OpenCore

Reference Manual (0.5~~.0~~.1)

[2019.10.07]

4.4 Block Properties

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

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

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

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

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

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

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

4.5 Patch Properties

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

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

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

4. `Find`
   Type: `plist data`
   Failsafe: Empty data
   Description: Data to find. Must equal to `Replace` in size.

5. `Limit`
   Type: `plist integer`

Failsafe: `0`
Description: Maximum number of bytes to search for. Can be set to `0` to look through the whole ACPI table.

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

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

- Avoid renaming devices with ACPI patches. This may fail or perform improper renaming of unrelated devices (e.g. `EC` and `EC0`), be unnecessary, or even fail to rename devices in select tables. For ACPI consistency it is much safer to rename devices at I/O Registry level, as done by WhateverGreen.
- 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 `_PRW` or `_DSM` whenever possible.

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

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

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.

If you are using this for the first time on a customised firmware, there is a list of checks to do first. Prior to starting please ensure that you have:

- Most up-to-date UEFI firmware (check your motherboard vendor website).
- `Fast Boot` and `Hardware Fast Boot` disabled in firmware settings if present.
- `Above 4G Decoding` or similar enabled in firmware settings if present. Note, that on some motherboards (notably ASUS WS-X299-PRO) this option causes adverse effects, and must be disabled. While no other motherboards with the same issue are known, consider this option to be first to check if you have erratic boot failures.
- `DisableIoMapper` quirk enabled, or `VT-d` disabled in firmware settings if present, or ACPI DMAR table dropped.
- **No** 'slide' boot argument present in NVRAM or anywhere else. It is not necessary unless you cannot boot at all or see `No slide values are usable! Use custom slide!` message in the log.
- `CFG Lock` (MSR `0xE2` write protection) disabled in firmware settings if present. Cconsider patching it if you have enough skills and no option is available. See VerifyMsrE2 nots for more details.
- `CSM` (Compatibility Support Module) disabled in firmware settings if present. You may need to flash GOP ROM on NVIDIA 6xx/AMD 2xx or older. Use GopUpdate or AMD UEFI GOP MAKER in case you are not sure how.
- `EHCI/XHCI Hand-off` enabled in firmware settings `only` if boot stalls unless USB devices are disconnected.
- `VT-x`, `Hyper Threading`, `Execute Disable Bit` enabled in firmware settings if present.
- While it may not be required, sometimes you have to disable `Thunderbolt support`, `Intel SGX`, and `Intel Platform Trust` in firmware settings present.

When debugging sleep issues you may want to (temporarily) disable Power Nap and automatic power off, which appear to sometimes cause wake to black screen or boot loop issues on older platforms. The particular issues may vary, but in general you should check ACPI tables first. Here is an example of a bug found in some Z68 motherboards. To turn Power Nap and the others off run the following commands in Terminal:

```
sudo pmset autopoweroff 0
sudo pmset powernap 0
sudo pmset standby 0
```

Note: ~~these~~ These settings may reset at hardware change and in certain other circumstances. To view their current state use `pmset -g` command in Terminal.

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.

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

5. ~~`MatchKernel`~~`MaxKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: Adds kernel driver on ~~selected macOS version only. The selection happens based on prefix match with~~ specified macOS version or older.

   Kernel version can be obtained with `uname -r` command, and should look like 3 numbers separated by dots, for example `18.7.0` is the kernel version ~~, i. e.~~ for `10.14.6`. Kernel version interpretation is implemented as follows:

$$\begin{aligned}
ParseDarwinVersion(\kappa,\lambda,\mu) = & (\lfloor\frac{\kappa}{10}\rfloor \cdot 10 + \kappa - \lfloor\frac{\kappa}{10}\rfloor \cdot 10) \cdot 10000 && \text{Where } \kappa \in (0,99) \text{ is kernel version major} \\
& + (\lfloor\frac{\lambda}{10}\rfloor \cdot 10 + \lambda - \lfloor\frac{\lambda}{10}\rfloor \cdot 10) \cdot 100 && \text{Where } \lambda \in (0,99) \text{ is kernel version minor} \\
& + (\lfloor\frac{\mu}{10}\rfloor \cdot 10 + \mu - \lfloor\frac{\mu}{10}\rfloor \cdot 10) && \text{Where } \mu \in (0,99) \text{ is kernel version patch}
\end{aligned}$$

   Kernel version comparison is implemented as follows:

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

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

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

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

   Here *ParseDarwinVersion* argument is assumed to be 3 integers obtained by splitting Darwin kernel version string from left to right by the `.` symbol. *FindDarwinVersion* function looks up Darwin kernel version by locating ~~`16.7.0`~~`"Darwin Kernel Version` $\kappa.\lambda.\mu$`"` ~~will match macOS 10.12.6 and `16.` will match any macOS 10.12. x version~~ string in the kernel image.

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

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

7. `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`~~`MaxKernel`
   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.~~ specified macOS version or older.

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

5. ~~`16.7.0`~~`MinKernel`~~will match macOS 10.12.6 and `16.` will match any macOS 10.12.x version~~
   Type: `plist string`
   Failsafe: Empty string
   Description: Blocks kernel driver on specified macOS version or newer.

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

7.5 Emulate Properties

1. `Cpuid1Data`
   Type: `plist data`, 16 bytes
   Failsafe: All zero
   Description: Sequence of `EAX`, `EBX`, `ECX`, `EDX` values in Little Endian order to replace `CPUID (1)` call in XNU kernel. Normally it is only the value of `EAX` that needs to be taken care of, which represents the exact CPUID. And the remainders are to be left as zeroes. For instance, `A9 06 03 00` stands for CPUID `0x0306A9` (Ivy Bridge). A good example can be found at acidanthera/bugtracker#365. (See `Special NOTES for Haswell+ low-end`)

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`
   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.

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

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

5. `Find`
   Type: `plist data`
   Failsafe: Empty data
   Description: Data to find. Can be set to empty for immediate replacement at `Base`. Must equal to `Replace` in size otherwise.

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

7. `Limit`
   Type: `plist integer`
   Failsafe: `0`
   Description: Maximum number of bytes to search for. Can be set to `0` to look through the whole kext or kernel.

8. `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.

9. ~~`MatchKernel`~~`MaxKernel`
   Type: `plist string`
   Failsafe: Empty string
   Description: ~~Adds kernel driver to selected macOS version only. The selection happens based on prefix match with the kernel version, i. e.~~ Patches data on specified macOS version or older.

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

10. ~~`16.7.0`~~`MinKernel`~~will match macOS 10.12.6 and `16.` will match any macOS 10.12.x version~~
    Type: `plist string`
    Failsafe: Empty string
    Description: Patches data on specified macOS version or newer.

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

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

12. `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.

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

7.7 Quirks Properties

1. `AppleCpuPmCfgLock`
   Type: `plist boolean`
   Failsafe: `false`
   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 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.

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. `BlessOverride`
   Type: `plist array`
   Description: Add custom scanning paths through bless model.

   Designed to be filled with `plist string` entries containing absolute UEFI paths to customised bootloaders, for example, `\EFI\Microsoft\bootmgfw.efi` for Microsoft bootloader. This allows unusual boot paths to be automaticlly discovered by the boot picker. Designwise they are equivalent to predefined blessed path, such as `\System\Library\CoreServices\boot.efi`, but unlike predefined bless paths they have highest priority.
3. `Debug`
   Type: `plist dict`
   Description: Apply debug configuration described in Debug Properties section below.
4. `Entries`
   Type: `plist array`
   Description: Add boot entries to boot picker.

   Designed to be filled with `plist dict` values, describing each load entry. See Entry Properties section below.
5. `Security`
   Type: `plist dict`
   Description: Apply security configuration described in Security Properties section below.
6. `Tools`
   Type: `plist array`
   Description: Add tool entries to boot picker.

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

   Note: Select tools, for example, UEFI Shell ~~or~~ 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.

   Note: This field is best to be left empty on most firmwares.
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.

- `CMD+S` — single user mode.
- `CMD+S+MINUS` — disable KASLR slide, requires disabled SIP.
- `CMD+V` — verbose mode.
- `Shift` — safe mode.

7. `Resolution`
   Type: `plist string`
   Failsafe: Empty string
   Description: Sets console output screen resolution.

   - Set to `WxH@Bpp` (e.g. `1920x1080@32`) or `WxH` (e.g. `1920x1080`) formatted string to request custom resolution from GOP if available.
   - Set to empty string not to change screen resolution.
   - Set to `Max` to try to use 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`.

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

9. `Timeout`
   Type: `plist integer`, 32 bit
   Failsafe: `0`
   Description: Timeout in seconds in boot picker before automatic booting of the default boot entry.

10. `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 picker oneself. Reference example of external graphics interface is provided in ExternalUi test driver.

    OpenCore built-in boot picker contains a set of actions chosen during the boot process. The list of supported actions is similar to Apple BDS and currently consists of the following options:

    - `Default` — this is the default option, and it lets OpenCore built-in boot picker to loads the default boot option as specified in Startup Disk preference pane.
    - `ShowPicker` — this option forces picker to show. Normally it can be achieved by holding `OPT` key during boot. Setting `ShowPicker` to `true` will make `ShowPicker` the default option.
    - `ResetNvram` — this option performs select UEFI variable erase and is normally achieved by holding `CMD+OPT+P+R` key combination during boot. Another way to erase UEFI variables is to choose `Reset NVRAM` in the picker. This option requires `AllowNvramReset` to be set to `true`.
    - `BootApple` — this options performs booting to the first found Apple operating system unless the default chosen operating system is already made by Apple. Hold `X` key to choose this option.
    - `BootAppleRecovery` — this option performs booting to Apple operating system recovery. Either the one related to the default chosen operating system, or first found in case default chosen operating system is not made by Apple or has no recovery. Hold `CMD+R` key combination to choose this option.

    Note: activated ~~`AppleGenericInput`~~`KeySupport`, `UsbKbDxe`, or similar driver is required for key handling to work. On many firmwares it is not possible to get all the keys function.

    In addition to `OPT` OpenCore supports `Escape` key `ShowPicker`. This key exists for firmwares with PS/2 keyboards that fail to report held `OPT` key and require continual presses of `Escape` key to enter the boot menu.

- `0x00040000` (bit 18) — `OC_SCAN_ALLOW_DEVICE_SCSI`, allow scanning SCSI devices.
- `0x00080000` (bit 19) — `OC_SCAN_ALLOW_DEVICE_NVME`, allow scanning NVMe devices.
- `0x00100000` (bit 20) — `OC_SCAN_ALLOW_DEVICE_ATAPI`, allow scanning CD/DVD devices.
- `0x00200000` (bit 21) — `OC_SCAN_ALLOW_DEVICE_USB`, allow scanning USB devices.
- `0x00400000` (bit 22) — `OC_SCAN_ALLOW_DEVICE_FIREWIRE`, allow scanning FireWire devices.
- `0x00800000` (bit 23) — `OC_SCAN_ALLOW_DEVICE_SDCARD`, allow scanning card reader devices.

Note: Given the above description, `0xF0103` value is expected to allow scanning of SATA, SAS, SCSI, and NVMe devices with APFS file system, and prevent scanning of any devices with HFS or FAT32 file systems in addition to not scanning APFS file systems on USB, CD, ~~USB,~~ and FireWire drives. The combination reads as:

- `OC_SCAN_FILE_SYSTEM_LOCK`
- `OC_SCAN_DEVICE_LOCK`
- `OC_SCAN_ALLOW_FS_APFS`
- `OC_SCAN_ALLOW_DEVICE_SATA`
- `OC_SCAN_ALLOW_DEVICE_SASEX`
- `OC_SCAN_ALLOW_DEVICE_SCSI`
- `OC_SCAN_ALLOW_DEVICE_NVME`

8.6 Entry Properties

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

2. `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.

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

4. `Name`
   Type: `plist string`
   Failsafe: Empty string
   Description: Human readable entry name displayed in boot picker.

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

   - `Entries` specify external boot options, and therefore take device paths in `Path` key. These values are not checked, thus be extremely careful. Example: `PciRoot(0x0)/Pci(0x1,0x1)/.../\EFI\COOL.EFI`
   - `Tools` specify internal boot options, which are part of bootloader vault, and therefore take file paths relative to `OC/Tools` directory. Example: `~~CleanNvram~~Shell.efi`.

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.
   - ~~— 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` — `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.
   - `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 builtin ~~`AppleGenericInput`~~`KeySypport`, 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 `HFSPlus` 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 UDK (EDK II) use the same command you do normally use for OpenCore compilation, but choose a corresponding package:

```
git clone https://github.com/acidanthera/audk 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. `Input`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual settings designed for input (keyboard and mouse) in Input Properties section below.

4. `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.

5. `Quirks`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual firmware quirks described in Quirks Properties section below.

11.3 Input Properties

1. `KeyForgetThreshold`
   Type: `plist integer`
   Failsafe: `0`
   Description: Remove key unless it was submitted during this timeout in milliseconds.

   `AppleKeyMapAggregator` protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers only report key presses as interrupts and pressing and holding the key on the keyboard results in subsequent submissions of this key with some defined time interval. As a result we use a timeout to remove once pressed keys from the buffer once the timeout expires and no new submission of this key happened.

   This option allows to set this timeout based on your platform. The recommended value that works on the majority of the platforms is `5` milliseconds. For reference, holding one key on VMware will repeat it roughly every `2` milliseconds and the same value for APTIO V is `3-4` milliseconds. Thus it is possible to set a slightly lower value on faster platforms and slightly higher value on slower platforms for more responsive input.

2. `KeyMergeThreshold`
   Type: `plist integer`
   Failsafe: `0`
   Description: Assume simultaneous combination for keys submitted within this timeout in milliseconds.

   Similarly to `KeyForgetThreshold`, this option works around the sequential nature of key submission. To be able to recognise simultaneously pressed keys in the situation when all keys arrive sequentially, we are required to set a timeout within which we assume the keys were pressed together.

   Holding multiple keys results in reports every `2` and `1` milliseconds for VMware and APTIO V respectively. Pressing keys one after the other results in delays of at least `6` and `10` milliseconds for the same platforms. The recommended value for this option is `2` milliseconds, but it may be decreased for faster platforms and increased for slower.

3. `KeySupport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable internal keyboard input translation to `AppleKeyMapAggregator` protocol.

This option activates the internal keyboard interceptor driver, based on `AppleGenericInput` aka `AptioIntputFix`), to fill `AppleKeyMapAggregator` database for input functioning. In case a separate driver is used, such as `UsbKbDxe`, this option should never be enabled.

4. `KeySupportMode`
   Type: `plist string`
   Failsafe: empty string
   Description: Set internal keyboard input translation to `AppleKeyMapAggregator` protocol mode.

   - `Auto` — Performs automatic choice as available with the following preference: `AMI`, `V2`, `V1`.
   - `V1` — Uses UEFI standard legacy input protocol `EFI_SIMPLE_TEXT_INPUT_PROTOCOL`.
   - `V2` — Uses UEFI standard modern input protocol `EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL`.
   - `AMI` — Uses APTIO input protocol `AMI_EFIKEYCODE_PROTOCOL`.

5. `KeySwap`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Swap `Command` and `Option` keys during submission.

   This option may be useful for keyboard layouts with `Option` key situated to the right of `Command` key.

6. `PointerSupport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable internal pointer driver.

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

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

   Currently the only supported variant is `ASUS`, using specialised protocol available on select Z87 and Z97 ASUS boards. More details can be found in `LongSoft/UefiTool#116`.

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

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

   The recommended value is `50000` (`5` milliseconds) or slightly higher. ASUS boards use `60000` for the interface. Apple boards use `100000`.

11.4 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. `AppleEvent`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reinstalls Apple Event protocol with a builtin version. This may be used to ensure File Vault 2 compatibility on VMs or legacy Macs.

3. `AppleImageConversion`
   Type: `plist boolean`

```
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`~~`RX`, and motherboard `GND` to USB UART ~~`RX`~~`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 you have to connect motherboard "`TX`" 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 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: `Misc` → `Debug` → `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. For early boot troubleshooting, for instance, when OpenCore menu does not appear, using UEFI Shell may help to see early debug messages.

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. **How to 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 firmwares deleting incompatible boot options, potentially including those created by macOS, you are strongly encouraged to use the `RequestBootVarRouting` quirk, which will preserve your selection made in the operating system within the OpenCore variable space. Note, that `RequestBootVarRouting` requires a separate driver for functioning.

4. **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 macrecovery.py tool from MacInfoPkg.

For offline installation refer to How to create a bootable installer for macOS article.

5. **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.

6. **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.

7. **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.

8. **How can I migrate from `AptioMemoryFix`?**

Behaviour similar to that of `AptioMemoryFix` can be obtained by installing `FwRuntimeServices` driver and enabling the quirks listed below. Please note, that most of these are not necessary to be enabled. Refer to their individual descriptions in this document for more details.

- `ProvideConsoleGop` (UEFI quirk)
- `AvoidRuntimeDefrag`
- `DiscardHibernateMap`
- `EnableSafeModeSlide`
- `EnableWriteUnprotector`
- `ForceExitBootServices`
- `ProtectCsmRegion`