OpenCore

Reference Manual (~~0.7.9~~0.8.0)

[2022.04.13]

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.

9. `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.

10. `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.

11. `ForceAquantiaEthernet`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.15.4
Description: Enable Aquantia AQtion AQC-107s based 10GbE network cards support.

This option enables Aquantia AQtion AQC-107s 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.

12. `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`.

13. `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.

14. `LapicKernelPanic`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.6 (64-bit)
Description: Disables kernel panic on LAPIC interrupts.

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 timeoutfor 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.

20. `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.

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

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

7.9 Scheme Properties

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

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

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

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

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

- 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: Configure PCD values required by `BaseSerialPortLib16550` for serial ports to properly function. Values are listed and described in the Serial Properties section below.

This section overrides the PCD values listed in BaseSerialPortLib16550.inf. 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, paticularly in vaulted configurations as well as those protected by secure boot, as such tools can be used to bypass the secure boot chain. Refer to the UEFI section for examples of UEFI tools.

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`
   - `0x00` — `EFI_BACKGROUND_BLACK`
   - `0x10` — `EFI_BACKGROUND_BLUE`
   - `0x20` — `EFI_BACKGROUND_GREEN`
   - `0x30` — `EFI_BACKGROUND_CYAN`
   - `0x40` — `EFI_BACKGROUND_RED`
   - `0x50` — `EFI_BACKGROUND_MAGENTA`
   - `0x60` — `EFI_BACKGROUND_BROWN`
   - `0x70` — `EFI_BACKGROUND_LIGHTGRAY`

   Note: This option may not work well with the `System` text renderer. Setting a background different from black could help with testing GOP functionality.

2. `HibernateMode`
   Type: `plist string`
   Failsafe: `None`
   Description: Hibernation detection mode. The following modes are supported:

   - `None` — Ignore hibernation state.
   - `Auto` — Use RTC and NVRAM detection.

(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. ~~`SerialInit`**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. 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:

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

   Warning: Certain firmware appear to have defective NVRAM garbage collection. As a result, they may not be able to always free space after variable deletion. Do not enable non-volatile NVRAM logging on such devices unless specifically required.

   While the OpenCore boot log already contains basic version information including build type and date, this information may also be found in the opencore-version NVRAM variable even when boot logging is disabled.

   File logging will create a file named opencore-YYYY-MM-DD-HHMMSS.txt (in UTC) under the 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 firmware are not reliable and may corrupt data when writing files through UEFI. Log writing is attempted in the safest manner and thus, is very slow. Ensure that DisableWatchDog is set to true when a slow drive is used. Try to avoid frequent use of this option when dealing with flash drives as large I/O amounts may speed up memory wear and render the flash drive unusable quicker.

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. `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. `Init`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform serial port initialisation. When this option is set to `false`, no keys from this section will be overridden.

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

   Refer to the `Debugging` section for details.

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

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

7. `PciDeviceInfo`
   Type: `plist data`

Failsafe: `0xFF`
Description: Set PCI serial device information.

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

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

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

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

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

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

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

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

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

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

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

8.7 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.

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`.

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 ~~`SerialInit`~~`Init` 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.