure otherwise.

Note 1: This process will only work reliably when the `RequestBootVarRouting` option is enabled or the firmware does not control UEFI boot options (`OpenDuetPkg` or custom BDS). When `LauncherOption` is not enabled, other operating systems may overwrite OpenCore settings and this property should therefore be enabled when planning to use other operating systems.

Note 2: UEFI variable boot options boot arguments will be removed, if present, as they may contain arguments that can compromise the operating system, which is undesirable when secure boot is enabled.

Note 3: Some operating systems, such as Windows, may create a boot option and mark it as the topmost option upon first boot or after NVRAM resets from within OpenCore. When this happens, the default boot entry choice will remain changed until the next manual reconfiguration.

8.2 Properties

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

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

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

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

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

5. `Entries`
   Type: `plist array`
   Failsafe: Empty
   Description: Add boot entries to OpenCore picker.

   To be filled with `plist dict` values, describing each load entry. Refer to the Entry Properties section below for details.

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

7. `Serial`
   Type: `plist dict`
   Description: Perform serial port initialisation and configure PCD values required by `BaseSerialPortLib16550` for serial ports to properly function. Values are listed and described in the Serial Properties and Serial Custom Properties section below.

   By enabling `Init`, this section ensures that the serial port is initialised when it is not done by firmware. In order for OpenCore to print logs to the serial port, bit `3` (i.e. serial logging) for `Target` under section `Misc->Debug` must be set.

   When debugging with serial ports, `BaseSerialPortLib16550` only recognises internal ones provided by the motherboard by default. If the option `Override` is enabled, this section will override the PCD values listed in BaseSerialPortLib16550.inf such that external serial ports (e.g. from a PCI card) will also function properly. Specifically, when troubleshooting macOS, in addition to overriding these PCD values, it is also necessary to turn the `CustomPciSerialDevice` kernel quirk on in order for the XNU to use such exterior serial ports.

   Refer to MdeModulePkg.dec for the explanations of each key.

8. `Tools`
   Type: `plist array`
   Failsafe: Empty
   Description: Add tool entries to the OpenCore picker.

   To be filled with `plist dict` values, describing each load entry. Refer to the Entry Properties section below for details.

   Note: Certain UEFI tools, such as UEFI Shell, can be very dangerous and **MUST NOT** appear in production configurations, paticularly in vaulted configurations as well as those protected by secure boot, as such tools can be used to bypass the secure boot chain. Refer to the UEFI section for examples of UEFI tools.

Failsafe: `false`
Description: Enable screen reader by default in the OpenCore picker.

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

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

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

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

    ~~On certain firmware, using modifier keys may be problematic due to driver incompatibilities. To workaround this problem, this option allows registering certain hotkeys in a more permissive manner from within the OpenCore picker. Such extensions include support for tapping on key combinations before selecting the boot item, and for reliable detection of the `Shift` key when selecting the boot item, in order to work around the fact that hotkeys which are continuously held during boot cannot be reliably detected on many PS/2 keyboards.~~

    ~~This list of known `modifier hotkeys` includes:~~

    - ~~`CMD+C+MINUS` — disable board compatibility checking.~~
    - ~~`CMD+K` — boot release kernel, similar to `kcsuffix=release`.~~
    - ~~`CMD+S` — single user mode.~~
    - ~~`CMD+S+MINUS` — disable KASLR slide, requires disabled SIP.~~
    - ~~`CMD+V` — verbose mode.~~
    - ~~`Shift+Enter`, `Shift+Index` — safe mode, may be used in combination with `CTRL+Enter`, `CTRL+Index`.~~

11. ~~`ShowPicker`~~
    Type: ~~`plist boolean`**Failsafe**: `false`**Description**: Show a simple picker to allow boot entry selection.~~

12. ~~`TakeoffDelay`**Type**: `plist integer, 32 bit`**Failsafe**: `0`**Description**: Delay in microseconds executed before handling the OpenCore picker startup and `action hotkeys`.~~

    ~~Introducing a delay may give extra time to hold the right `action hotkey` sequence to, for instance, boot into recovery mode. On most systems, the appearance of the initial boot logo is a good indication of the time from which hotkeys can be held down. Earlier than this, the key press may not be registered. On some platforms, setting this option to a minimum of `5000-10000` microseconds is also required to access `action hotkeys` due to the nature of the keyboard driver.~~

    ~~If the boot chime is configured (see audio configuration options) then at the expense of slower startup, an even longer delay of half to one second (`500000-1000000`) may be used to create behaviour similar to a real Mac, where the chime itself can be used as a signal for when hotkeys can be pressed. The boot chime is inevitably later in the boot sequence in OpenCore than on Apple hardware, due to the fact that non-native drivers have to be loaded and connected first. Configuring the boot chime and adding this longer additional delay can also be useful in systems where fast boot time and/or slow monitor signal synchronisation may cause the boot logo not to be shown at all on some boots or reboots.~~

13. ~~`Timeout`**Type**: `plist integer, 32 bit`**Failsafe**: `0`**Description**: Timeout in seconds in the OpenCore picker before automatic booting of the default boot entry. Set to `0` to disable.~~

14. ~~`PickerMode`**Type**:~~ `plist string`
    Failsafe: `Builtin`
    Description: Choose picker used for boot management.

    `PickerMode` describes the underlying boot management with an optional user interface responsible for handling boot options.

- `Acidanthera\GoldenGate` — macOS 11 styled icon set.
- `Acidanthera\Syrah` — ~~macOS~~ OS X 10.10 styled icon set.
- `Acidanthera\Chardonnay` — ~~macOS~~ Mac OS X 10.4 styled icon set.

For convenience purposes there also are predefined aliases:

- `Auto` — Automatically select one set of icons based on the `DefaultBackground` colour: `Acidanthera\GoldenGate` for Syrah Black and `Acidanthera\Chardonnay` for Light Gray.
- `Default` — `Acidanthera\GoldenGate`.

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

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

    On certain firmware, using modifier keys may be problematic due to driver incompatibilities. To workaround this problem, this option allows registering certain hotkeys in a more permissive manner from within the OpenCore picker. Such extensions include support for tapping on key combinations before selecting the boot item, and for reliable detection of the `Shift` key when selecting the boot item, in order to work around the fact that hotkeys which are continuously held during boot cannot be reliably detected on many PS/2 keyboards.

    This list of known `modifier hotkeys` includes:

    - `CMD+C+MINUS` — disable board compatibility checking.
    - `CMD+K` — boot release kernel, similar to `kcsuffix=release`.
    - `CMD+S` — single user mode.
    - `CMD+S+MINUS` — disable KASLR slide, requires disabled SIP.
    - `CMD+V` — verbose mode.
    - `Shift+Enter`, `Shift+Index` — safe mode, may be used in combination with `CTRL+Enter`, `CTRL+Index`.

17. `ShowPicker`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Show a simple picker to allow boot entry selection.

18. `TakeoffDelay`
    Type: `plist integer`, 32 bit
    Failsafe: `0`
    Description: Delay in microseconds executed before handling the OpenCore picker startup and `action hotkeys`.

    Introducing a delay may give extra time to hold the right `action hotkey` sequence to, for instance, boot into recovery mode. On most systems, the appearance of the initial boot logo is a good indication of the time from which hotkeys can be held down. Earlier than this, the key press may not be registered. On some platforms, setting this option to a minimum of `5000-10000` microseconds is also required to access `action hotkeys` due to the nature of the keyboard driver.

    If the boot chime is configured (see audio configuration options) then at the expense of slower startup, an even longer delay of half to one second (`500000-1000000`) may be used to create behaviour similar to a real Mac, where the chime itself can be used as a signal for when hotkeys can be pressed. The boot chime is inevitably later in the boot sequence in OpenCore than on Apple hardware, due to the fact that non-native drivers have to be loaded and connected first. Configuring the boot chime and adding this longer additional delay can also be useful in systems where fast boot time and/or slow monitor signal synchronisation may cause the boot logo not to be shown at all on some boots or reboots.

19. `Timeout`
    Type: `plist integer`, 32 bit
    Failsafe: `0`

- `OCDM` — OcDeviceMiscLib
- `OCFS` — OcFileLib
- `OCFV` — OcFirmwareVolumeLib
- `OCHS` — OcHashServicesLib
- `OCI4` — OcAppleImg4Lib
- `OCIC` — OcImageConversionLib
- `OCII` — OcInputLib
- `OCJS` — OcApfsLib
- `OCKM` — OcAppleKeyMapLib
- `OCL` — OcLogAggregatorLib
- `OCM` — OcMiscLib
- `OCMCO` — OcMachoLib
- `OCME` — OcHeciLib
- `OCMM` — OcMemoryLib
- `OCPE` — OcPeCoffLib, OcPeCoffExtLib
- `OCPI` — OcFileLib, partition info
- `OCPNG` — OcPngLib
- `OCRAM` — OcAppleRamDiskLib
- `OCRTC` — OcRtcLib
- `OCSB` — OcAppleSecureBootLib
- `OCSMB` — OcSmbiosLib
- `OCSMC` — OcSmcLib
- `OCST` — OcStorageLib
- `OCS` — OcSerializedLib
- `OCTPL` — OcTemplateLib
- `OCUC` — OcUnicodeCollationLib
- `OCUT` — OcAppleUserInterfaceThemeLib
- `OCVAR` — OcVariableLib
- `OCXML` — OcXmlLib

8.5 Entry Properties

1. `Arguments`
   Type: `plist string`
   Failsafe: Empty
   Description: Arbitrary ASCII string used as boot arguments (load options) of the specified entry.

2. `Auxiliary`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set to `true` to hide this entry when `HideAuxiliary` is also set to `true`. Press the `Spacebar` key to enter "Extended Mode" and display the entry when hidden.

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

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

5. `Flavour`
   Type: `plist string`
   Failsafe: `Auto`
   Description: Specify the content flavour for this entry. See `OC_ATTR_USE_FLAVOUR_ICON` flag for documentation.

6. `FullNvramAccess`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disable `OpenRuntime` NVRAM protection during usage of a tool.

   This disables all of the NVRAM protections provided by `OpenRuntime.efi`, during the time a tool is in use. It should normally be avoided, but may be required for instance if a tool needs to access NVRAM directly without the redirections put in place by `RequestBootVarRouting`.

   Note: This option is only valid for `Tools` and cannot be specified for `Entries` (is always `false`).

7. `Name`
   Type: `plist string`
   Failsafe: Empty
   Description: Human readable entry name displayed in the OpenCore picker.

8. `Path`
   Type: `plist string`
   Failsafe: Empty
   Description: Entry location depending on entry type.

   - `Entries` specify external boot options, and therefore take device paths in the `Path` key. Care should be exercised as these values are not checked. Example: `PciRoot(0x0)/Pci(0x1,0x1)/.../\EFI\COOL.EFI`
   - `Tools` specify internal boot options, which are part of the bootloader vault, and therefore take file paths relative to the `OC/Tools` directory. Example: `OpenShell.efi`.

9. `RealPath`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Pass full path to the tool when launching.

   This should typically be disabled as passing the tool directory may be unsafe with tools that accidentally attempt to access files without checking their integrity. Reasons to enable this property may include cases where tools cannot work without external files or may need them for enhanced functionality such as `memtest86` (for logging and configuration), or `Shell` (for automatic script execution).

   Note: This option is only valid for `Tools` and cannot be specified for `Entries` (is always `true`).

10. `TextMode`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Run the entry in text mode instead of graphics mode.

    This setting may be beneficial for some older tools that require text output as all the tools are launched in graphics mode by default. Refer to the Output Properties section below for information on text modes.

8.6 Security Properties

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

   Note 1: May be used in combination with `Shift+Enter` or `Shift+Index` when `PollAppleHotKeys` is enabled.

   Note 2: In order to support systems with unresponsive modifiers during preboot (which includes `V1` and `V2` `KeySupport` mode on some firmware) OpenCore also allows holding the `=/+` key in order to trigger 'set default' mode.

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

Note 2: A halt will only occur after the configuration is loaded and logging is configured. If any log messages occur at the specified halt level in early log (i.e. before this), they will cause a halt when they are flushed to the log once it has been configured.

9. `PasswordHash`
Type: `plist data` 64 bytes
Failsafe: all zero
Description: Password hash used when `EnablePassword` is set.

10. `PasswordSalt`
Type: `plist data`
Failsafe: empty
Description: Password salt used when `EnablePassword` is set.

11. ~~`Vault`**Type**: `plist string`**Failsafe**: `Secure`**Description**: Enables the OpenCore vaulting mechanism.~~

    ~~Valid values:~~

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

    ~~The `vault.plist` file should contain SHA-256 hashes for all files used by OpenCore. The presence of this file is highly recommended to ensure that unintentional file modifications (including filesystem corruption) do not go unnoticed. To create this file automatically, use the `create_vault.sh` script. Notwithstanding the underlying file system, the path names and cases between `config.plist` and `vault.plist` must match.~~

    ~~The `vault.sig` file should contain a raw 256 byte RSA-2048 signature from a SHA-256 hash of `vault.plist`. The signature is verified against the public key embedded into `OpenCore.efi`.~~

    ~~To embed the public key, either one of the following should be performed:~~

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

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

    ~~The complete set of commands to:~~

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

    ~~Can look as follows:~~

    ~~*Note 1*: While it may appear obvious, an external method is required to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this, it is recommended to enable UEFI SecureBoot using a custom certificate and to sign `OpenCore.efi` and `BOOTx64.efi` with a custom key. More details on customising secure boot on modern firmware can be found in the Taming UEFI SecureBoot paper (in Russian).~~

    ~~*Note 2*: Regardless of this option, `vault.plist` is always used when present, and both `vault.plist` and `vault.sig` are used and required when a public key is embedded into `OpenCore.efi`, and errors will abort the boot process in either case. Setting this option allows OpenCore to warn the user if the configuration is not as required to achieve an expected higher security level.~~

12. `ScanPolicy`
Type: `plist integer`, 32 bit

partition, resulting in boot failures. This is likely to be the case when an "OCB: Apple Secure Boot prohibits this boot entry, enforcing!" message is logged.

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 the UEFI Secure Boot section.

14. `Vault`
Type: `plist string`
Failsafe: `Secure`
Description: Enables the OpenCore vaulting mechanism.

Valid values:

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

The `vault.plist` file should contain SHA-256 hashes for all files used by OpenCore. The presence of this file is highly recommended to ensure that unintentional file modifications (including filesystem corruption) do not go unnoticed. To create this file automatically, use the `create_vault.sh` script. Notwithstanding the underlying file system, the path names and cases between `config.plist` and `vault.plist` must match.

The `vault.sig` file should contain a raw 256 byte RSA-2048 signature from a SHA-256 hash of `vault.plist`. The signature is verified against the public key embedded into `OpenCore.efi`.

To embed the public key, either one of the following should be performed:

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

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

The complete set of commands to:

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

Can look as follows:

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

Note 1: While it may appear obvious, an external method is required to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this, it is recommended to enable UEFI SecureBoot using a custom certificate and to sign `OpenCore.efi` and `BOOTx64.efi` with a custom key. More details on customising secure boot on modern firmware can be found in the Taming UEFI SecureBoot paper (in Russian).

Note 2: Regardless of this option, `vault.plist` is always used when present, and both `vault.plist` and `vault.sig` are used and required when a public key is embedded into `OpenCore.efi`, and errors will abort the boot process in either case. Setting this option allows OpenCore to warn the user if the configuration is not as required to achieve an expected higher security level.

8.7 Serial Properties

1. `Custom`
   Type: `plist dict`
   Description: Update serial port properties in `BaseSerialPortLib16550`.

   This section lists the PCD values that are used by the `BaseSerialPortLib16550`. When option `Override` is set to `false`, this dictionary is optional.

2. `Init`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform serial port initialisation.

   This option will perform serial port initialisation within OpenCore prior to enabling (any) debug logging.

   Refer to the `Debugging` section for details.

3. `Override`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Override serial port properties. When this option is set to `false`, no keys from `Custom` will be overridden.

   This option will override serial port properties listed in the `Serial Custom Properties` section below.

8.7.1 Serial Custom Properties

1. `BaudRate`
   Type: `plist integer`
   Failsafe: `115200`
   Description: Set the baud rate for serial port.

   This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialBaudRate` defined in MdeModulePkg.dec.

2. `ClockRate`
   Type: `plist integer`
   Failsafe: `1843200`
   Description: Set the clock rate for serial port.

   This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialClockRate` defined in MdeModulePkg.dec.

3. `DetectCable`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable serial port cable detection.

   This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialDetectCable` defined in MdeModulePkg.dec.

4. `ExtendedTxFifoSize`
   Type: `plist integer`
   Failsafe: `64`
   Description: Set the extended transmit FIFO size for serial port.

   This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialExtendedTxFifoSize` defined in MdeModulePkg.dec.

5. `FifoControl`
   Type: `plist integer`

Failsafe: `0x07`
Description: Configure serial port FIFO Control settings.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialFifoControl` defined in MdeModulePkg.dec.

6. `LineControl`
Type: `plist integer`
Failsafe: `0x07`
Description: Configure serial port Line Control settings.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialLineControl` defined in MdeModulePkg.dec.

7. `PciDeviceInfo`
Type: `plist data`
Failsafe: `0xFF`
Description: Set PCI serial device information.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialPciDeviceInfo` defined in MdeModulePkg.dec.

Note: The maximum allowed size of this option is 41 bytes. Refer to acidanthera/bugtracker#1954 for more details.

Note 2: This option can be set by running the `FindSerialPort` tool.

8. `RegisterAccessWidth`
Type: `plist integer`
Failsafe: `8`
Description: Set serial port register access width.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialRegisterAccessWidth` defined in MdeModulePkg.dec.

9. `RegisterBase`
Type: `plist integer`
Failsafe: `0x03F8`
Description: Set the base address of serial port registers.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialRegisterBase` defined in MdeModulePkg.dec.

10. `RegisterStride`
Type: `plist integer`
Failsafe: `1`
Description: Set the serial port register stride in bytes.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialRegisterStride` defined in MdeModulePkg.dec.

11. `UseHardwareFlowControl`
Type: `plist boolean`
Failsafe: `false`
Description: Enable serial port hardware flow control.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialUseHardwareFlowControl` defined in MdeModulePkg.dec.

12. `UseMmio`
Type: `plist boolean`
Failsafe: `false`
Description: Indicate whether the serial port registers are in MMIO space.

This option will override the value of `gEfiMdeModulePkgTokenSpaceGuid.PcdSerialUseMmio` defined in MdeModulePkg.dec.

8.8 ~~Entry Properties~~

1. ~~`Arguments`**Type**: `plist string`**Failsafe**: Empty**Description**: Arbitrary ASCII string used as boot arguments (load options) of the specified entry.~~

2. ~~`Auxiliary`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Set to `true` to hide this entry when `HideAuxiliary` is also set to `true`. Press the `Spacebar` key to enter "Extended Mode" and display the entry when hidden.~~

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

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

5. ~~`Flavour`**Type**: `plist string`**Failsafe**: `Auto`**Description**: Specify the content flavour for this entry. See flag for documentation.~~

6. ~~`FullNvramAccess`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Disable `OpenRuntime` NVRAM protection during usage of a tool.~~

   ~~This disables all of the NVRAM protections provided by `OpenRuntime.efi`, during the time a tool is in use. It should normally be avoided, but may be required for instance if a tool needs to access NVRAM directly without the redirections put in place by `RequestBootVarRouting`.~~

   ~~*Note*: This option is only valid for `Tools` and cannot be specified for `Entries` (is always `false`).~~

7. ~~`Name`**Type**: `plist string`**Failsafe**: Empty**Description**: Human readable entry name displayed in the OpenCore picker.~~

8. ~~`Path`**Type**: `plist string`**Failsafe**: Empty**Description**: Entry location depending on entry type.~~

9. ~~`Entries` specify external boot options, and therefore take device paths in the `Path` key. Care should be exercised as these values are not checked. Example: `PciRoot(0x0)/Pci(0x1,0x1)/.../\EFI\COOL.EFI`~~

10. ~~`Tools` specify internal boot options, which are part of the bootloader vault, and therefore take file paths relative to the `OC/Tools` directory. Example: `OpenShell.efi`.~~

11. ~~`RealPath`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Pass full path to the tool when launching.~~

    ~~This should typically be disabled as passing the tool directory may be unsafe with tools that accidentally attempt to access files without checking their integrity. Reasons to enable this property may include cases where tools cannot work without external files or may need them for enhanced functionality such as `memtest86` (for logging and configuration), or `Shell` (for automatic script execution).~~

    ~~*Note*: This option is only valid for `Tools` and cannot be specified for `Entries` (is always `true`).~~

12. ~~`TextMode`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Run the entry in text mode instead of graphics mode.~~

    ~~This setting may be beneficial for some older tools that require text output as all the tools are launched in graphics mode by default. Refer to the section below for information on text modes.~~

9 NVRAM

9.1 Introduction

This section allows setting non-volatile UEFI variables commonly described as NVRAM variables. Refer to `man nvram` for details. The macOS operating system extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication. Hence, the supply of several NVRAM variables is required for the proper functioning of macOS.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' the NVRAM variable belongs to. The macOS operating system makes use of several GUIDs, including but not limited to:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14` (`APPLE_VENDOR_VARIABLE_GUID`)
- `7C436110-AB2A-4BBB-A880-FE41995C9F82` (`APPLE_BOOT_VARIABLE_GUID`)
- `5EDDA193-A070-416A-85EB-2A1181F45B18` (Apple Hardware Configuration Storage for `MacPro7,1`)
- `8BE4DF61-93CA-11D2-AA0D-00E098032B8C` (`EFI_GLOBAL_VARIABLE_GUID`)
- `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` (`OC_VENDOR_VARIABLE_GUID`)

Note: Some of the variables may be added by the PlatformNVRAM or Generic subsections of the PlatformInfo section. Please ensure that variables set in this section do not conflict with items in those subsections as the implementation behaviour is undefined otherwise.

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

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

9.2 Properties

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

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

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

2. `Delete`
   Type: `plist dict`
   Description: Removes NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

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

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

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

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

10 PlatformInfo

Platform information consists of several identification fields generated or filled manually to be compatible with macOS services. The base part of the configuration may be obtained from `AppleModels`, which itself generates a set of interfaces based on a database in YAML format. These fields are written to three destinations:

- SMBIOS
- Data Hub
- NVRAM

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

The dmidecode utility can be used to inspect SMBIOS contents and a version with macOS specific enhancements can be downloaded from Acidanthera/dmidecode.

10.1 Properties

1. `Automatic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Generate PlatformInfo based on the `Generic` section instead of using values from the `DataHub`, `NVRAM`, and `SMBIOS` sections.

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

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

   Warning: Setting this option to `false` is strongly discouraged when intending to update platform information. A `false` setting is typically only valid for minor corrections to SMBIOS values on legacy Apple hardware. In all other cases, setting `Automatic` to `false` may lead to hard-to-debug errors resulting from inconsistent or invalid settings.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

11. `UpdateSMBIOSMode`
    Type: `plist string`
    Failsafe: `Create`
    Description: Update SMBIOS fields approach:

    - `TryOverwrite` — `Overwrite` if new size is <= than the page-aligned original and there are no issues with legacy region unlock. `Create` otherwise. Has issues on some types of firmware.
    - `Create` — Replace the tables with newly allocated EfiReservedMemoryType at AllocateMaxAddress without any fallbacks.
    - `Overwrite` — Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
    - `Custom` — Write SMBIOS tables (`gEfiSmbios(3)TableGuid`) to `gOcCustomSmbios(3)TableGuid` to workaround firmware overwriting SMBIOS contents at ExitBootServices. Otherwise equivalent to `Create`. Requires patching AppleSmbios.kext and AppleACPIPlatform.kext to read from another GUID: `"EB9D2D31"` - `"EB9D2D35"` (in ASCII), done automatically by `CustomSMBIOSGuid` quirk.

    Note: A side effect of using the `Custom` approach that it makes SMBIOS updates exclusive to macOS, avoiding a collision with existing Windows activation and custom OEM software but potentially obstructing the operation of Apple-specific tools.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

10.2 ~~Generic~~ DataHub Properties

1. ~~`SpoofVendor`~~`ARTFrequency`
   Type: `plist` ~~`boolean`~~`integer`, 64-bit
   Failsafe: ~~`false`~~`0` (Automatic)
   Description: Sets ~~SMBIOS vendor fields to `Acidanthera`.~~

   ~~It can be dangerous to use "Apple" in SMBIOS vendor fields for reasons outlined in the~~ `ARTFrequency` in ~~`SystemManufacturer`description. However, certain firmware may not provide valid values otherwise, which could obstruct the operation of some software.~~ `gEfiProcessorSubClassGuid`.

2. ~~`AdviseFeatures`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Updates `FirmwareFeatures` with supported bits.~~ This value contains CPU ART frequency, also known as crystal clock frequency. Its existence is exclusive to the Skylake generation and newer. The value is specified in Hz, and is normally 24 MHz for the client Intel segment, 25 MHz for the server Intel segment, and 19.2 MHz for Intel Atom CPUs. macOS till 10.15 inclusive assumes 24 MHz by default.

   ~~Added bits to `FirmwareFeatures`:~~

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

- ~~FW_FEATURE_SUPPORTS_APFS (0x00080000) - Without this bit, it is not possible to install macOS on an APFS disk.~~
- ~~FW_FEATURE_SUPPORTS_LARGE_BASESYSTEM (0x800000000) - Without this bit, it is not possible to install macOS versions with large BaseSystem images, such as macOS 12.~~

Note: On ~~most newer firmwares these bits are already set, the option may be necessary when "upgrading" the firmware with new features.~~

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.

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

4. ~~SystemMemoryStatus~~BoardProduct
   Type: plist string
   Failsafe: ~~Auto~~Empty (Not installed)
   Description: ~~Indicates whether system memory is upgradable in~~ Sets board-id in ~~PlatformFeature~~gEfiMiscSubClassGuid
   ~~This controls the visibility of the Memory tab in "About This Mac".~~

   ~~Valid values:~~

5. ~~Auto — use the original PlatformFeature value .~~

6. The value found on Macs is equal to SMBIOS ~~Upgradable-- explicitly unset PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMOR~~
   ~~(0x2) in PlatformFeature~~BoardProduct in ASCII.

7. ~~Soldered~~BoardRevision~~ — explicitly set PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY (0x2) in PlatformFeature.~~

   ~~*Note*: On certain Mac models, such as the MacBookPro10,x and any MacBookAir, SPMemoryReporter.spreporter will ignore PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY and assume that system memory is non-upgradable.~~

8. ~~ProcessorType~~
   Type: plist ~~integer~~data, 1 byte
   Failsafe: 0~~(Automatic)~~
   Description: ~~Refer to SMBIOS~~ Sets ~~ProcessorType.~~

9. ~~SystemProductNameType: plist stringFailsafe: Empty (OEM specified or not installed)Description: Refer to SMBIOS SystemProductName~~board-rev in gEfiMiscSubClassGuid. The value found on Macs seems to correspond to internal board revision (e.g. 01).

10. ~~SystemSerialNumber~~DevicePathsSupported
    Type: plist ~~string~~integer, 32-bit
    Failsafe: ~~Empty (OEM specified or not~~ 0 (Not installed)
    Description: ~~Refer to SMBIOS~~ Sets DevicePathsSupported in gEfiMiscSubClassGuid. Must be set to ~~SystemSerialNumber.~~

    ~~Specify special string value OEM to extract current value from NVRAM (SSN1variable) or SMBIOS and use it throughout the sections. This feature can only be used on Mac-compatible firmware~~for AppleACPIPlatform.kext to append SATA device paths to Boot#### and efi-boot-device-data variables. Set to 1 on all modern Macs.

11. ~~SystemUUID~~FSBFrequency
    Type: plist ~~string~~integer, ~~GUID~~64-bit
    Failsafe: ~~Empty (OEM specified or not installed~~0 (Automatic)
    Description: ~~Refer to SMBIOS~~ Sets ~~SystemUUID~~FSBFrequency in gEfiProcessorSubClassGuid.

    ~~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~~ Sets CPU FSB frequency. This value equals to CPU nominal frequency divided by CPU maximum bus ratio and is specified in Hz. Refer to ~~UseRawUuidEncoding~~MSR_NEHALEM_PLATFORM_INFO~~to determine how SMBIOS value is parsed.~~

12. ~~`MLB`**Type**: `plist string`**Failsafe**: Empty (OEM specified or not installed) **Description**: Refer to SMBIOS `BoardSerialNumber`.~~ (`CEh`) MSR value to determine maximum bus ratio on modern Intel CPUs.

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

13. ~~`ROM`~~`InitialTSC`
    Type: `plist` ~~`multidata`~~`integer`, ~~6 bytes~~64-bit
    Failsafe: ~~Empty (OEM specified or not installed)~~`0`
    Description: ~~Refer to~~ Sets ~~`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~~

`InitialTSC` in `gEfiProcessorSubClassGuid`. Sets initial TSC value, normally 0.

14. `PlatformName`
    Type: `plist string`
    Failsafe: Empty (Not installed)
    Description: Sets `name` in `gEfiMiscSubClassGuid`. The value found on Macs is `platform` in ASCII.

15. ~~`SystemProductName`~~`SmcBranch`
    Type: `plist` ~~`string`~~`data`, 8 bytes
    Failsafe: Empty (Not installed)
    Description: Sets ~~`Model`~~`RBr` in `gEfiMiscSubClassGuid`. ~~The value found on Macs is equal to SMBIOS~~ Custom property read by `VirtualSMC` or `FakeSMC` to generate SMC ~~`SystemProductName`~~`RBr` ~~in Unicode~~key.

16. ~~`SystemSerialNumber`~~`SmcPlatform`
    Type: `plist` ~~`string`~~`data`, 8 bytes
    Failsafe: Empty (Not installed)
    Description: Sets ~~`SystemSerialNumber`~~`RPlt` in `gEfiMiscSubClassGuid`. ~~The value found on Macs is equal to SMBIOS~~ Custom property read by `VirtualSMC` or `FakeSMC` to generate SMC ~~`SystemSerialNumber`~~`RPlt` ~~in Unicode~~key.

17. ~~`SystemUUID`~~`SmcRevision`
    Type: `plist` ~~`string`~~`data`, ~~GUID~~6 bytes
    Failsafe: Empty (Not installed)
    Description: Sets ~~`system-id`~~`REV` in `gEfiMiscSubClassGuid`. ~~The value found on Macs is equal to SMBIOS~~ Custom property read by `VirtualSMC` or `FakeSMC` to generate SMC ~~`SystemUUID`~~`REV` ~~(with swapped byte order)~~key.

18. ~~`BoardProduct`**Type**: `plist string`**Failsafe**: Empty (Not installed)**Description**: Sets `board-id` in `gEfiMiscSubClassGui`~~
    ~~The value found on Macs is equal to SMBIOS `BoardProduct` in ASCII.~~

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

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

    - `0x00000001` — Shutdown cause was a `PWROK` event (Same as `GEN_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)

21. ~~`InitialTSC`~~`SystemProductName`
    Type: plist ~~`integer`~~`string`~~, 64 bit~~
    Failsafe: ~~`0`~~Empty (Not installed)
    Description: Sets ~~`InitialTSC`~~`Model` in ~~`gEfiProcessorSubClassGuid`~~`gEfiMiscSubClassGuid`. ~~Sets initial TSC value , normally 0.~~ The value found on Macs is equal to SMBIOS `SystemProductName` in Unicode.

22. ~~`FSBFrequency`~~`SystemSerialNumber`
    Type: plist ~~`integer`~~`string`~~, 64 bit~~
    Failsafe: ~~`0` (Automatic~~Empty (Not installed)
    Description: Sets ~~`FSBFrequency`~~`SystemSerialNumber` in ~~`gEfiProcessorSubClassGuid`~~`gEfiMiscSubClassGuid`.

    ~~Sets CPU FSB frequency. This value equals to CPU nominal frequency divided by CPU maximum bus ratio and is specified in Hz. Refer to~~ The value found on Macs is equal to SMBIOS `SystemSerialNumber` in Unicode.

23. ~~`MSR_NEHALEM_PLATFORM_INFO`~~`SystemUUID` ~~(CEh)MSR value to determine maximum bus ratio on modern Intel CPUs.~~
    Type: plist `string`, GUID
    Failsafe: Empty (Not installed)
    Description: Sets `system-id` in `gEfiMiscSubClassGuid`. The value found on Macs is equal to SMBIOS `SystemUUID` (with swapped byte order).

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

10.3 Generic Properties

1. ~~`ARTFrequency`~~`AdviseFeatures`
   Type: plist ~~`integer`~~`boolean`~~, 64 bit~~
   Failsafe: ~~`0`~~`false`~~(Automatic)~~
   Description: ~~Sets~~ Updates ~~`ARTFrequency`~~`FirmwareFeatures` ~~in `gEfiProcessorSubClassGuid`~~with supported bits.

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

   - `FW_FEATURE_SUPPORTS_CSM_LEGACY_MODE` (`0x1`) - Without this bit, it is not possible to reboot to Windows installed on a drive with an EFI partition that is not the first partition on the disk.
   - `FW_FEATURE_SUPPORTS_UEFI_WINDOWS_BOOT` (`0x20000000`) - Without this bit, it is not possible to reboot to Windows installed on a drive with an EFI partition that is the first partition on the disk.
   - `FW_FEATURE_SUPPORTS_APFS` (`0x00080000`) - Without this bit, it is not possible to install macOS on an APFS disk.
   - `FW_FEATURE_SUPPORTS_LARGE_BASESYSTEM` (`0x800000000`) - Without this bit, it is not possible to install macOS versions with large BaseSystem images, such as macOS 12.

   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.~~ most newer firmwares these bits are already set, the option may be necessary when "upgrading" the firmware with new features.

2. `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.

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

4. ~~`DevicePathsSupported`~~`ProcessorType`
   Type: `plist integer`~~, 32 bit~~
   Failsafe: `0` (~~Not~~ Automatic)
   Description: Refer to SMBIOS `ProcessorType`.

5. `ROM`
   Type: `plist multidata`, 6 bytes
   Failsafe: Empty (OEM specified or not installed)
   Description: ~~Sets~~ 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.

6. `SpoofVendor`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Sets SMBIOS vendor fields to ~~`DevicePathsSupported`~~`Acidanthera`.

   It can be dangerous to use "Apple" in SMBIOS vendor fields for reasons outlined in the `SystemManufacturer` description. However, certain firmware may not provide valid values otherwise, which could obstruct the operation of some software.

7. `SystemMemoryStatus`~~in `gEfiMiscSubClassGuid`. Must be set to `1` for AppleACPIPlatform. kext to append SATA device paths to~~
   Type: `plist string`
   Failsafe: `Auto`
   Description: Indicates whether system memory is upgradable in `PlatformFeature`. This controls the visibility of the Memory tab in "About This Mac".

   Valid values:

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

   Note: On certain Mac models, such as the `MacBookPro10,x` and any ~~`Boot####`and `efi-boot-device-data`~~`MacBookAir`~~variab~~
   ~~to~~, SPMemoryReporter.spreporter will ignore ~~`1`~~`PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` ~~on all modern Macs.~~
   and assume that system memory is non-upgradable.

8. ~~`SmcRevision`~~`SystemProductName`
   Type: `plist` ~~`data`~~`string`~~, 6 bytes~~
   Failsafe: Empty (~~Not~~ OEM specified or not installed)
   Description: ~~Sets `REV` in `gEfiMiscSubClassGuid`. Custom property read by `VirtualSMC` or `FakeSMC` to generate SMC~~ Refer to SMBIOS ~~`REV`~~`SystemProductName`~~key~~.

9. ~~`SmcBranch`~~`SystemSerialNumber`
   Type: `plist` ~~`data`~~`string`~~, 8 bytes~~
   Failsafe: Empty (~~Not~~ OEM specified or not installed)
   Description: ~~Sets~~ Refer to SMBIOS ~~`RBr`in `gEfiMiscSubClassGuid`. Custom property read by~~ `SystemSerialNumber`.

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

10. ~~SmcPlatform~~SystemUUID
Type: plist ~~data~~string, ~~8 bytes~~GUID
Failsafe: Empty (~~Not~~ OEM specified or not installed)
Description: ~~Sets~~ Refer to SMBIOS ~~RPltin gEfiMiscSubClassGuid. Custom property read by~~ SystemUUID.

Specify special string value ~~VirtualSMCor FakeSMC to generate SMC RPlt key.~~ 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.

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.

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

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

- `0x01` — Other
- `0x02` — Unknown
- `0x03` — None
- `0x04` — Parity
- `0x05` — Single-bit ECC
- `0x06` — Multi-bit ECC
- `0x07` — CRC

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

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

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

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

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

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

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

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

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

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

10.4.1 Memory Device Properties

1. `AssetTag`
   Type: `plist string`
   Failsafe: `Unknown`
   SMBIOS: Memory Device (Type 17) — Asset Tag
   Description: Specifies the asset tag of this memory device.

2. `BankLocator`
   Type: `plist string`
   Failsafe: `Unknown`
   SMBIOS: Memory Device (Type 17) — Bank Locator
   Description: Specifies the physically labeled bank where the memory device is located.

3. `DeviceLocator`
   Type: `plist string`

Failsafe: `Unknown`
SMBIOS: Memory Device (Type 17) — Device Locator
Description: Specifies the physically-labeled socket or board position where the memory device is located.

4. `Manufacturer`
   Type: `plist string`
   Failsafe: `Unknown`
   SMBIOS: Memory Device (Type 17) — Manufacturer
   Description: Specifies the manufacturer of this memory device.

   For empty slot this must be set to `NO DIMM` for macOS System Profiler to correctly display memory slots on certain Mac models, e.g. `MacPro7,1`. `MacPro7,1` imposes additional requirements on the memory layout:

   - The amount of installed sticks must one of the following: 4, 6, 8, 10, 12. Using any different value will cause an error in the System Profiler.
   - The amount of memory slots must equal to 12. Using any different value will cause an error in the System Profiler.
   - Memory sticks must be installed in dedicated memory slots as explained on the support page. SMBIOS memory devices are mapped to the following slots: `8, 7, 10, 9, 12, 11, 5, 6, 3, 4, 1, 2`.

5. `PartNumber`
   Type: `plist string`
   Failsafe: `Unknown`
   SMBIOS: Memory Device (Type 17) — Part Number
   Description: Specifies the part number of this memory device.

6. `SerialNumber`
   Type: `plist string`
   Failsafe: `Unknown`
   SMBIOS: Memory Device (Type 17) — Serial Number
   Description: Specifies the serial number of this memory device.

7. `Size`
   Type: `plist integer`, 32-bit
   Failsafe: `0`
   SMBIOS: Memory Device (Type 17) — Size
   Description: Specifies the size of the memory device, in megabytes. `0` indicates this slot is not populated.

8. `Speed`
   Type: `plist integer`, 16-bit
   Failsafe: `0`
   SMBIOS: Memory Device (Type 17) — Speed
   Description: Specifies the maximum capable speed of the device, in megatransfers per second (MT/s). `0` indicates an unknown speed.

10.5 PlatformNVRAM Properties

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

2. ~~`ROM`~~`FirmwareFeatures`
   Type: `plist data`, ~~6~~ 8 bytes
   Failsafe: Empty (Not installed)
   Description: This variable comes in pair with `FirmwareFeaturesMask`. Specifies the values of NVRAM variables:

   - `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:`~~`HW_ROM`~~`FirmwareFeatures` ~~and~~
   - `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:`~~`ROM`~~`ExtendedFirmwareFeatures` ~~.~~

3. ~~`MLB`~~`FirmwareFeaturesMask`
   Type: `plist` ~~`string`~~data, 8 bytes

Failsafe: Empty (Not installed)
Description: This variable comes in pair with FirmwareFeatures. Specifies the values of NVRAM variables:

- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:~~HW_MLB~~FirmwareFeaturesMask ~~and~~
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:~~MLB~~ExtendedFirmwareFeaturesMask ~~.~~

4. ~~FirmwareFeatures~~MLB
Type: plist ~~data~~string~~, 8 bytes~~
Failsafe: Empty (Not installed)
Description: ~~This variable comes in pair with FirmwareFeaturesMask.~~ Specifies the values of NVRAM variables ~~:~~

5. 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:~~FirmwareFeatures~~HW_MLB

6. and 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:~~ExtendedFirmwareFeatures~~MLB.

7. ~~FirmwareFeaturesMask~~ROM
Type: plist data, ~~8~~ 6 bytes
Failsafe: Empty (Not installed)
Description: ~~This variable comes in pair with FirmwareFeatures.~~ Specifies the values of NVRAM variables ~~:~~

8. 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:~~FirmwareFeaturesMask~~HW_ROM

9. and 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:~~ExtendedFirmwareFeaturesMask~~ROM.

10. 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.

11. SystemUUID
Type: plist string
Failsafe: Empty (Not installed)
Description: Specifies the value of NVRAM variable 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:system-id for boot services only. The value found on Macs is equal to SMBIOS SystemUUID.

10.6 SMBIOS Properties

1. BIOSReleaseDate
Type: plist string
Failsafe: Empty (OEM specified)
SMBIOS: BIOS Information (Type 0) — BIOS Release Date
Description: Firmware release date. Similar to BIOSVersion. May look like 12/08/2017.

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

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

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

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

4. ~~`BIOSReleaseDate`~~`BoardAssetTag`
   Type: `plist string`
   Failsafe: Empty (OEM specified)
   SMBIOS: ~~BIOS Information (Type 0) — BIOS Release Date **Description**: Firmware release date. Similar to `BIOSVersion`. May look like `12/08/2017`.~~

5. ~~`SystemManufacturer` **Type**: `plist string` **Failsafe**: Empty (OEM specified) **SMBIOS**: System~~ Baseboard (or Module) Information (Type ~~1~~2) — ~~Manufacturer~~Asset Tag
   Description: ~~OEM manufacturer of the particular board. Use failsafe unless strictly required. Do not override to contain `Apple Inc.` on non-Apple hardware, as this confuses numerous services present in the operating system, such as firmware updates, 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.~~ Asset tag number. Varies, may be empty or `Type2 - Board Asset Tag`.

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

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

7. ~~`SystemVersion`~~`BoardLocationInChassis`
   Type: `plist string`
   Failsafe: Empty (OEM specified)
   SMBIOS: ~~System Information (Type 1) — Version **Description**: Product iteration version number. May look like `1.1`.~~

8. ~~`SystemSerialNumber` **Type**: `plist string` **Failsafe**: Empty (OEM specified) **SMBIOS**: System~~ Baseboard (or Module) Information (Type ~~1~~2) — ~~Serial Number~~Location in Chassis
   Description~~: Product serial number in defined format. Known formats are described in macserial.~~

9. ~~`SystemUUID` **Type**~~: ~~`plist string`, GUID **Failsafe**: Empty (OEM specified) **SMBIOS**: System Information (Type 1) — UUID **Description**: A UUID is an identifier that is designed to be unique across both time and space. It requires no central registration process.~~

10. ~~`SystemSKUNumber` **Type**: `plist string` **Failsafe**: Empty (OEM specified) **SMBIOS**: System Information (Type 1) — SKU Number **Description**: Mac Board ID (`board-id`). May look like `Mac-7BA5B2D9E42DDD94` or~~ Varies, may be empty or ~~`Mac-F221BEC8 in older models. Sometimes it can be just empty.`~~ `Part Component`.

11. ~~`SystemFamily` **Type**: `plist string` **Failsafe**: Empty (OEM specified) **SMBIOS**: System Information (Type 1) — Family **Description**: Family name. May look like `iMac Pro`.~~

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

13. `BoardProduct`
    Type: `plist string`
    Failsafe: Empty (OEM specified)

SMBIOS: Baseboard (or Module) Information (Type 2) - Product
Description: Mac Board ID (`board-id`). May look like `Mac-7BA5B2D9E42DDD94` or `Mac-F221BEC8` in older models.

14. ~~`BoardVersion`**Type**: `plist string`**Failsafe**: Empty (OEM specified)**SMBIOS**: Baseboard (or Module) Information (Type 2) - Version**Description**: Board version number. Varies, may match `SystemProductName` or `SystemProductVersion`.~~

15. `BoardSerialNumber`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) — Serial Number
Description: Board serial number in defined format. Known formats are described in macserial.

16. ~~`BoardAssetTag`**Type**: `plist string`**Failsafe**: Empty (OEM specified)**SMBIOS**: Baseboard (or Module) Information (Type 2) — Asset Tag**Description**: Asset tag number. Varies, may be empty or `Type2 - Board Asset Tag`.~~

17. `BoardType`
Type: `plist integer`
Failsafe: `0` (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) — Board Type
Description: Either `0xA` (Motherboard (includes processor, memory, and I/O) or `0xB` (Processor/Memory Module). Refer to Table 15 – Baseboard: Board Type for details.

18. ~~`BoardLocationInChassis`~~`BoardVersion`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: Baseboard (or Module) Information (Type 2) ~~— Location in Chassis~~ Version
Description: Board version number. Varies, may ~~be empty or~~ match `SystemProductName` or ~~`Part Component`~~`SystemProduc`

19. ~~`ChassisManufacturer`~~`ChassisAssetTag`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: System Enclosure or Chassis (Type 3) — ~~Manufacturer~~Asset Tag Number
Description: ~~Board manufacturer. All rules of `SystemManufacturer` do apply.~~

20. ~~`ChassisType`**Type**: `plist integer`**Failsafe**: `0` (OEM specified)**SMBIOS**: System Enclosure or Chassis (Type 3) — Type**Description**: Chassis type~~ Chassis type name. Varies, could be empty or `MacBook-Aluminum`. ~~Refer to Table 17 — System Enclosure or Chassis Types for details.~~

21. ~~`ChassisVersion`~~`ChassisManufacturer`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: System Enclosure or Chassis (Type 3) — ~~Version~~Manufacturer
Description: ~~Should match `BoardProduct`.~~ Board manufacturer. All rules of `SystemManufacturer` do apply.

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

23. ~~`ChassisAssetTag`~~`ChassisType`
Type: `plist` ~~`string`~~`integer`
Failsafe: ~~Empty~~ `0` (OEM specified)
SMBIOS: System Enclosure or Chassis (Type 3) — ~~Asset Tag Number~~Type
Description: Chassis type~~name. Varies, could be empty or `MacBook-Aluminum`~~.

24. ~~`PlatformFeature`**Type**: `plist integer, 32-bit`**Failsafe**: `0xFFFFFFFF` (OEM specified on Apple hardware, do not provide the table otherwise)**SMBIOS**: `APPLE_SMBIOS_TABLE_TYPE133 - PlatformFeature`**Description**: Platform features bitmask (Missing on older Macs). Refer to AppleFeatures.h~~ Refer to Table 17 — System Enclosure or Chassis Types for details.

25. ~~`SmcVersion`~~`ChassisVersion`
Type: `plist` ~~`data`~~`string`~~, 16 bytes~~
Failsafe: ~~All zero~~ Empty (OEM specified~~on Apple hardware, do not provide the table otherwise~~)
SMBIOS: ~~`APPLE_SMBIOS_TABLE_TYPE134 - Version`~~System Enclosure or Chassis (Type 3) — Version
Description: ~~ASCII string containing SMC version in upper case. Missing on T2 based Macs.~~ Should match `BoardProduct`.

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

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

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

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

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

31. `SystemFamily`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — Family
Description: Family name. May look like `iMac Pro`.

32. `SystemManufacturer`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — Manufacturer
Description: OEM manufacturer of the particular board. Use failsafe unless strictly required. Do not override to contain `Apple Inc.` on non-Apple hardware, as this confuses numerous services present in the operating system, such as firmware updates, 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.

33. `SystemProductName`
Type: `plist string`
Failsafe: Empty (OEM specified)

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

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

34. `SystemSKUNumber`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — SKU Number
Description: Mac Board ID (`board-id`). May look like `Mac-7BA5B2D9E42DDD94` or `Mac-F221BEC8` in older models. Sometimes it can be just empty.

35. `SystemSerialNumber`
Type: `plist string`
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — Serial Number
Description: Product serial number in defined format. Known formats are described in macserial.

36. `SystemUUID`
Type: `plist string`, GUID
Failsafe: Empty (OEM specified)
SMBIOS: System Information (Type 1) — UUID
Description: A UUID is an identifier that is designed to be unique across both time and space. It requires no central registration process.

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

11 UEFI

11.1 Introduction

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

11.2 Drivers

Depending on the firmware, a different set of drivers may be required. Loading an incompatible driver may lead the system to unbootable state or even cause permanent firmware damage. Some of the known drivers are listed below:

`AudioDxe*`	HDA audio support driver in UEFI firmware for most Intel and some other analog audio controllers. Staging driver, refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
`btrfs_x64`	Open source BTRFS file system driver, required for booting with OpenLinuxBoot from a file system which is now quite commonly used with Linux.
`BiosVideo*`	CSM video driver implementing graphics output protocol based on VESA and legacy BIOS interfaces. Used for UEFI firmware with fragile GOP support (e.g. low resolution). Requires `ReconnectGraphicsOnConnect`. Included in OpenDuet out of the box.
`CrScreenshotDxe*`	Screenshot making driver saving images to the root of OpenCore partition (ESP) or any available writeable filesystem upon pressing `F10`. Accepts optional driver argument `--enable-mouse-click` to additionally take screenshot on mouse click. (It is recommended to enable this option only if a keypress would prevent a specific screenshot, and disable it again after use.) This is a modified version of `CrScreenshotDxe` driver by Nikolaj Schlej.
`EnableGop{Direct}*`	Early beta release firmware-embeddable driver providing pre-OpenCore non-native GPU support on MacPro5,1. Installation instructions can be found in the `Utilities/EnableGop` directory of the OpenCore release zip file - proceed with caution.
`ExFatDxe`	Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `ExFatDxeLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`ext4_x64`	Open source EXT4 file system driver, required for booting with OpenLinuxBoot from the file system most commonly used with Linux.
`FirmwareSettings*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to add an entry to the boot picker menu which reboots into UEFI firmware settings, when this is supported by the firmware.
`HfsPlus`	Recommended. Proprietary HFS file system driver with bless support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `HfsPlusLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`HiiDatabase*`	HII services support driver from `MdeModulePkg`. This driver is included in most types of firmware starting with the Ivy Bridge generation. Some applications with GUI, such as UEFI Shell, may need this driver to work properly.
`EnhancedFatDxe`	FAT filesystem driver from `FatPkg`. This driver is embedded in all UEFI firmware and cannot be used from OpenCore. Several types of firmware have defective FAT support implementation that may lead to corrupted filesystems on write attempts. Embedding this driver within the firmware may be required in case writing to the EFI partition is needed during the boot process.
`NvmExpressDxe*`	NVMe support driver from `MdeModulePkg`. This driver is included in most firmware starting with the Broadwell generation. For Haswell and earlier, embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
`OpenCanopy*`	OpenCore plugin implementing graphical interface.
`OpenRuntime*`	OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol.
`OpenLegacyBoot*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to allow detection and booting of legacy operating systems from OpenCore on Macs, OpenDuet and systems with a CSM.

any, from `nvram.fallback` are used instead. `Launchd.command` itself always copies the previous NVRAM settings to fallback, each time it saves new settings.

This strategy is used to work round the limitation that the `Launchd.command` script is not running, and therefore cannot save NVRAM changes (particularly default boot entry changes), during the second and subsequent restarts of the macOS installer.

In brief, this fallback strategy allows full or incremental OTA updates of recent macOS, which are started from within an existing macOS (with the `Launchd.command` script installed), to proceed without manual intervention.

However, for full installs, there can be more than one full restart back to the `macOS Installer` entry. In this case the fallback strategy will lose track of the correct startup item (i.e. `macOS Installer`) from the second reboot onwards. Equally, if installing to a drive other than the current default boot partition, this will not be automatically selected after the installer completes, as it would be when using non-emulated NVRAM. (This behaviour remains preferable to not having the fallback strategy, in which case a `macOS Installer` entry would be continually recreated in the picker menu, even once it no longer exists).

In both the above two cases it is recommended to use the following settings, to make it easy to manually control which boot entry is selected during the installer process:

- Set `ShowPicker=true`.
- Set `Timeout=0`.
- Set `DisableWatchdog=true`.
- If possible, start from a situation where there are no other pending `macOS Installer` entries in the boot menu (to avoid potential confusion as to which is relevant).

The first reboot should correctly select `macOS Installer`. For second and subsequent reboots, if a `macOS Installer` entry is still present it should be manually selected (using just Enter, not `CTRL+Enter`). Once a `macOS Installer` entry is no longer present, the entry for the new OS will still be automatically selected if it was the previous boot default. If not, it should be manually selected (at this point, `CTRL+Enter` is a good idea as any final remaining ~~installion~~ installation restarts will be to this entry).

Note 1: When using emulated NVRAM but not installing from within an existing installed macOS (i.e. when installing from within macOS Recovery, or from an installation USB), please refer to this forum post (in Russian) for additional options.

Note 2: After upgrading from an ~~ealier~~ earlier macOS version to macOS Sonoma, the `Launchd.command` script should be reinstalled, as a different strategy is required in order for NVRAM to be saved successfully.

Note 3: In macOS Sonoma the following additional constraints apply to the ESP ~~paritition~~ partition on which OpenCore is installed, in order for the `Launchd.command` script to work successfully:

- It must not be set to automount.
- It must be unmounted again before shutdown or restart if it was manually mounted.

11.11 Properties

1. `APFS`
   Type: `plist dict`
   Description: Provide APFS support as configured in the APFS Properties section below.

2. `AppleInput`
   Type: `plist dict`
   Description: Configure the re-implementation of the Apple Event protocol described in the AppleInput Properties section below.

3. `Audio`
   Type: `plist dict`
   Description: Configure audio backend support described in the `Audio Properties` section below.

   Unless documented otherwise (e.g. `ResetTrafficClass`) settings in this section are for UEFI audio support only (e.g. OpenCore generated boot chime and audio assist) and are unrelated to any configuration needed for OS audio support (e.g. `AppleALC`).

compromise a computer if such drivers are used after support ends. This option permits restricting APFS drivers to current macOS versions.

- `0` — require the default supported version of APFS in OpenCore. The default version will increase with time and thus this setting is recommended. Currently set to allow macOS Big Sur and newer (`1600000000000000`).
- `-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`.

11.13 AppleInput Properties

1. `AppleEvent`
   Type: `plist string`
   Failsafe: `Auto`
   Description: Determine whether the OpenCore builtin or the OEM Apple Event protocol is used.

   This option determines whether the OEM Apple Event protocol is used (where available), or whether OpenCore's reversed engineered and updated re-implementation is used. In general OpenCore's re-implementation should be preferred, since it contains updates such as noticeably improved fine mouse cursor movement and configurable key repeat delays.

   - `Auto` — Use the OEM Apple Event implementation if available, connected and recent enough to be used, otherwise use the OpenCore re-implementation. On non-Apple hardware, this will use the OpenCore builtin implementation. On some Macs such as Classic Mac Pros, this will prefer the Apple implementation but on both older and newer Mac models than these, this option will typically use the OpenCore re-implementation instead. On older Macs, this is because the implementation available is too old to be used while on newer Macs, it is because of optimisations added by Apple which do not connect the Apple Event protocol except when needed – e.g. except when the Apple boot picker is explicitly started. Due to its somewhat unpredicatable results, this option is not typically recommended.
   - `Builtin` — Always use OpenCore's updated re-implementation of the Apple Event protocol. Use of this setting is recommended even on Apple hardware, due to improvements (better fine mouse control, configurable key delays) made in the OpenCore re-implementation of the protocol.
   - `OEM` — Assume Apple's protocol will be available at driver connection. On all Apple hardware where a recent enough Apple OEM version of the protocol is available – whether or not connected automatically by Apple's firmware – this option will reliably access the Apple implementation. On all other systems, this option will result in no keyboard or mouse support. For the reasons stated, `Builtin` is recommended in preference to this option in most cases.

2. `CustomDelays`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable custom key repeat delays when using the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

   - `true` — The values of `KeyInitialDelay` and `KeySubsequentDelay` are used.
   - `false` — Apple default values of 500ms (`50`) and 50ms (`5`) are used.

3. `GraphicsInputMirroring`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Apple's own implementation of AppleEvent prevents keyboard input during graphics applications from appearing on the basic console input stream.

   With the default setting of `false`, OpenCore's builtin implementation of AppleEvent replicates this behaviour.

   On non-Apple hardware this can stop keyboard input working in graphics-based applications such as Windows BitLocker which use non-Apple key input methods.

   The recommended setting on all hardware is `true`.

   Note: AppleEvent's default behaviour is intended to prevent unwanted queued keystrokes from appearing after exiting graphics-based UEFI applications; this issue is already handled separately within OpenCore.

- `true` — Allow keyboard input to reach graphics mode apps which are not using Apple input protocols.
- `false` — Prevent key input mirroring to non-Apple protocols when in graphics mode.

4. `KeyInitialDelay`
   Type: `plist integer`
   Failsafe: `50` (500ms before first key repeat)
   Description: Configures the initial delay before keyboard key repeats in the OpenCore re-implementation of the Apple Event protocol, in units of 10ms.

   The Apple OEM default value is `50` (500ms).

   Note 1: On systems not using `KeySupport`, this setting may be freely used to configure key repeat behaviour.

   Note 2: On systems using `KeySupport`, but which do not show the 'two long delays' behavior (see Note 3) and/or which always show a solid 'set default' indicator (see `KeyForgetThreshold`) then this setting may also be freely used to configure key repeat initial delay behaviour, except that it should never be set to less than `KeyForgetThreshold` to avoid uncontrolled key repeats.

   Note 3: On some systems using `KeySupport`, you may find that you see one additional slow key repeat before normal speed key repeat starts, when holding a key down. If so, you may wish to configure `KeyInitialDelay` and `KeySubsequentDelay` according to the instructions at Note 3 of `KeySubsequentDelay`.

5. `KeySubsequentDelay`
   Type: `plist integer`
   Failsafe: `5` (50ms between subsequent key repeats)
   Description: Configures the gap between keyboard key repeats in the OpenCore re-implementation of the Apple Event protocol, in units of 10ms.

   The Apple OEM default value is `5` (50ms). `0` is an invalid value for this option (will issue a debug log warning and use `1` instead).

   Note 1: On systems not using `KeySupport`, this setting may be freely used to configure key repeat behaviour.

   Note 2: On systems using `KeySupport`, but which do not show the 'two long delays' behaviour (see Note 3) and/or which always show a solid 'set default' indicator (see `KeyForgetThreshold`) (which should apply to many/most systems using `AMI KeySupport` mode) then this setting may be freely used to configure key repeat subsequent delay behaviour, except that it should never be set to less than `KeyForgetThreshold` to avoid uncontrolled key repeats.

   Note 3: On some systems using `KeySupport`, particularly `KeySupport` in non-`AMI` mode, you may find that after configuring `KeyForgetThreshold` you get one additional slow key repeat before normal speed key repeat starts, when holding a key down. On systems where this is the case, it is an unavoidable artefect of using `KeySupport` to emulate raw keyboard data, which is not made available by UEFI. While this 'two long delays' issue has minimal effect on overall usability, nevertheless you may wish to resolve it, and it is possible to do so as follows:

   - Set `CustomDelays` to `true`
   - Set `KeyInitialDelay` to `0`
   - Set `KeySubsequentDelay` to at least the value of your `KeyForgetThreshold` setting

   The above procedure works as follows:

   - Setting `KeyInitialDelay` to `0` cancels the Apple Event initial repeat delay (when using the OpenCore builtin Apple Event implementation with `CustomDelays` enabled), therefore the only long delay you will see is the the non-configurable and non-avoidable initial long delay introduced by the BIOS key support on these machines.
   - Key-smoothing parameter `KeyForgetThreshold` effectively acts as the shortest time for which a key can appear to be held, therefore a key repeat delay of less than this will guarantee at least one extra repeat for every key press, however quickly the key is physically tapped.
   - In the unlikely event that you still get frequent, or occasional, double key responses after setting `KeySubsequentDelay` equal to your system's value of `KeyForgetThreshold`, then increase `KeySubsequentDelay` by one or two more until this effect goes away.

6. ~~`GraphicsInputMirroring`~~`PointerDwellClickTimeout`
   Type: `plist` ~~`boolean`~~`integer`

Failsafe: ~~false~~0
Description: ~~Apple's own implementation of AppleEvent prevents keyboard input during graphics applications from appearing on the basic console input stream.~~

~~With the default setting of false, OpenCore's builtin implementation of AppleEvent replicates this behaviour~~Configure pointer dwell-clicking single left click timeout in milliseconds in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

~~On non-Apple hardware this can stop keyboard input working in graphics-based applications such as Windows BitLocker which use non-Apple key input methods.~~

~~The recommended setting on all hardware is true.~~

~~*Note*: AppleEvent's default behaviour is intended to prevent unwanted queued keystrokes from appearing after exiting graphics-based UEFI applications; this issue is already handled separately within OpenCore~~When the timeout expires, a single left click is issued at the current position. `0` indicates the timeout is disabled.

7. ~~true~~`PointerDwellDoubleClickTimeout`~~ — Allow keyboard input to reach graphics mode apps which are not using Apple input protocols.~~

8. ~~false — Prevent key input mirroring to non-Apple protocols when in graphics mode.~~

9. ~~PointerPollMin~~
Type: `plist integer`
Failsafe: `0`
Description: Configure ~~minimal pointer polling period in ms.~~ pointer dwell-clicking single left double click timeout in milliseconds in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

~~This is the minimal period the OpenCore builtin AppleEvent driver polls pointer devices (e.g. mice, trackpads)for motion events. The current implementation defaults to 10 ms. Setting~~ When the timeout expires, a single left double click is issued at the current position. `0` ~~leaves this default unchanged~~indicates the timeout is disabled.

~~*Note*: The OEM Apple implementation uses a polling rate of 2 ms.~~

10. ~~PointerPollMax~~`PointerDwellRadius`
Type: `plist integer`
Failsafe: `0`
Description: Configure ~~maximum pointer polling period in ms.~~

~~This is the maximum period the OpenCore builtin AppleEvent driver polls pointer devices (e.g. mice, trackpads)for motion events. The period is increased up to this value as long as the devices do not respond in time. The current implementation defaults to 80 ms. Setting 0 leaves this default unchanged.~~

~~Certain trackpad drivers often found in Dell laptops can be very slow to respond when no physical movement happens. This can affect OpenCanopy and FileVault 2 user interface responsiveness and loading times. Increasing the polling periods can reduce the impact~~pointer dwell-clicking tolerance radius in pixels in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

The radius is scaled by `UIScale`. When the pointer leaves this radius, the timeouts for `PointerDwellClickTimeout` and `PointerDwellDoubleClickTimeout` are reset and the new position is the centre for the new dwell-clicking tolerance radius.

~~*Note*: The OEM Apple implementation uses a polling rate of 2 ms.~~

11. `PointerPollMask`
Type: `plist integer, 32 bit`
Failsafe: `-1`
Description: Configure indices of polled pointers.

Selects pointer devices to poll for AppleEvent motion events. `-1` implies all devices. A bit sum is used to determine particular devices. E.g. to enable devices 0, 2, 3 the value will be `1+4+8` (corresponding powers of two). A total of 32 configurable devices is supported.

Certain pointer devices can be present in the firmware even when no corresponding physical devices are available. These devices usually are placeholders, aggregate devices, or proxies. Gathering information from these devices may result in inaccurate motion activity in the user interfaces and even cause performance issues. Disabling such pointer devices is recommended for laptop setups having issues of this kind.

The amount of pointer devices available in the system can be found in the log. Refer to `Found N pointer devices` message for more details.

Note: Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

12. `PointerPollMax`
Type: `plist integer`
Failsafe: `0`
Description: Configure maximum pointer polling period in ms.

    This is the maximum period the OpenCore builtin AppleEvent driver polls pointer devices (e.g. mice, trackpads) for motion events. The period is increased up to this value as long as the devices do not respond in time. The current implementation defaults to 80 ms. Setting `0` leaves this default unchanged.

    Certain trackpad drivers often found in Dell laptops can be very slow to respond when no physical movement happens. This can affect OpenCanopy and FileVault 2 user interface responsiveness and loading times. Increasing the polling periods can reduce the impact.

    Note: The OEM Apple implementation uses a polling rate of 2 ms.

13. `PointerPollMin`
Type: `plist integer`
Failsafe: `0`
Description: Configure minimal pointer polling period in ms.

    This is the minimal period the OpenCore builtin AppleEvent driver polls pointer devices (e.g. mice, trackpads) for motion events. The current implementation defaults to 10 ms. Setting `0` leaves this default unchanged.

    Note: The OEM Apple implementation uses a polling rate of 2 ms.

14. `PointerSpeedDiv`
Type: `plist integer`
Failsafe: `1`
Description: Configure pointer speed divisor in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

    Configures the divisor for pointer movements. The Apple OEM default value is `1`. `0` is an invalid value for this option.

    Note: The recommended value for this option is `1`. This value may optionally be modified in combination with `PointerSpeedMul`, according to user preference, to achieve customised mouse movement scaling.

15. `PointerSpeedMul`
Type: `plist integer`
Failsafe: `1`
Description: Configure pointer speed multiplier in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).

    Configures the multiplier for pointer movements. The Apple OEM default value is `1`.

    Note: The recommended value for this option is `1`. This value may optionally be modified in combination with `PointerSpeedDiv`, according to user preference, to achieve customised mouse movement scaling.

16. ~~`PointerDwellClickTimeout`**Type**: `plist integer`**Failsafe**: `0`**Description**: Configure pointer dwell-clicking single left click timeout in milliseconds in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).~~

    ~~When the timeout expires, a single left click is issued at the current position. `0` indicates the timeout is disabled.~~

17. ~~`PointerDwellDoubleClickTimeout`**Type**: `plist integer`**Failsafe**: `0`**Description**: Configure pointer dwell-clicking single left double click timeout in milliseconds in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).~~

    ~~When the timeout expires, a single left double click is issued at the current position. `0` indicates the timeout is disabled.~~

18. ~~`PointerDwellRadius`**Type**: `plist integer`**Failsafe**: `0`**Description**: Configure pointer dwell-clicking tolerance radius in pixels in the OpenCore re-implementation of the Apple Event protocol. Has no effect when using the OEM Apple implementation (see `AppleEvent` setting).~~

    ~~The radius is scaled by `UIScale`. When the pointer leaves this radius, the timeouts for `PointerDwellClickTimeout` and `PointerDwellDoubleClickTimeout` are reset and the new position is the centre for the new dwell-clicking tolerance radius.~~

11.14 Audio Properties

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

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

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

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

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

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

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

   If using `AudioDxe`, the available controller device paths are also output on lines formatted like this:

   ```
   HDA: Connecting controller - PciRoot(0x0)/Pci(0x1B,0x0)
   ```

   Finally, `gfxutil -f HDEF` command can be used in macOS to obtain the device path.

   Specifying an empty device path results in the first available codec and audio controller being used. The value of `AudioCodec` is ignored in this case. This can be a convenient initial option to try to get UEFI audio working. Manual settings as above will be required when this default value does not work.

3. `AudioOutMask`
   Type: `plist integer`
   Failsafe: `-1`
   Description: Bit field indicating which output channels to use for UEFI sound.

   Audio mask is 1 « audio output (equivalently 2 ˆ audio output). E.g. for audio output `0` the bitmask is `1`, for output `3` it is `8`, and for outputs `0` and `3` it is `9`.

   The number of available output nodes (`N`) for each HDA codec is shown in the debug log (marked in bold-italic), audio outputs `0` to `N - 1` may be selected:

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

11.15 Drivers Properties

1. `Arguments`
   Type: `plist string`
   Failsafe: Empty
   Description: Some OpenCore plugins accept optional additional arguments which may be specified as a string here.

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

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: If `false` this driver entry will be ignored.

4. ~~`Path`**Type**: `plist string`**Failsafe**: Empty**Description**: Path of file to be loaded as a UEFI driver from `OC/Drivers` directory.~~

5. `LoadEarly`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load the driver early in the OpenCore boot process, before NVRAM setup.

   Note: Do not enable this option unless specifically recommended to do so for a given driver and purpose.

6. ~~`Arguments`~~`Path`
   Type: `plist string`
   Failsafe: Empty
   Description: ~~Some OpenCore plugins accept optional additional arguments which may be specified as a string here~~Path of file to be loaded as a UEFI driver from `OC/Drivers` directory.

11.16 Input Properties

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

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

2. `KeyForgetThreshold`
   Type: `plist integer`
   Failsafe: `0`
   Description: Treat duplicate key presses as held keys if they arrive during this timeout, in 10 ms units. Only applies to systems using `KeySupport`.

   `AppleKeyMapAggregator` protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers which require `KeySupport` report key presses as interrupts, with automatically generated key repeat behaviour with some defined initial and subsequent delay. As a result, to emulate the raw key behaviour required by several Apple boot systems, we use a timeout to merge multiple repeated keys which are submitted within a small timeout window.

   This option allows setting this timeout based on the platform. The recommended value for the majority of platforms is from `5` (`50` milliseconds) to `7` (`70` milliseconds), although values up to `9` (`90` milliseconds) have been

Failsafe: `false`
Description: Enable internal pointer driver.

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

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

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

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

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

   The recommended value is `50000` (`5` milliseconds) or slightly higher. Select ASUS Z87 boards use `60000` for the interface. Apple boards use `100000`. In case of issues, this option can be left as `0` to not change the timer resolution.

11.17 Output Properties

1. ~~`InitialMode`~~`ClearScreenOnModeSwitch`
Type: `plist` ~~`string`~~`boolean`
Failsafe: ~~`Auto`~~`false`
Description: ~~Selects the internal `ConsoleControl` mode in which `TextRenderer` will operate.~~

   ~~Available values are `Auto`, `Text` and `Graphics`. `Text` and `Graphics` specify the named mode. `Auto` uses the current mode of the system `ConsoleControl` protocol when one exists, defaulting to `Text` mode otherwise.~~

   ~~UEFI firmware typically supports `ConsoleControl` with two rendering modes: `Graphics` and `Text`.~~ Some types of firmware ~~do not provide a native `ConsoleControl` and rendering modes. OpenCore and macOS expect text to only be shown in `Text` mode but graphics to be drawn in any mode, and this is how the OpenCore `Builtin` renderer behaves. Since this is not required by the UEFI specification, behaviour of the system `ConsoleControl` protocol, when it exists, may vary.~~

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

   ~~Currently two renderers are supported: `Builtin` and `System`. The `System` renderer uses firmware services for text rendering, however with additional options provided to sanitize the output. The `Builtin` renderer bypasses firmware services and performs text rendering on its own. Each renderer supports a different set of options. It is recommended to use the `Builtin` renderer, as it supports HiDPI modeand uses full screen resolution.~~

   ~~Each renderer provides its own `ConsoleControl` protocol (in the case of `SystemGeneric` only, this passes some operations through to the system `ConsoleControl` protocol, if one exists).~~

   ~~Valid values of this option are combinations of the renderer to use and the `ConsoleControl` mode to set on the underlying system `ConsoleControl` protocol before starting. To control the initial mode of the provided `ConsoleControl` protocol once started, use the `InitialMode` option.~~

3. ~~`BuiltinGraphics` — Switch to `Graphics` modethen use `Builtin` renderer with custom `ConsoleControl`.~~

4. ~~`BuiltinText` — Switch to `Text` mode then use `Builtin` renderer with custom `ConsoleControl`~~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.

5. ~~`SystemGraphics` — Switch to `Graphics` mode then use `System` renderer with custom `ConsoleControl`.~~

6. ~~SystemText — Switch to Text mode then use System renderer with custom ConsoleControl.~~

7. ~~SystemGeneric — Use System renderer with custom a ConsoleControl protocol which passes its mode set and get operations through to system ConsoleControl when it exists.~~

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

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

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

8. `ConsoleFont`
   Type: `plist string`
   Failsafe: Empty (use OpenCore builtin console font)
   Description: Specify the console font to use for OpenCore `Builtin` text renderer.

   The font file must be located in `EFI/OC/Resources/Font/{font-name}.hex` and must be 8x16 resolution. Various console fonts can be found online in either `.bdf` or `.hex` format. `.bdf` can be converted to `.hex` format using `gbdfed` (available for Linux or macOS).

   There is often no need to change console font, the main use-case being to provide an extended character set for those relatively rare EFI applications which have multi-lingual support (e.g. `memtest86`).

   The OcBinaryData repository includes:

   - Terminus — A font with extensive character support suitable for applications such as the above.
   - TerminusCore — A lightly modified version of the Terminus font, making some glyphs (`@KMRSTVWimrsw`) more similar to the free ISO Latin font used in XNU and OpenCore.

   `Terminus` and `TerminusCore` are provided under the SIL Open Font License, Version 1.1. Some additional GPL licensed fonts from the EPTO Fonts library, converted to the required `.hex` format, can be found here.

   Note 1: On many newer systems the `System` text renderer already provides a full set of international characters, in which case this can be used without requiring the `Builtin` renderer and a custom font.

   Note 2: This option only affects the `Builtin` text renderer and only takes effect from the point at which the `Builtin` renderer is configured. When console output is visible before this point, it is using the system console font.

9. `ConsoleMode`
   Type: `plist string`
   Failsafe: Empty (Maintain current console mode)
   Description: Sets console output mode as specified with the `WxH` (e.g. `80x24`) formatted string.

   Set to `Max` to attempt using the largest available console mode.

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

10. ~~Resolution~~`DirectGopRendering`
    Type: `plist` ~~string~~`boolean`
    Failsafe: ~~Empty (Maintain current screen resolution)~~`false`
    Description: ~~Sets console output screen resolution~~Use builtin graphics output protocol renderer for console.

11. ~~Set to~~ On certain firmware, such as on the ~~WxH@Bpp~~`MacPro5,1` ~~(e.g. 1920x1080@32) or WxH (e.g. 1920x1080) formatted string to request custom resolution from GOP if available.~~

12. ~~Set to Max to attempt using the largest available screen resolution.~~ , 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.

~~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 the section for 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`~~This renderer fully supports `AppleEg2Info` protocol and will provide screen rotation for all EFI applications. In order to provide seamless rotation compatibility with `EfiBoot`, builtin `AppleFramebufferInfo` should also be used, i.e. it may need to be overridden on Mac EFI.

13. `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 and certain ATI GPUs.

14. ~~`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.~~

15. ~~`DirectGopRendering`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Use builtin graphics output protocol renderer for console.~~

    ~~On certain 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.~~

    ~~This renderer fully supports `AppleEg2Info` protocol and will provide screen rotation for all EFI applications. In order to provide seamless rotation compatibility with `EfiBoot`, builtin `AppleFramebufferInfo` should also be used, i.e. it may need to be overridden on Mac EFI.~~

16. `GopBurstMode`
Type: `plist boolean`
Failsafe: `false`
Description: Enable write-combining (WC) caching for GOP memory, if system firmware has not already enabled it.

    Some older firmware (e.g. EFI-era Macs) fails to set write-combining caching (aka burst mode) for GOP memory, even though the CPU supports it. Setting this can give a considerable speed-up for GOP operations, especially on systems which require `DirectGopRendering`.

    Note 1: This quirk takes effect whether or not `DirectGopRendering` is set, and in some cases may give a noticeable speed-up to GOP operations even when `DirectGopRendering` is `false`.

    Note 2: On most systems from circa 2013 onwards, write-combining caching is already applied by the firmware to GOP memory, in which case `GopBurstMode` is unnecessary. On such systems enabling the quirk should normally be harmless, producing an `OCC:` debug log entry indicating that burst mode is already started.

    Note 3: Some caution should be taken when enabling this quirk, as it has been observed to cause hangs on a few systems. Since additional guards have been added to try to prevent this, please log a bugtracker issue if such a system is found.

17. `GopPassThrough`
Type: `plist string`
Failsafe: `Disabled`
Description: Provide GOP protocol instances on top of UGA protocol instances.

    This option provides the GOP protocol via a UGA-based proxy for firmware that do not implement the protocol. The supported values for the option are as follows:

- `Enabled` — provide GOP for all UGA protocols.
- `Apple` — provide GOP for `AppleFramebufferInfo`-enabled protocols.
- `Disabled` — do not provide GOP.

Note: This option requires `ProvideConsoleGop` to be enabled.

18. `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 if console control is not in `Text` mode.

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

19. ~~`ReplaceTabWithSpace`~~`InitialMode`
Type: `plist` ~~`boolean`~~`string`
Failsafe: ~~`false`~~`Auto`
Description: Selects the internal `ConsoleControl` mode in which `TextRenderer` will operate.

Available values are `Auto`, `Text` and `Graphics`. `Text` and `Graphics` specify the named mode. `Auto` uses the current mode of the system `ConsoleControl` protocol when one exists, defaulting to `Text` mode otherwise.

UEFI firmware typically supports `ConsoleControl` with two rendering modes: `Graphics` and `Text`. 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~~provide a native `ConsoleControl` and rendering modes. OpenCore and macOS expect text to only be shown in `Text` mode but graphics to be drawn in any mode, and this is how the OpenCore `Builtin` renderer behaves. Since this is not required by the UEFI specification, behaviour of the system `ConsoleControl` protocol, when it exists, may vary.

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

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

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

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

21. `ReconnectGraphicsOnConnect`
Type: `plist boolean`
Failsafe: `false`
Description: Reconnect all graphics drivers during driver connection.

On certain firmware, it may be desireable to use an alternative graphics driver, for example BiosVideo.efi, providing better screen resolution options on legacy machines, or a driver supporting `ForceResolution`. This option attempts to disconnect all currently connected graphics drivers before connecting newly loaded drivers.

Note: This option requires `ConnectDrivers` to be enabled.

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

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

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

23. `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.

24. `Resolution`
Type: `plist string`
Failsafe: Empty (Maintain current screen resolution)
Description: Sets console output screen resolution.

    - Set to `WxH@Bpp` (e.g. `1920x1080@32`) or `WxH` (e.g. `1920x1080`) formatted string to request custom resolution from GOP if available.
    - Set to `Max` to attempt using the largest available screen resolution.

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

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

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

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

    Currently two renderers are supported: `Builtin` and `System`. The `System` renderer uses firmware services for text rendering, however with additional options provided to sanitize the output. The `Builtin` renderer bypasses firmware services and performs text rendering on its own. Each renderer supports a different set of options. It is recommended to use the `Builtin` renderer, as it supports HiDPI mode and uses full screen resolution.

    Each renderer provides its own `ConsoleControl` protocol (in the case of `SystemGeneric` only, this passes some operations through to the system `ConsoleControl` protocol, if one exists).

    Valid values of this option are combinations of the renderer to use and the `ConsoleControl` mode to set on the underlying system `ConsoleControl` protocol before starting. To control the initial mode of the provided `ConsoleControl` protocol once started, use the `InitialMode` option.

    - `BuiltinGraphics` — Switch to `Graphics` mode then use `Builtin` renderer with custom `ConsoleControl`.
    - `BuiltinText` — Switch to `Text` mode then use `Builtin` renderer with custom `ConsoleControl`.
    - `SystemGraphics` — Switch to `Graphics` mode then use `System` renderer with custom `ConsoleControl`.
    - `SystemText` — Switch to `Text` mode then use `System` renderer with custom `ConsoleControl`.
    - `SystemGeneric` — Use `System` renderer with custom a `ConsoleControl` protocol which passes its mode set and get operations through to system `ConsoleControl` when it exists.

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

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

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

27. `UIScale`
Type: `plist integer`, 8 bit
Failsafe: `-1`
Description: User interface scaling factor.

Corresponds to `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:UIScale` variable.

- `1` — 1x scaling, corresponds to normal displays.
- `2` — 2x scaling, corresponds to HiDPI displays.
- `-1` — leaves the current variable unchanged.
- `0` — automatically chooses scaling based on the current resolution.

Note 1: Automatic scale factor detection works on the basis of total pixel area and may fail on small HiDPI displays, in which case the value may be manually managed using the NVRAM section.

Note 2: When switching from manually specified NVRAM variable to this preference an NVRAM reset may be needed.

28. `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.

11.18 ProtocolOverrides Properties

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

Apple audio protocols allow OpenCore and the macOS bootloader to play sounds and signals for screen reading or audible error reporting. Supported protocols are beep generation and VoiceOver. The VoiceOver protocol is only provided natively by Gibraltar machines (T2), however versions of macOS which support VoiceOver will see and use the implementation provided by OpenCore, on screens such as FileVault 2 unlock. VoiceOver is not supported before macOS High Sierra (10.13). Older macOS versions use the AppleHDA protocol (which is not currently implemented) instead.

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

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

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

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

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

4. `AppleEg2Info`
Type: `plist boolean`
Failsafe: `false`
Description: Replaces the Apple EFI Graphics 2 protocol with a builtin version.

   Note 1: This protocol allows newer `EfiBoot` versions (at least 10.15) to expose screen rotation to macOS. Refer to `ForceDisplayRotationInEFI` variable description on how to set screen rotation angle.

   Note 2: On systems without native support for `ForceDisplayRotationInEFI`, `DirectGopRendering=true` is also required for this setting to have an effect.

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

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

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

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

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

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

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

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

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

11.19 Quirks Properties

1. `ActivateHpetSupport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Activates HPET support.

   Older boards like ICH6 may not always have HPET setting in the firmware preferences, this option tries to force enable it.

2. `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 using UEFI Secure Boot.

3. `EnableVectorAcceleration`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable AVX vector acceleration of SHA-512 and SHA-384 hashing algorithms.

   Note: This option may cause issues on certain laptop firmwares, including Lenovo.

4. `EnableVmx`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable Intel virtual machine extensions.

   Note: Required to allow virtualization in Windows on some Mac hardware. VMX is enabled or disabled and locked by BIOS before OpenCore starts on most firmware. Use BIOS to enable virtualization where possible.

5. ~~`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 using UEFI Secure Boot.~~

6. `ExitBootServicesDelay`
   Type: `plist integer`
   Failsafe: `0`
   Description: Adds delay in microseconds after `EXIT_BOOT_SERVICES` event.

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

7. `ForceOcWriteFlash`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enables writing to flash memory for all OpenCore-managed NVRAM system variables.

   Note: This value should be disabled on most types of firmware but is left configurable to account for firmware that may have issues with volatile variable storage overflows or similar. Boot issues across multiple OSes can be observed on e.g. Lenovo Thinkpad T430 and T530 without this quirk. Apple variables related to Secure Boot and hibernation are exempt from this for security reasons. Furthermore, some OpenCore variables are exempt for different reasons, such as the boot log due to an available user option, and the TSC frequency due to timing issues. When toggling this option, a NVRAM reset may be required to ensure full functionality.

8. `ForgeUefiSupport`
   Type: `plist boolean`

Failsafe: `false`
Description: Implement partial UEFI 2.x support on EFI 1.x firmware.

This setting allows running some software written for UEFI 2.x firmware, such as NVIDIA GOP Option ROMs, on hardware with older EFI 1.x firmware (e.g. `MacPro5,1`).

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

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

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

11. `ReloadOptionRoms`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Query PCI devices and reload their Option ROMs if available.

    For example, this option allows reloading NVIDIA GOP Option ROM on older Macs after the firmware version is upgraded via `ForgeUefiSupport`.

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

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

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

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

13. ~~`ResizeUsePciRbIo`**Type**: `plist boolean`**Failsafe**: `false`**Description**: Use PciRootBridgeIo for~~ `ResizeGpuBars`~~and `ResizeAppleGpuBars`~~

    ~~The quirk makes `ResizeGpuBars` and `ResizeAppleGpuBars` use `PciRootBridgeIo` instead of PciIo. This is needed on systems with a buggy `PciIo` implementation where trying to configure Resizable BAR results in `Capability I/O Error`. Typically this is required on older systems which have been modified with ReBarUEFI.~~

14. ~~`ShimRetainProtocol`~~
    Type: ~~`plist boolean`**Failsafe**: `false`**Description**: Request Linux Shim to keep protocol installed for subsequent image loads.~~

    ~~This option is only required if chaining OpenCore from Shim. It must be set in order to allow OpenCore to launch items which are verified by certificates present in Shim, but not in the system Secure Boot database.~~

15. ~~ResizeGpuBarsType:~~ `plist integer`
Failsafe: `-1`
Description: Configure GPU PCI BAR sizes.

This quirk sets GPU PCI BAR sizes as specified or chooses the largest available below the `ResizeGpuBars` value. The specified value follows PCI Resizable BAR spec. Use `0` for 1 MB, `1` for 2 MB, `2` for 4 MB, and so on up to `19` for 512 GB.

Resizable BAR technology allows to ease PCI device programming by mapping a configurable memory region, BAR, into CPU address space (e.g. VRAM to RAM) as opposed to a fixed memory region. This technology is necessary, because one cannot map the largest memory region by default, for the reasons of backwards compatibility with older hardware not supporting 64-bit BARs. Consequentially devices of the last decade use BARs up to 256 MB by default (4 remaining bits are used by other data) but generally allow resizing them to both smaller and larger powers of two (e.g. from 1 MB up to VRAM size).

Operating systems targeting x86 platforms generally do not control PCI address space, letting UEFI firmware decide on the BAR addresses and sizes. This illicit practice resulted in Resizable BAR technology being unused up until 2020 despite being standardised in 2008 and becoming widely available in the hardware soon after.

Modern UEFI firmware allow the use of Resizable BAR technology but generally restrict the configurable options to failsafe default (`OFF`) and maximum available (`ON`). This quirk allows to fine-tune this value for testing and development purposes.

Consider a GPU with 2 BARs:

- `BAR0` supports sizes from 256 MB to 8 GB. Its value is 4 GB.
- `BAR1` supports sizes from 2 MB to 256 MB. Its value is 256 MB.

Example 1: Setting `ResizeGpuBars` to 1 GB will change `BAR0` to 1 GB and leave `BAR1` unchanged.
Example 2: Setting `ResizeGpuBars` to 1 MB will change `BAR0` to 256 MB and `BAR0` to 2 MB.
Example 3: Setting `ResizeGpuBars` to 16 GB will change `BAR0` to 8 GB and leave `BAR1` unchanged.

Note 1: This quirk shall not be used to workaround macOS limitation to address BARs over 1 GB. `ResizeAppleGpuBars` should be used instead.

Note 2: While this quirk can increase GPU PCI BAR sizes, this will not work on most firmware as is, because the quirk does not relocate BARs in memory, and they will likely overlap. In most cases it is best to either update the firmware to the latest version or customise it with a specialised driver like ReBarUEFI.

16. `ResizeUsePciRbIo`
Type: `plist boolean`
Failsafe: `false`
Description: Use PciRootBridgeIo for `ResizeGpuBars` and `ResizeAppleGpuBars`

The quirk makes `ResizeGpuBars` and `ResizeAppleGpuBars` use `PciRootBridgeIo` instead of PciIo. This is needed on systems with a buggy `PciIo` implementation where trying to configure Resizable BAR results in `Capability I/O Error`. Typically this is required on older systems which have been modified with ReBarUEFI.

17. `ShimRetainProtocol`
Type: `plist boolean`
Failsafe: `false`
Description: Request Linux Shim to keep protocol installed for subsequent image loads.

This option is only required if chaining OpenCore from Shim. It must be set in order to allow OpenCore to launch items which are verified by certificates present in Shim, but not in the system Secure Boot database.

18. `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 extension such as VoodooTSCSync, TSCAdjustReset, or CpuTscSync (a more specialised variant of VoodooTSCSync for newer laptops).

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

19. `UnblockFsConnect`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Some types of firmware block partition handles by opening them in `By Driver` mode, resulting in an inability to install File System protocols.

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

11.20 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. Refer to the `Sample.plist` file for details.

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

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

4. `Size`
   Type: `plist integer`
   Failsafe: `0`
   Description: Size of the reserved memory region, must be page-aligned (4 KBs).

5. `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`

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

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

12.1.1 ~~macOS~~ OS X 10.8 and 10.9

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

12.1.2 ~~macOS~~ Mac OS X 10.7

- All previous issues apply.
- `SSSE3` support (not to be confused with `SSE3` support) is a hard requirement for ~~macOS~~ Mac OS X 10.7 kernel.
- Many kexts, including `Lilu` when 32-bit kernel is used and a lot of `Lilu` plugins, are unsupported on ~~macOS~~Mac OS X 10.7 and older as they require newer kernel APIs, which are not part of the ~~macOS~~Mac OS X 10.7 SDK.
- Prior to ~~macOS~~OS X 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~~ Mac OS X 10.6

- All previous issues apply.
- `SSSE3` support is a requirement for ~~macOS~~ Mac OS X 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~~Mac OS X 10.6 are ~~macOS~~Mac OS X 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~~Mac OS X 10.6 is legally owned. Refer to the `DIGEST.txt` file for details. Note that these are the earliest tested versions of ~~macOS~~Mac OS X 10.6 with OpenCore.

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

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

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

12.1.4 ~~macOS~~ Mac OS X 10.5

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

12.1.5 ~~macOS~~ Mac OS X 10.4

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

12.2 UEFI Secure Boot

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

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