



OpenCore

Reference Manual (~~0.7.9~~0.8.0)

[2022.04.16]

The recommended approach to interpreting `invalid` values is to conform to the following convention where applicable:

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

2.3 Configuration Structure

The `OC config` file is separated into subsections, as described in separate sections of this document, and is designed so as to attempt not to enable anything by default as well as to provide kill switches via an `Enable` property for `plist dict` entries that represent optional plugins and similar.

The file is structured to group related elements in subsections as follows:

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

Root configuration entries consist of the following:

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

Basic validation of an `OC config` file is possible using the `ocvalidate` utility. Please note that the version of `ocvalidate` used must match the OpenCore release and that ~~notwithstanding~~ notwithstanding this, it may not detect all configuration issues present in an `OC config` file.

Note: To maintain system integrity, properties typically have predefined values even when such predefined values are not specified in the `OC config` file. However, all properties must be explicitly specified in the `OC config` file and this behaviour should not be relied on.

2. **AppleXcpmCfgLock**
Type: plist boolean
Failsafe: false
Requirement: 10.8 (not required for older)
Description: Disables PKG_CST_CONFIG_CONTROL (0xE2) MSR modification in XNU kernel, commonly causing early kernel panic, when it is locked from writing (XCPM power management).
Note: This option should be avoided whenever possible. Refer to the `AppleCpuPmCfgLock` description for details.
3. **AppleXcpmExtraMsrs**
Type: plist boolean
Failsafe: false
Requirement: 10.8 (not required for older)
Description: Disables multiple MSR access critical for certain CPUs, which have no native XCPM support.
This is typically used in conjunction with the `Emulate` section on Haswell-E, Broadwell-E, Skylake-SP, and similar CPUs. More details on the XCPM patches are outlined in `acidanthera/bugtracker#365`.
Note: Additional not provided patches will be required for Ivy Bridge or Pentium CPUs. It is recommended to use `AppleIntelCpuPowerManagement.kext` for the former.
4. **AppleXcpmForceBoost**
Type: plist boolean
Failsafe: false
Requirement: 10.8 (not required for older)
Description: Forces maximum performance in XCPM mode.
This patch writes 0xFF00 to MSR_IA32_PERF_CONTROL (0x199), effectively setting maximum multiplier for all the time.
Note: While this may increase the performance, this patch is strongly discouraged on all systems but those explicitly dedicated to scientific or media calculations. Only certain Xeon models typically benefit from the patch.
5. **CustomPciSerialDevice**
Type: plist boolean
Failsafe: false
Requirement: 10.7
Description: Performs change of PMIO register base address on a customised PCI serial device.
The patch changes the PMIO register base address that the XNU kernel uses for serial input and output, from that of the default built-in COM1 serial port 0x3F8, to the base address stored in the first IO BAR of a specified PCI device or to a specific base address (e.g. 0x2F8 for COM2).
Note: By default, serial logging is disabled. `serial=3` boot argument, which enables serial input and output, should be used for XNU to print logs to the serial port.
Note 2: In addition to this patch, `kext Apple16X50PCI0` should be prevented from attaching to have `kprintf` method working properly. This can be achieved by setting (i.e. `Delete`, then `Add`) the `class-code` property of the PCI serial port device to `FFFFFFFF` in `DeviceProperties` section. As an alternative solution, a codeless `kext PCISerialDisable.kext` shown in the spoiler `PCISerialDisable.kext/Contents/Info.plist` at `acidanthera/bugtracker` may also be used.
Note 3: For this patch to be correctly applied, `Override` must be enabled with all keys properly set in `Custom`, under section `Misc->Serial`.
Note 4: This patch is for PMIO support and is therefore not applied if `UseMmio` under section `Misc->Serial->Custom` is false. For MMIO, there are boot arguments `pcie_mmio_uart=ADDRESS` and `mmio_uart=ADDRESS` that allow the kernel to use MMIO for serial port access.
Note 5: The serial baud rate must be correctly set in both `BaudRate` under section `Misc->Serial->Custom` and via `serialbaud=VALUE` boot argument, both of which should match against each other. The default baud rate is 115200.
6. **CustomSMBIOSGuid**
Type: plist boolean

Failsafe: false

Requirement: 10.4

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

7. DisableIoMapper

Type: plist boolean

Failsafe: false

Requirement: 10.8 (not required for older)

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

Note 1: This option is a preferred alternative to deleting DMAR ACPI table and disabling VT-d in firmware preferences, which does not obstruct VT-d support in other systems in case they need this.

Note 2: Misconfigured IOMMU in the firmware may result in broken devices such as ethernet or Wi-Fi adapters. For instance, an ethernet adapter may cycle in link-up link-down state infinitely and a Wi-Fi adapter may fail to discover networks. Gigabyte is one of the most common OEMs with these issues.

8. DisableLinkeditJettison

Type: plist boolean

Failsafe: false

Requirement: 11

Description: Disables __LINKEDIT jettison code.

This option lets Lilu.kext, and possibly other kexts, function in macOS Big Sur at their best performance levels without requiring the keepsyms=1 boot argument.

9. DisableRtcChecksum

Type: plist boolean

Failsafe: false

Requirement: 10.4

Description: Disables primary checksum (0x58-0x59) writing in AppleRTC.

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

Note 2: This option will not protect areas from being overwritten at firmware stage (e.g. macOS bootloader), see AppleRtcRam protocol description if this is desired.

10. ExtendBTFeatureFlags

Type: plist boolean

Failsafe: false

Requirement: 10.8-11

Description: Set FeatureFlags to 0x0F for full functionality of Bluetooth, including Continuity.

Note: This option is a substitution for BT4LEContinuityFixup.kext, which does not function properly due to late patching progress.

11. ExternalDiskIcons

Type: plist boolean

Failsafe: false

Requirement: 10.4

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

Note: This option should be avoided whenever possible. Modern firmware typically have compatible AHCI controllers.

12. [ForceAquantiaEthernet](#)

[Type: plist boolean](#)

[Failsafe: false](#)

[Requirement: 10.15.4](#)

[Description: Enable Aquantia AQtion based 10GbE network cards support.](#)

[This option enables Aquantia AQtion based 10GbE network cards support, which used to work natively before macOS 10.15.4.](#)

Note: In order for Aquantia cards to properly function, `DisableIoMapper` must be disabled, `DMAR ACPI` table must not be dropped, and `VT-d` must be enabled in BIOS.

Note 2: While this patch should enable ethernet support for all Aquantia AQtion series, it has only been tested on AQC-107s based 10GbE network cards.

13. `ForceSecureBootScheme`

Type: plist boolean

Failsafe: false

Requirement: 11

Description: Force x86 scheme for IMG4 verification.

Note: This option is required on virtual machines when using `SecureBootModel` different from `x86legacy`.

14. `IncreasePciBarSize`

Type: plist boolean

Failsafe: false

Requirement: 10.10

Description: Allows `IOPCIFamily` to boot with 2 GB PCI BARs.

Normally macOS restricts PCI BARs to 1 GB. Enabling this option (still) does not let macOS actually use PCI devices with larger BARs.

Note: This option should be avoided whenever possible. A need for this option indicates misconfigured or defective firmware.

15. `LapicKernelPanic`

Type: plist boolean

Failsafe: false

Requirement: 10.6 (64-bit)

Description: Disables kernel panic on LAPIC interrupts.

16. `LegacyCommpage`

Type: plist boolean

Failsafe: false

Requirement: 10.4 - 10.6

Description: Replaces the default 64-bit commpage bcopy implementation with one that does not require SSSE3, useful for legacy platforms. This prevents a `commpage no match for last panic` due to no available 64-bit bcopy functions that do not require SSSE3.

17. `PanicNoKextDump`

Type: plist boolean

Failsafe: false

Requirement: 10.13 (not required for older)

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

18. `PowerTimeoutKernelPanic`

Type: plist boolean

Failsafe: false

Requirement: 10.15 (not required for older)

Description: Disables kernel panic on `setPowerState` timeout.

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

19. `ProvideCurrentCpuInfo`

Type: plist boolean

Failsafe: false

Requirement: 10.8 (10.14)

Description: Provides current CPU info to the kernel.

This quirk works differently depending on the CPU:

- For Microsoft Hyper-V it provides the correct TSC and FSB values to the kernel, as well as disables CPU topology validation (10.8+).
- For KVM and other hypervisors it provides precomputed MSR 35h values solving kernel panic with `-cpu host`.
- For Intel CPUs it adds support for asymmetrical SMP systems (e.g. Intel Alder Lake) by patching core count to thread count along with the supplemental required changes (10.14+).

20. SetApfsTrimTimeout

Type: plist integer

Failsafe: -1

Requirement: 10.14 (not required for older)

Description: Set trim timeout in microseconds for APFS filesystems on SSDs.

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

Depending on the SSD controller and the level of drive fragmentation, the trim procedure may take a considerable amount of time, causing noticeable boot slowdown. The APFS driver explicitly ignores previously unmapped areas and repeatedly trims them on boot. To mitigate against such boot slowdowns, the macOS driver introduced a timeout (9.999999 seconds) that stops the trim operation when not finished in time.

On several controllers, such as Samsung, where the deallocation process is relatively slow, this timeout can be reached very quickly. Essentially, it means that the level of fragmentation is high, thus macOS will attempt to 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. ~~Set~~ Setting this option to a high value, such as 4294967295 ~~, to ensure~~ 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 ~~, e.g. 999,~~ while 0 entirely disables it. Refer to this article for details.

~~On macOS 12+~~ 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 ~~set trim timeout for APFS filesystems~~ specify trim timeout. However, ~~trim it~~ can be disabled ~~when the timeout value is set to~~ 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.

21. 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.

22. XhciPortLimit

Type: plist boolean

Failsafe: false

Requirement: 10.11 (not required for older)

- 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 Boot Properties 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. Debug

Type: plist dict

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

4. 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.

5. Security

Type: plist dict

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

6. 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.

7. 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, particularly 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.

8.3 Boot Properties

1. ConsoleAttributes

Type: plist integer

Failsafe: 0

Description: Sets specific attributes for the console.

The text renderer supports colour arguments as a sum of foreground and background colours based on the UEFI specification. The value for black background and for black foreground, 0, is reserved.

List of colour values and names:

- 0x00 — EFI_BLACK
- 0x01 — EFI_BLUE
- 0x02 — EFI_GREEN
- 0x03 — EFI_CYAN
- 0x04 — EFI_RED
- 0x05 — EFI_MAGENTA
- 0x06 — EFI_BROWN
- 0x07 — EFI_LIGHTGRAY
- 0x08 — EFI_DARKGRAY
- 0x09 — EFI_LIGHTBLUE
- 0x0A — EFI_LIGHTGREEN
- 0x0B — EFI_LIGHTCYAN
- 0x0C — EFI_LIGHTRED
- 0x0D — EFI_LIGHTMAGENTA
- 0x0E — EFI_YELLOW
- 0x0F — EFI_WHITE
- 0x10 — EFI_BACKGROUND_BLACK
- 0x11 — EFI_BACKGROUND_BLUE
- 0x12 — EFI_BACKGROUND_GREEN
- 0x13 — EFI_BACKGROUND_CYAN
- 0x14 — EFI_BACKGROUND_RED
- 0x15 — EFI_BACKGROUND_MAGENTA
- 0x16 — EFI_BACKGROUND_BROWN
- 0x17 — EFI_BACKGROUND_LIGHTGRAY

8.4 Debug Properties

1. AppleDebug

Type: plist boolean

Failsafe: false

Description: Enable writing the `boot.efi` debug log to the OpenCore log.

Note: This option only applies to 10.15.4 and newer.

2. ApplePanic

Type: plist boolean

Failsafe: false

Description: Save macOS kernel panic output to the OpenCore root partition.

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

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

In cases where the OpenCore kernel panic saving mechanism is not used, kernel panic logs may still be found in the `/Library/Logs/DiagnosticReports` directory.

Starting with macOS Catalina, kernel panics are stored in JSON format and thus need to be preprocessed before passing to `kpdescribe.sh`:

```
cat Kernel.panic | grep macOSProcessedStackshotData |  
python -c 'import json,sys;print(json.load(sys.stdin)["macOSPanicString"])'  
python3 -c 'import json,sys;print(json.load(sys.stdin)["macOSPanicString"])'
```

3. DisableWatchDog

Type: plist boolean

Failsafe: false

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

4. DisplayDelay

Type: plist integer

Failsafe: 0

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

5. DisplayLevel

Type: plist integer, 64 bit

Failsafe: 0

Description: EDK II debug level bitmask (sum) showed onscreen. Unless **Target** enables console (onscreen) printing, onscreen debug output will not be visible.

The following levels are supported (discover more in `DebugLib.h`):

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

6. LogModules

Type: plist string

Failsafe: *

Description: Filter log entries by module.

This option filters logging generated by specific modules, both in the log and onscreen. Two modes are supported:

- + — Positive filtering: Only present selected modules.

- - — Negative filtering: Exclude selected modules.

When multiple ones are selected, comma (,) should be used as the splitter. For instance, `+OCCPU,OCA,OCB` means *only* OCCPU, OCA, OCB being printed, while `-OCCPU,OCA,OCB` indicates these modules being filtered out (i.e. *not* logged). When no symbol is specified, positive filtering (+) will be used. * indicates all modules being logged.

Note 1: Acronyms of libraries can be found in the **Libraries** section below.

Note 2: Messages printed before the configuration of log protocol cannot be filtered.

7. ~~SerialInitType: plist boolean Failsafe: false Description: Perform serial port initialisation.~~

~~This option will perform serial port initialisation within OpenCore prior to enabling (any) debug logging. Serial port configuration is defined via PCDs at compile time in gEfiMdeModulePkgTokenSpaceGuid GUID.~~

~~Default values as found in MdeModulePkg.dec are as follows:~~

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

~~Refer to the section for details.~~

8. SysReport

Type: plist boolean

Failsafe: false

Description: Produce system report on ESP folder.

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

Note: To maintain system integrity, the **SysReport** option is **not** available in **RELEASE** builds. Use a **DEBUG** build if this option is required.

9. Target

Type: plist integer

Failsafe: 0

Description: A bitmask (sum) of enabled logging targets. Logging output is hidden by default and this option must be set when such output is required, such as when debugging.

The following logging targets are supported:

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

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

To obtain Data Hub logs, use the following command in macOS (Note that Data Hub logs do not log kernel and kext patches):

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

UEFI variable log does not include some messages and has no performance data. To maintain system integrity, the log size is limited to 32 kilobytes. Some types of firmware may truncate it much earlier or drop completely if they have no memory. Using the **non-volatile** flag will cause the log to be written to NVRAM flash after every printed line.

To obtain UEFI variable logs, use the following command in macOS:

- (d) Boot failures might occur when the platform requires certain settings, but they have not been enabled because the associated issues were not discovered earlier. Be extra careful with `IgnoreInvalidFlexRatio` or `HashServices`.
- (e) Operating systems released before Apple Secure Boot was released (e.g. macOS 10.12 or earlier), will still boot until UEFI Secure Boot is enabled. This is so because Apple Secure Boot treats these as incompatible and they are then handled by the firmware (as Microsoft Windows is).
- (f) On older CPUs (e.g. before Sandy Bridge), enabling Apple Secure Boot might cause slightly slower loading (by up to 1 second).
- (g) As the `Default` value will increase with time to support the latest major released operating system, it is not recommended to use the `ApECID` and the `Default` settings together.
- (h) Installing macOS with Apple Secure Boot enabled is not possible while using HFS+ target volumes. This may include HFS+ formatted drives when no spare APFS drive is available.

The installed operating system may have sometimes outdated Apple Secure Boot manifests on the `Preboot` 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.

8.6 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.

8.6.1 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.d`

3. **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`.

4. **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`.

5. **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`.

6. **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.

7. **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`.

8. **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`.

9. **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`.

10. 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`.

11. 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`.

8.7 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. Name

Type: plist string

Failsafe: Empty

Description: Human readable entry name displayed in the OpenCore picker.

7. 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`.

8. RealPath

Type: plist boolean

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. LegacyEnable
Type: `plist boolean`
Failsafe: `false`
Description: Enables loading a NVRAM variable file named `nvram.plist` from EFI volume root.

This file must have a 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 the Delete (and Add) phases. Unless LegacyOverwrite is enabled, it will not overwrite any existing variable. Variables allowed to be set must be specified in LegacySchema.

- `nvda_drv=1` (legacy way to enable NVIDIA web driver, removed in 10.12)
- `npci=0x2000` (legacy, disables `kIOPCIConfiguratorPFM64`)
- `lapic_dont_panic=1` (disable lapic spurious interrupt panic on AP cores)
- `panic_on_display_hang=1` (trigger panic on display hang)
- `panic_on_gpu_hang=1` (trigger panic on GPU hang)
- `serial=VALUE` (configure serial logging mode) — The following bits are used by XNU:
 - * `0x01` (`SERIALMODE_OUTPUT`, bit 0) — Enable serial output.
 - * `0x02` (`SERIALMODE_INPUT`, bit 1) — Enable serial input.
 - * `0x04` (`SERIALMODE_SYCDRAIN`, bit 2) — Enable serial drain synchronisation.
 - * `0x08` (`SERIALMODE_BASE_TTY`, bit 3) — Load Base/Recovery/FVUnlock TTY.
 - * `0x10` (`SERIALMODE_NO_IOLOG`, bit 4) — Prevent IOLogs writing to serial.
- `slide=VALUE` (manually set KASLR slide)
- `smcdebug=VALUE` (AppleSMC debug mask)
- `spin_wait_for_gpu=1` (reduces GPU timeout on high load)
- `-amd_no_dgpu_accel` (alternative to WhateverGreen's `-radvesa` for new GPUs)
- `-nehalem_error_disable` (disables the AppleTyMCEDriver)
- `-no_compat_check` (disable model checking on 10.7+)
- `-s` (single mode)
- `-v` (verbose mode)
- `-x` (safe mode)

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

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg`
 Booter arguments, similar to `boot-args` but for `boot.efi`. Accepts a set of arguments, which are hexadecimal 64-bit values with or without `0x`. At different stages `boot.efi` will request different debugging (logging) modes (e.g. after `ExitBootServices` it will only print to serial). Several booter arguments control whether these requests will succeed. The list of known requests is covered below:
 - `0x00` – `INIT`.
 - `0x01` – `VERBOSE` (e.g. `-v`, force console logging).
 - `0x02` – `EXIT`.
 - `0x03` – `RESET:OK`.
 - `0x04` – `RESET:FAIL` (e.g. unknown `board-id`, hibernate mismatch, panic loop, etc.).
 - `0x05` – `RESET:RECOVERY`.
 - `0x06` – `RECOVERY`.
 - `0x07` – `REAN:START`.
 - `0x08` – `REAN:END`.
 - `0x09` – `DT` (can no longer log to `DeviceTree`).
 - `0x0A` – `EXITBS:START` (forced serial only).
 - `0x0B` – `EXITBS:END` (forced serial only).
 - `0x0C` – `UNKNOWN`.

In 10.15, debugging support was defective up to the 10.15.4 release due to refactoring issues as well as the introduction of a new debug protocol. Some of the arguments and their values below may not be valid for versions prior to 10.15.4. The list of known arguments is covered below:

- `boot-save-log=VALUE` — debug log save mode for normal boot.
 - * `0`
 - * `1`
 - * `2` — (default).
 - * `3`
 - * `4` — (save to file).
- `wake-save-log=VALUE` — debug log save mode for hibernation wake.
 - * `0` — disabled.
 - * `1`
 - * `2` — (default).
 - * `3` — (unavailable).
 - * `4` — (save to file, unavailable).
- `breakpoint=VALUE` — enables debug breaks (missing in production `boot.efi`).
 - * `0` — disables debug breaks on errors (default).
 - * `1` — enables debug breaks on errors.

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

2. **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.](#)

3. **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.

4. **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.

5. **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.

6. **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.

7. **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 like NVIDIA GOP Option ROMs on hardware with older EFI 1.x firmware like MacPro5,1.

8. **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.

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

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

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

Command (? for help): p
Disk \\.\\physicaldrive0: 419430400 sectors, 200.0 GiB
Sector size (logical): 512 bytes
Disk identifier (GUID): DEC57EB1-B3B5-49B2-95F5-3B8C4D3E4E12
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 419430366
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

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

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

Command (? for help): w

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

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

Listing 4: Relabeling Windows volume

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

12.4 Debugging

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

To obtain the log during boot serial port debugging can be used. Serial port debugging is enabled in Target, e.g. 0xB for onscreen with serial. To initialise serial within OpenCore use [SerialInitInit](#) configuration option [under Misc->Serial with other values properly set](#). For macOS the best choice is CP2102-based UART devices. Connect motherboard TX to USB UART RX, and motherboard GND to USB UART GND. Use `screen` utility to get the output, or download GUI software, such as CoolTerm.

Note: On several motherboards (and possibly USB UART dongles) PIN naming may be incorrect. It is very common to have GND swapped with RX, thus, motherboard “TX” must be connected to USB UART GND, and motherboard “GND” to USB UART RX.

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

12.5 Tips and Tricks

1. How do I debug boot failures?

Obtaining the actual error message is usually adequate. For this, ensure that:

- A DEBUG or NOOPT version of OpenCore is used.
- Logging is enabled (1) and shown onscreen (2): Misc → Debug → Target = 3.
- Logged messages from at least DEBUG_ERROR (0x80000000), DEBUG_WARN (0x00000002), and DEBUG_INFO (0x00000040) levels are visible onscreen: Misc → Debug → DisplayLevel = 0x80000042.
- Critical error messages, such as DEBUG_ERROR, stop booting: Misc → Security → HaltLevel = 0x80000000.
- Watch Dog is disabled to prevent automatic reboot: Misc → Debug → DisableWatchDog = true.
- Boot Picker (entry selector) is enabled: Misc → Boot → ShowPicker = true.

If there is no obvious error, check the available workarounds in the Quirks sections one by one. For early boot troubleshooting, for instance, when OpenCore menu does not appear, using UEFI Shell (bundled with OpenCore) may help to see early debug messages.

2. How do I debug macOS boot failures?

- Refer to boot-args values such as `debug=0x100`, `keepyms=1`, `-v`, and similar.
- Do not forget about AppleDebug and ApplePanic properties.
- [For macOS to correctly recognise and set up serial ports, the CustomPciSerialDevice quirk may be enabled when a PCI serial port card is installed.](#)
- Take care of Booter, Kernel, and UEFI quirks.
- Consider using serial port to inspect early kernel boot failures. For this `debug=0x108`, `serial=5`, and `msgbuf=1048576` boot arguments are needed. Refer to the patches in Sample.plist when dying before serial init.
- Always read the logs carefully.

3. How do I customise boot entries?

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

4. How do I choose the default boot entry?

OpenCore uses the primary UEFI boot option to select the default entry. This choice can be altered from UEFI Setup, with the macOS Startup Disk preference, or the Windows Boot Camp Control Panel. Since choosing OpenCore’s `BOOTx64.EFI` as a primary boot option limits this functionality in addition to several types of firmware deleting incompatible boot options, potentially including those created by macOS, users are strongly encouraged to use the `RequestBootVarRouting` quirk, which will preserve the selection made in the operating system within the OpenCore variable space. Note, that `RequestBootVarRouting` requires a separate driver for functioning.

5. What is the simplest way to install macOS?

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

To download recovery online `macrecovery.py` can be used.

For offline installation refer to How to create a bootable installer for macOS article. Apart from App Store and `softwareupdate` utility there also are third-party utilities to download an offline image.

6. Why do online recovery images (`*.dmg`) fail to load?

This may be caused by missing HFS+ driver, as all presently known recovery volumes have HFS+ filesystem.