



OpenCore

Reference Manual (0.0.~~3~~.4)

[2019.08.07]

Contents

1	Introduction	2
1.1	Known defects	2
2	Generic Terms	3
1.1	Generic Terms	3
2	Overview Configuration	4
2.1	Configuration Terms	4
2.2	Configuration Processing	4
2.3	Configuration Structure	5
3	Setup	6
3.1	Directory Structure	6
3.2	Installation and Upgrade	7
3.3	Contribution	7
4	ACPI	9
4.1	Introduction	9
4.2	Properties	9
4.3	Add Properties	9
4.6	Quirks Properties	11
5	Booter	13
5.1	Introduction	13
5.2	Properties	13
5.3	Quirks Properties	13
7.4	Block Properties	18
7.5	Emulate Properties	18
7.6	Patch Properties	18
8	Misc	22
8.1	Introduction	22
8.2	Properties	22
8.3	Boot Properties	22
8.5	Security Properties	25
9	NVRAM	28
9.1	Introduction	28
9.2	Properties	28
11	UEFI	39
11.1	Introduction	39
11.2	Properties	39
11.3	Protocols Properties	40
11.4	Quirks Properties	41
12	Troubleshooting	43
12.1	Windows support	43
12.2	Debugging	44
12.3	Tips and Tricks	44

1 Introduction

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

1.1 Known defects

For OpenCore issues please refer to Acidanthera Bugtracker.

2 Generic Terms

1.1 Generic Terms

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

2 Overview Configuration

2.1 Configuration Terms

- **OC config** — OpenCore Configuration file in **plist** format named **config.plist**. It has to provide extensible way to configure OpenCore and is structured to be separated into multiple named sections situated in the root **plist** dictionary. These sections are permitted to have **plist array** or **plist dictionary** types and are described in corresponding sections of this document.
- **valid key** — **plist key** object of **OC config** described in this document or its future revisions. Besides explicitly described **valid keys**, keys starting with **#** symbol (e.g. **#Hello**) are also considered **valid keys** and behave as comments, effectively discarding their value, which is still required to be a **valid plist object**. All other **plist keys** are not valid, and their presence yields to **undefined behaviour**.
- **valid value** — **valid plist object** of **OC config** described in this document that matches all the additional requirements in specific **plist object** description if any.
- **invalid value** — **valid plist object** of **OC config** described in this document that is of other **plist type**, does not conform to additional requirements found in specific **plist object** description (e.g. value range), or missing from the corresponding collection. **Invalid value** is read with or without an error message as any possible value of this **plist object** in an undetermined manner (i.e. the values may not be same across the reboots). Whilst reading an **invalid value** is equivalent to reading certain defined **valid value**, applying incompatible value to the host system may yield to **undefined behaviour**.
- **optional value** — **valid value** of **OC config** described in this document that reads in a certain defined manner provided in specific **plist object** description (instead of **invalid value**) when not present in **OC config**. All other cases of **invalid value** do still apply. Unless explicitly marked as **optional value**, any other value is required to be present and reads to **invalid value** if missing.
- **fatal behaviour** — behaviour leading to boot termination. Implementation must stop the boot process from going any further until next host system boot. It is allowed but not required to perform cold reboot or show any warning message.
- **undefined behaviour** — behaviour not prescribed by this document. Implementation is allowed to take any measures including but not limited to **fatal behaviour**, assuming any states or values, or ignoring, unless these measures negatively affect system security in general.

2.2 Configuration Processing

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

OC config has size, nesting, and key amount limitations. **OC config** size does not exceed 16 MBs. **OC config** has no more than 8 nesting levels. **OC config** has up to 16384 XML nodes (i.e. one **plist dictionary** item is counted as a pair of nodes) within each **plist object**.

Reading malformed **OC config** file leads to **undefined behaviour**. Examples of malformed **OC config** cover at least the following cases:

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

It is recommended but not required to abort loading malformed **OC config** and continue as if no **OC config** was present. For forward compatibility it is recommended but not required for the implementation to warn about the use of **invalid values**. Recommended practice of interpreting **invalid values** is to conform to the following convention where applicable:

Type	Value
plist string	Empty string (<string></string>)
plist data	Empty data (<data></data>)

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

2.3 Configuration Structure

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

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

Root configuration entries consist of the following:

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

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

3 [Setup](#)

3.1 Directory Structure



[Figure 1. Directory Structure](#)

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

- `BOOTx64.efi`
Initial booter, which loads `OpenCore.efi` unless it was already started as a driver.
- `ACPI`
Directory used for storing supplemental ACPI information for `ACPI` section.
- `Drivers`
Directory used for storing supplemental UEFI drivers for `UEFI` section.
- `Kexts`
Directory used for storing supplemental kernel information for `Kernel` section.
- `Tools`
Directory used for storing supplemental tools.
- `OpenCore.efi`

Main booter driver responsible for operating system loading.

- `vault.plist`
Hashes for all files potentially loadable by OC Config.
- `config.plist`
OC Config.
- `vault.sig`
Signature for `vault.plist`.
- `nvr.plist`
OpenCore variable import file.
- `opencore-opencore-YYYY-MM-DD-HHMMSS.log.txt`
OpenCore log file.

~~Figure 1. Directory Structure~~

3.2 Installation and Upgrade

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

OC config, just like any property lists can be edited with any stock textual editor (e.g. nano, vim), but specialised software may provide better experience. On macOS the preferred GUI application is Xcode. For a lightweight cross-platform and open-source alternative ProperTree editor can be utilised.

For BIOS booting a third-party UEFI environment provider will have to be used. **DuetPkg** is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system you can install **DuetPkg** with a dedicated tool: **BootInstall**.

For upgrade purposes refer to **Differences.pdf** document, providing the information about the changes affecting the configuration compared to the previous release, and **Changelog.md** document, containing the list of modifications across all published updates.

3.3 Contribution

OpenCore can be compiled as an ordinary EDK II. Since UDK development was abandoned by TianoCore, OpenCore requires the use of EDK II Stable. Currently supported EDK II release (potentially with patches enhancing the experience) is hosted in acidanthera/audk.

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

Required external package dependencies include EfiPkg, MacInfoPkg, and OcSupportPkg.

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

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

Listing 1: Compilation Commands

~~NOOPT or DEBUG build modes instead of RELEASE can produce a lot more debug output. With NOOPT source-level debugging with GDB or IDA Pro is also available. For GDB check page. For IDA Pro you will need IDA Pro 7.3 or newer.~~

4 ACPI

4.1 Introduction

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

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

4.2 Properties

1. Add

Type: plist array

Failsafe: Empty

Description: Load selected tables from OC/ACPI directory.

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

2. Block

Type: plist array

Failsafe: Empty

Description: Remove selected tables from ACPI stack.

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

3. Patch

Type: plist array

Failsafe: Empty

Description: Perform binary patches in ACPI tables before table addition or removal.

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

4. Quirks

Type: plist dict

Description: Apply individual ACPI quirks described in Quirks Properties section below.

4.3 Add Properties

1. Comment

Type: plist string

Failsafe: Empty string

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

2. Enabled

Type: plist boolean

Failsafe: false

Description: This ACPI table will not be added unless set to `true`.

3. Path

Type: plist string

Failsafe: Empty string

Description: File paths meant to be loaded as ACPI tables. Example values include `DSDT.aml`, `SubDir/SSDT-8.aml`, `SSDT-USBX.aml`, etc.

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

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

6. Mask
Type: plist data
Failsafe: Empty data
Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Can be set to empty data to be ignored. Must equal to **Replace** in size otherwise.
7. OemTableId
Type: plist data, 8 bytes
Failsafe: All zero
Description: Match table OEM ID to be equal to this value unless all zero.
8. Replace
Type: plist data
Failsafe: Empty data
Description: Replacement data of one or more bytes.
9. ReplaceMask
Type: plist data
Failsafe: Empty data
Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Can be set to empty data to be ignored. Must equal to **Replace** in size otherwise.
10. Skip
Type: plist integer
Failsafe: 0
Description: Number of found occurrences to be skipped before replacement is done.
11. TableLength
Type: plist integer
Failsafe: 0
Description: Match table size to be equal to this value unless 0.
12. TableSignature
Type: textttplist data, 4 bytes
Failsafe: All zero
Description: Match table signature to be equal to this value unless all zero.

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

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

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

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

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

4.6 Quirks Properties

1. FadtEnableReset
Type: plist boolean

Failsafe: false

Description: Provide reset register and flag in FADT table to enable reboot and shutdown on legacy hardware. Not recommended unless required.

2. NormalizeHeaders

Type: plist boolean

Failsafe: false

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

3. RebaseRegions

Type: plist boolean

Failsafe: false

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

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

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

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

4. ResetHwSig

Type: plist boolean

Failsafe: false

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

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

5. ResetLogoStatus

Type: plist boolean

Failsafe: false

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

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

5 Booter

5.1 Introduction

This section allows to apply different kinds of UEFI modifications on Apple bootloader (`boot.efi`). The modifications currently provide various patches and environment alterations for different firmwares. Some of these features were originally implemented as a part of `AptioMemoryFix.efi`, which is no longer maintained. See [Tips and Tricks](#) section for migration steps.

5.2 Properties

1. Quirks

Type: `plist dict`

Description: Apply individual booter quirks described in [Quirks Properties](#) section below.

5.3 Quirks Properties

1. 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 many firmwares using SMM backing for select services like variable storage. SMM may try to access physical addresses, but they get moved by `boot.efi`.

Note: Most but Apple and VMware firmwares need this quirk.

2. DisableVariableWrite

Type: `plist boolean`

Failsafe: `false`

Description: Protect from macOS NVRAM write access.

This is a security option allowing one to restrict NVRAM access in macOS. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `FwRuntimeServices.efi`.

Note: This quirk can also be used as an ugly workaround to buggy UEFI runtime services implementations that fail to write variables to NVRAM and break the rest of the operating system.

3. DiscardHibernateMap

Type: `plist boolean`

Failsafe: `false`

Description: Reuse original hibernate memory map.

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

Note: This may be used to workaround buggy memory maps on older hardware, and is now considered rare legacy. Do not use this unless you fully understand the consequences.

4. EnableSafeModeSlide

Type: `plist boolean`

Failsafe: `false`

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

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

Note: The necessity of this quirk is determined by safe mode availability. If booting to safe mode fails, this option can be tried to be enabled.

5. EnableWriteUnprotector
Type: plist boolean
Failsafe: false
Description: Permit write access to UEFI runtime services code.
This option bypasses R_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 FwRuntimeServices.efi.
Note: The necessity of this quirk is determined by early boot crashes of the firmware.
6. ForceExitBootServices
Type: plist boolean
Failsafe: false
Description: Retry ExitBootServices with new memory map on failure.
Try to ensure that ExitBootServices call succeeds even with outdated MemoryMap key argument by obtaining current memory map and retrying ExitBootServices call.
Note: The necessity of this quirk is determined by early boot crashes of the firmware. Do not use this unless you fully understand the consequences.
7. ProtectCsmRegion
Type: plist boolean
Failsafe: false
Description: Protect CSM region areas from relocation.
Ensure that CSM memory regions are marked as ACPI NVS to prevent boot.efi or XNU from relocating or using them.
Note: The necessity of this quirk is determined by artifacts and sleep wake issues. As AvoidRuntimeDefrag resolves a similar problem, no known firmwares should need this quirk. Do not use this unless you fully understand the consequences.
8. ProvideCustomSlide
Type: plist boolean
Failsafe: false
Description: Provide custom KASLR slide on low memory.
This option performs memory map analysis of your firmware and checks whether all slides (from 1 to 255) can be used. As boot.efi generates this value randomly with rdrand or pseudo randomly rdtsc, there is a chance of boot failure when it chooses a conflicting slide. In case potential conflicts exist, this option forces macOS to use a pseudo random value among the available ones. This also ensures that slide= argument is never passed to the operating system for security reasons.
Note: The necessity of this quirk is determined by OCABC: Only N/256 slide values are usable! message in the debug log. If the message is present, this option is to be enabled.
9. SetupVirtualMap
Type: plist boolean
Failsafe: false
Description: Setup virtual memory at SetVirtualAddresses.
Select firmwares access memory by virtual addresses after SetVirtualAddresses call, which results in early boot crashes. This quirk workarounds the problem by performing early boot identity mapping of assigned virtual addresses to physical memory.
Note: The necessity of this quirk is determined by early boot failures.
10. ShrinkMemoryMap
Type: plist boolean
Failsafe: false
Description: Attempt to join similar memory map entries.

Select firmwares have very large memory maps, which do not fit Apple kernel, permitting up to 64 slots for runtime memory. This quirk attempts to unify contiguous slots of similar types to prevent boot failures.

Note: The necessity of this quirk is determined by early boot failures. Do not use this unless you fully understand the consequences.

Failsafe: Empty string

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

5. `MatchKernel`

Type: `plist string`

Failsafe: Empty string

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

6. `PlistPath`

Type: `plist string`

Failsafe: Empty string

Description: Kext `Info.plist` path relative to bundle (e.g. `Contents/Info.plist`).

7.4 Block Properties

1. `Comment`

Type: `plist string`

Failsafe: Empty string

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

2. `Enabled`

Type: `plist boolean`

Failsafe: `false`

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

3. `Identifier`

Type: `plist string`

Failsafe: Empty string

Description: Kext bundle identifier (e.g. `com.apple.driver.AppleTyMCEDriver`).

4. `MatchKernel`

Type: `plist string`

Failsafe: Empty string

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

7.5 Emulate Properties

1. `Cpuid1Data`

Type: `plist data`, 16 bytes

Failsafe: All zero

Description: Sequence of `EAX`, `EBX`, `ECX`, `EDX` values in Little Endian order to replace `CPUID` (1) call in XNU kernel.

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](#).

7.6 Patch Properties

1. `Base`

Type: `plist string`

Failsafe: Empty string

Description: Selects symbol-matched base for patch lookup (or immediate replacement) by obtaining the address of provided symbol name. Can be set to empty string to be ignored.

2. `Comment`

Type: `plist string`

Description: Disables `PKG_CST_CONFIG_CONTROL` (0xE2) MSR modification in `AppleIntelCPUPowerManagement.kext`, commonly causing early kernel panic, when it is locked from writing.

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

2. `AppleXcpmCfgLock`

Type: `plist boolean`

Failsafe: `false`

Description: Disables `PKG_CST_CONFIG_CONTROL` (0xE2) MSR modification in XNU kernel, commonly causing early kernel panic, when it is locked from writing (XCPM power management).

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

3. `AppleXcpmExtraMsrs`

Type: `plist boolean`

Failsafe: `false`

Description: Disables multiple MSR access critical for select CPUs, which have no native XCPM support.

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

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

4. `CustomSMBIOSGuid`

Type: `plist boolean`

Failsafe: `false`

Description: Performs GUID patching for `UpdateSMBIOSMode Custom` mode. Usually relevant for Dell laptops.

5. `DisableIoMapper`

Type: `plist boolean`

Failsafe: `false`

Description: Disables `IoMapper` support in XNU (VT-d), which may conflict with the firmware implementation.

Note: This option is a preferred alternative to dropping `DMAR` ACPI table and disabling VT-d in firmware preferences, which does not break VT-d support in other systems in case they need it.

6. `ExternalDiskIcons`

Type: `plist boolean`

Failsafe: `false`

Description: Apply icon type patches to `AppleAHCIPort.kext` to force internal disk icons for all AHCI disks.

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

7. `LapicKernelPanic`

Type: `plist boolean`

Failsafe: `false`

Description: Disables kernel panic on LAPIC interrupts.

8. `PanicNoKextDump`

Type: `plist boolean`

Failsafe: `false`

Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

9. `ThirdPartyTrim`

Type: `plist boolean`

Failsafe: `false`

Description: Patch `IOAHCIBlockStorage.kext` to force TRIM command support on AHCI SSDs.

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

8 Misc

8.1 Introduction

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

8.2 Properties

1. Boot
Type: plist dict
Description: Apply boot configuration described in Boot Properties section below.
2. Debug
Type: plist dict
Description: Apply debug configuration described in Debug Properties section below.
3. Security
Type: plist dict
Description: Apply security configuration described in Security Properties section below.
4. Tools
Type: plist array
Description: Add new entries to boot picker.

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

Note: Select tools, for example, ~~UEFI Shell or NVRAM cleaning~~ UEFI Shell [or](#) CleanNvram are very dangerous and **MUST NOT** appear in production configurations, especially in vaulted ones and protected with secure boot, as they may be used to easily bypass secure boot chain.

8.3 Boot Properties

1. ConsoleMode
Type: plist string
Failsafe: Empty string
Description: Sets console output mode as specified with the WxH (e.g. 80x24) formatted string. Set to empty string not to change console mode. Set to `Max` to try to use largest available console mode.
2. ConsoleBehaviourOs
Type: plist string
Failsafe: Empty string
Description: Set console control behaviour upon operating system load.

Console control is a legacy protocol used for switching between text and graphics screen output. Some firmwares do not provide it, yet select operating systems require its presence, which is what `ConsoleControl` UEFI protocol is for.

When console control is available, OpenCore can be made console control aware, and and set different modes for the operating system booter (`ConsoleBehaviourOs`), which normally runs in graphics mode, and its own user interface (`ConsoleBehaviourUi`), which normally runs in text mode. Possible behaviours, set as values of these options, include:

- Empty string — Do not modify console control mode.
- `Text` — Switch to text mode.
- `Graphics` — Switch to graphics mode.
- `ForceText` — Switch to text mode and preserve it (requires `ConsoleControl`).
- `ForceGraphics` — Switch to graphics mode and preserve it (require `ConsoleControl`).

Hints:

- Unless empty works, firstly try to set `ConsoleBehaviourOs` to `Graphics` and `ConsoleBehaviourUi` to `Text`.
- On APTIO IV (Haswell and earlier) it is usually enough to have `ConsoleBehaviourOs` set to `Graphics` and `ConsoleBehaviourUi` set to `ForceText` to avoid visual glitches.

- On APTIO V (Broadwell and newer) `ConsoleBehaviourOs` set to `ForceGraphics` and `ConsoleBehaviourUi` set to `ForceText` usually works best.
- On Apple firmwares `ConsoleBehaviourOs` set to `Graphics` and `ConsoleBehaviourUi` set to `Text` is supposed to work best.

Note: `IgnoreTextInGraphics` and `SanitiseClearScreen` may need to be enabled for select firmware implementations. [Particularly APTIO firmwares.](#)

3. `ConsoleBehaviourUi`

Type: `plist string`

Failsafe: Empty string

Description: Set console control behaviour upon OpenCore user interface load. Refer to `ConsoleBehaviourOs` description for details.

4. `HibernateMode`

Type: `plist string`

Failsafe: `None`

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

- `None` — Avoid hibernation for your own good.
- `Auto` — Use RTC and NVRAM detection.
- `RTC` — Use RTC detection.
- `NVRAM` — Use NVRAM detection.

5. `HideSelf`

Type: `plist boolean`

Failsafe: `false`

Description: Hides own boot entry from boot picker. This may potentially hide other entries, for instance, when another UEFI OS is installed on the same volume and driver boot is used.

6. `Resolution`

Type: `plist string`

Failsafe: Empty string

Description: Sets console output screen resolution.

- Set to `WxH@Bpp` (e.g. `1920x1080@32`) `WxH` (e.g. `1920x1080`) formatted string to request custom resolution from GOP if available.
- Set to empty string not to change screen resolution.
- Set to `Max` to try to use 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 FileVault 2 UEFI password interface and boot screen logo. Refer to Recommended Variables section for more details.

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

7. `ShowPicker`

Type: `plist boolean`

Failsafe: `false`

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

8. `Timeout`

Type: `plist integer, 32 bit`

Failsafe: `0`

Description: Timeout in seconds in boot picker before automatic booting of the default boot entry.

9. `UsePicker`

Type: `plist boolean`

Failsafe: `false`

Description: Use OpenCore built-in boot picker for boot management.

`UsePicker` set to `false` entirely disables all boot management in OpenCore except policy enforcement. In this case a custom user interface may utilise `OcSupportPkg OcBootManagementLib` to implement a user friendly boot

Warning: Some firmwares are reported to have broken NVRAM garbage collection. This means that they may not be able to always free space after variable deletion. Do not use non-volatile NVRAM logging without extra need on such devices.

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

File logging will create a file named `opencoreopencore-YYYY-MM-DD-HHMMSS.log.txt` at EFI volume root with log contents ([the upper case letter sequence is replaced with date and time from the firmware](#)). Please be warned that some file system drivers present in firmwares are not reliable, and may corrupt data when writing files through UEFI. Log is attempted to be written in the safest manner, and thus is very slow. Ensure that `DisableWatchDog` is set to `true` when you use a slow drive.

8.5 Security Properties

1. ExposeSensitiveData

Type: plist integer

Failsafe: 2

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

- 0x01 — Expose printable booter path as an UEFI variable.
- 0x02 — Expose OpenCore version as an UEFI variable.

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

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

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

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

To obtain OpenCore version use the following command in macOS:

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

2. HaltLevel

Type: plist integer, 64 bit

Failsafe: 0x80000000 (DEBUG_ERROR)

Description: EDK II debug level bitmask (sum) causing CPU to halt (stop execution) after obtaining a message of `HaltLevel`. Possible values match `DisplayLevel` values.

3. RequireSignature

Type: plist boolean

Failsafe: true

Description: Require `vault.sig` signature file for `vault.plist` in OC directory.

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

To embed the public key you should do either of the following:

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

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

Note: `vault.sig` is used regardless of this option when public key is embedded into `OpenCore.efi`. Setting it to `true` will only ensure configuration sanity, and abort the boot process when public key is not set but was supposed to be used for verification.

9 NVRAM

9.1 Introduction

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

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

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

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

9.2 Properties

1. Add

Type: `plist dict`

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

Created variables get `EFI_VARIABLE_BOOTSERVICE_ACCESS` and `EFI_VARIABLE_RUNTIME_ACCESS` attributes set. Variables will only be set if not present and not blocked. To overwrite a variable add it to `Block` section. This approach enables to provide default values till the operating system takes the lead.

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

2. Block

Type: `plist dict`

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

3. LegacyEnable

Type: `plist boolean`

Failsafe: `false`

Description: Enables loading of NVRAM variable file named `nvram.plist` from EFI volume root.

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

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

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

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

4. LegacySchema

Type: `plist dict`

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

You can use `*` value to accept all variables for select GUID.

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

11 UEFI

11.1 Introduction

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

11.2 Properties

1. ConnectDrivers

Type: plist boolean

Failsafe: false

Description: Perform UEFI controller connection after driver loading. This option is useful for loading filesystem drivers, which usually follow UEFI driver model, and may not start by themselves. While effective, this option is not necessary with e.g. APFS loader driver, and may slightly slowdown the boot.

2. Drivers

Type: plist array

Failsafe: None

Description: Load selected drivers from `OC/Drivers` directory.

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

- **ApfsDriverLoader** — APFS file system bootstrap driver adding the support of embedded APFS drivers in bootable APFS containers in UEFI firmwares.
- **AppleUiSupport** — Apple-specific user interface support driver. This driver brings the support for FileVault 2 GUI, hotkey parsing (shift, cmd+v, etc.), language collation support, and certain other features important for normal macOS functioning. For hotkey support **AppleKeyMapAggregator**-compatible driver is required.
- **AppleGenericInput** — user input driver adding the support of **AppleKeyMapAggregator** protocols on top of different UEFI input protocols. Additionally resolves mouse input issues on select firmwares. This is an alternative to **UsbKbDxe**, which may work better or worse depending on the firmware.
- **FwRuntimeServices** — ~~a set of quirks for various firmwares. While it primarily targets APTIO firmwares, other firmwares may be compatible as well.~~ Among the resolved issues are hibernation support, KASLR, Lilu NVRAM security enhancements, NVRAM, and UEFI Boot entry preservation. OC FIRMWARE RUNTIME protocol implementation that increases the security of OpenCore and Lilu by supporting read-only and write-only NVRAM variables. Some quirks, like RequestBootVarRouting, require this driver for proper function. Due to the nature of being a runtime driver, i.e. functioning in parallel with the target operating system, it cannot be implemented within OpenCore itself.
- **EmuVariableRuntimeDxe** — NVRAM emulation driver from **MdeModulePkg**. NVRAM is supported by most modern firmwares. For firmwares with macOS incompatible NVRAM implementation an emulated driver may be used. This driver will not preserve NVRAM contents across the reboots.
- **EnglishDxe** — Unicode collation driver from **MdeModulePkg**. This driver is a lightweight alternative to **AppleUiSupport**, which contains no Apple-specific code, and only provides unicode collation support. The driver is not recommended for use on any hardware but few original Macs.
- **EnhancedFatDxe** — FAT filesystem driver from **FatPkg**. This driver is embedded in all UEFI firmwares, and cannot be used from OpenCore. It is known that multiple firmwares have a bug in their FAT support implementation, which leads to corrupted filesystems on write attempt. Embedding this driver within the firmware may be required in case writing to EFI partition is needed during the boot process.
- **NvmExpressDxe** — NVMe support driver from **MdeModulePkg**. This driver is included in most firmwares starting with Broadwell generation. For Haswell and earlier embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
- **UsbKbDxe** — USB keyboard driver adding the support of **AppleKeyMapAggregator** protocols on top of a custom USB keyboard driver implementation. This is an alternative to **AptioInputFix**, which may work better or worse depending on the firmware.
- **VirtualSmc** — UEFI SMC driver, required for proper FileVault 2 functionality and potentially other macOS

specifics. An alternative, named `SMCHelper`, is not compatible with `VirtualSmc` and `OpenCore`, which is unaware of its specific interfaces. In case `FakeSMC` kernel extension is used, manual NVRAM variable addition may be needed and `VirtualSmc` driver should still be used.

- `VBoxHfs` — HFS file system driver with bless support. This driver is an alternative to a closed source `HFSPplus` driver commonly found in Apple firmwares. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit.
- `XhciDxe` — XHCI USB controller support driver from `MdeModulePkg`. This driver is included in most firmwares starting with Sandy Bridge generation. For earlier firmwares or legacy systems it may be used to support external USB 3.0 PCI cards.

To compile the drivers from TianoCore UDK use the same command you do normally use for OpenCore compilation, but choose a corresponding package:

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

3. Protocols

Type: plist dict

Failsafe: None

Description: Force builtin versions of select protocols described in Protocols Properties section below.

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

4. Quirks

Type: plist dict

Failsafe: None

Description: Apply individual firmware quirks described in Quirks Properties section below.

11.3 Protocols Properties

1. AppleBootPolicy

Type: plist boolean

Failsafe: false

Description: Reinstalls Apple Boot Policy protocol with a builtin version. This may be used to ensure APFS compatibility on VMs or legacy Macs.

2. ConsoleControl

Type: plist boolean

Failsafe: false

Description: Replaces Console Control protocol with a builtin version.

macOS bootloader requires console control protocol for text output, which some firmwares miss. This option is required to be set when the protocol is already available in the firmware, and other console control options are used, such as `IgnoreTextInGraphics`, `SanitiseClearScreen`, and sometimes `ConsoleBehaviourOs` with `ConsoleBehaviourUi`).

3. DataHub

Type: plist boolean

Failsafe: false

Description: Reinstalls Data Hub protocol with a builtin version. This will drop all previous properties if the protocol was already installed.

4. DeviceProperties

Type: plist boolean

Failsafe: false

Description: Reinstalls Device Property protocol with a builtin version. This will drop all previous properties if it was already installed. This may be used to ensure full compatibility on VMs or legacy Macs.

11.4 Quirks Properties

1. AvoidHighAlloc

Type: plist boolean

Failsafe: false

Description: Advises allocators to avoid allocations above first 4 GBs of RAM.

This is a workaround for select board firmwares, namely GA-Z77P-D3 (rev. 1.1), failing to properly access higher memory in UEFI Boot Services. Not recommended unless required for booting. May cause recovery boot failures on unaffected boards.

2. ExitBootServicesDelay

Type: plist integer

Failsafe: 0

Description: Adds delay in microseconds after EXIT_BOOT_SERVICES event.

This is a very ugly quirk to circumvent "Still waiting for root device" message on select APTIO IV firmwares, namely ASUS Z87-Pro, when using FileVault 2 in particular. It seems that for some reason they execute code in parallel to EXIT_BOOT_SERVICES, which results in SATA controller being inaccessible from macOS. A better approach should be found in some future. Expect 3-5 seconds to be enough in case the quirk is needed.

3. IgnoreInvalidFlexRatio

Type: plist boolean

Failsafe: false

Description: Select firmwares, namely APTIO IV, may contain invalid values in MSR_FLEX_RATIO (0x194) MSR register. These values may cause macOS boot failure on Intel platforms.

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

4. IgnoreTextInGraphics

Type: plist boolean

Failsafe: false

Description: Select firmwares output text onscreen in both graphics and text mode. This is normally unexpected, because random text may appear over graphical images and cause UI corruption. Setting this option to **true** will discard all text output when console control is in mode different from **Text**.

Note: While the option is not supposed to induce harm on unaffected firmwares, its usage is not recommended when it is not required. This option may hide onscreen error messages. **ConsoleControl** may need to be set to **true** for this to work.

5. ProvideConsoleGop

Type: plist boolean

Failsafe: false

Description: macOS bootloader requires GOP (Graphics Output Protocol) to be present on console handle. This option will install it if missing.

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

6. ReleaseUsbOwnership

Type: plist boolean

Failsafe: false

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

7. RequestBootVarRouting

Type: plist boolean

Failsafe: false

Description: Request ~~NVRAM driver (or AptioMemoryFix)~~ to redirectBoot prefixed variables from EFI_GLOBAL_VARIABLE_GUID to OC_VENDOR_VARIABLE_GUID.

12 Troubleshooting

12.1 Windows support

Can I install Windows?

While no official Windows support is provided, 64-bit UEFI Windows installations (Windows 8 and above) prepared with Boot Camp are supposed to work. Third-party UEFI installations as well as systems partially supporting UEFI boot, like Windows 7, might work with some extra precautions. Things to keep in mind:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- Installing Windows and macOS on the same drive is currently unsupported but will be addressed later.
- All the modifications applied (to ACPI, NVRAM, SMBIOS, etc.) are supposed to be operating system agnostic, i.e. apply equally regardless of the OS booted. This enables Boot Camp software experience on Windows.
- macOS requires the first partition to be EFI System Partition, and does not support the default Windows layout. While OpenCore does have a workaround for this, it is highly recommend not to rely on it and install properly.
- Windows may need to be reactivated. To avoid it consider leaving SystemUUID field empty, so that the original firmware UUID is used. Be warned, on old firmwares it may be invalid, i.e. not random. In case you still have issues, consider using HWID or KMS38 license. The nuances of Windows activation are out of the scope of this document and can be found online.

What additional software do I need?

To enable operating system switching and install relevant drivers in the majority of cases you will need Windows support software from Boot Camp. For simplicity of the download process or when configuring an already installed Windows version a third-party utility, Brigadier, can be used successfully. Note, that you may have to download and install 7-Zip prior to using Brigadier.

Remember to always use the latest version of Windows support software from Boot Camp, as versions prior to 6.1 do not support APFS, and thus will not function correctly. To download newest software pass most recent Mac model to Brigadier, for example `./brigadier.exe -m iMac19,1`. To install Boot Camp on an unsupported Mac model afterwards run PowerShell as Administrator and enter `msiexec /i BootCamp.msi`. In case you already have a previous version of Boot Camp installed you will have to remove it first by running `msiexec /x BootCamp.msi` command. `BootCamp.msi` file is located in `BootCamp/Drivers/Apple` directory and can be reached through Windows Explorer.

While Windows support software from Boot Camp solves most of compatibility problems, sometimes you may have to address some of them manually:

- To invert mouse wheel scroll direction `FlipFlopWheel` must be set to 1 as explained on SuperUser.
- `RealTimeIsUniversal` must be set to 1 to avoid time desync between Windows and macOS as explained on SuperUser (this one is usually not needed).
- To access Apple filesystems like HFS and APFS separate software may need to be installed. Some of the known tools are: Apple HFS+ driver (hack for Windows 10), HFSExplorer, MacDrive, Paragon APFS, Paragon HFS+, TransMac, etc. Remember to never ever attempt to modify Apple file systems from Windows as this often leads to irrecoverable data loss.

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

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

```
PS C:\gdisk> .\gdisk64.exe \\.\\physicaldrive0
GPT fdisk (gdisk) version 1.0.4
```

```
Command (? for help): p
Disk \\.\\physicaldrive0: 419430400 sectors, 200.0 GiB
Sector size (logical): 512 bytes
Disk identifier (GUID): DEC57EB1-B3B5-49B2-95F5-3B8C4D3E4E12
Partition table holds up to 128 entries
```


Main partition table begins at sector 2 and ends at sector 33
 First usable sector is 34, last usable sector is 419430366
 Partitions will be aligned on 2048-sector boundaries
 Total free space is 4029 sectors (2.0 MiB)

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

Command (? for help): c
 Partition number (1-4): 4
 Enter name: BOOTCAMP

Command (? for help): w

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

Do you want to proceed? (Y/N): Y
 OK; writing new GUID partition table (GPT) to \\.\physicaldrive0.
 Disk synchronization succeeded! The computer should now use the new partition table.
 The operation has completed successfully.

Listing 3: Relabeling Windows volume

How to choose Windows BOOTCAMP with custom NTFS drivers?

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

12.2 Debugging

Similar to other projects working with hardware OpenCore supports auditing and debugging. The use of NOOPT or DEBUG build modes instead of RELEASE can produce a lot more debug output. With NOOPT source level debugging with GDB or IDA Pro is also available. For GDB check OcSupport Debug page. For IDA Pro you will need IDA Pro 7.3 or newer, refer to Debugging the XNU Kernel with IDA Pro for more details.

To obtain the log during boot you can make the use of serial port debugging. Serial port debugging is enabled in Target, e.g. 0xB for onscreen with serial. OpenCore uses 115200 baud rate, 8 data bits, no parity, and 1 stop bit. For macOS your best choice are CP2102-based UART devices. Connect motherboard TX to USB UART GND, and motherboard GND to USB UART RX. Use screen utility to get the output, or download GUI software, such as CoolTerm.

Remember to enable COM port in firmware settings, and never use USB cables longer than 1 meter to avoid output corruption. To additionally enable XNU kernel serial output you will need debug=0x8 boot argument.

12.3 Tips and Tricks

1. How to debug boot failure?

Normally it is enough to obtain the actual error message. For this ensure that:

- You have a DEBUG or NOOPT version of OpenCore.
- Logging is enabled (1) and shown onscreen (2): Misc → Debug → Target = 3.
- Logged messages from at least DEBUG_ERROR (0x80000000), DEBUG_WARN (0x00000002), and DEBUG_INFO (0x00000040) levels are visible onscreen: Misc → Debug → DisplayLevel = 0x80000042.
- Critical error messages, like DEBUG_ERROR, stop booting: Misc → Security → HaltLevel = 0x80000000.

- Watch Dog is disabled to prevent automatic reboot: `UefiMisc` → `QuirksDebug` → `DisableWatchDog = true`.
- Boot Picker (entry selector) is enabled: `Misc` → `Boot` → `ShowPicker = true`.

If there is no obvious error, check the available hacks in `Quirks` sections one by one.

2. How to customise boot entries?

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

3. What is the simplest way to install macOS?

Copy online recovery image (`*.dmg` and `*.chunklist` files) to `com.apple.recovery.boot` directory on a FAT32 partition with OpenCore. Load OpenCore Boot Picker and choose the entry, it will have a `(dmg)` suffix. Custom name may be created by providing `.contentDetails` file.

To download recovery online you may use Recovery tool from `OcSupportPkg`.

4. Why do online recovery images (*.dmg fail to load?

This may be caused by missing HFS+ driver, as all presently known recovery volumes have HFS+ filesystem. Another cause may be buggy firmware allocator, which can be worked around with `AvoidHighAlloc` UEFI quirk.

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

Sure, most relatively modern Mac models including `MacPro5,1` and virtual machines are fully supported. Even though there are little to none specific details relevant to Mac hardware, some ongoing instructions can be found in [acidanthera/bugtracker#377](#).

6. Why do Find&Replace patches must equal in length?

For machine code (x86 code) it is not possible to do such replacements due to relative addressing. For ACPI code this is risky, and is technically equivalent to ACPI table replacement, thus not implemented. More detailed explanation can be found on [AppleLife.ru](#).