



OpenCore

Reference Manual (~~0.0.4~~0.5.0)

[2019.08.31]

1 Introduction

This document provides information on OpenCore user configuration file format used to setup the correct functioning of macOS operating system. It is to be read as the official clarification of expected OpenCore behaviour. All deviations, if found in published OpenCore releases, shall be considered documentation or implementation bugs, and are requested to be reported through Acidanthera Bugtracker. All other sources or translations of this document are unofficial and may contain errors.

1.1 ~~Known defects~~

~~For OpenCore issues please refer to~~ This document is structured as a specification, and is not meant to provide a step by step algorithm for configuring end-user board support package (BSP). Any third-party articles, tools, books, etc., providing such material are prone to their authors' preferences, tastes, this document misinterpretation, and essential obsolescence. In case you still use these sources, for example, Opencore Vanilla Desktop Guide, please ensure following this document for every made decision and judging its consequences. Regardless of the sources used you are required to fully understand every dedicated OpenCore configuration option and concept prior to reporting any issues in Acidanthera Bugtracker.

1.1 Generic Terms

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

Main booter driver responsible for operating system loading.

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

3.2 Installation and Upgrade

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

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

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

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

3.3 Contribution

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

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

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

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

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

Listing 1: Compilation Commands

For IDE usage Xcode projects are available in the root of the repositories. Another approach could be Sublime Text with `EasyClangComplete` plugin. Add `.clang_complete` file with similar content to your UDK root:

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
-I/UefiPackages/OcSupportPkg/Include
-I/UefiPackages/MacInfoPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IIInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
```

Listing 2: ECC Configuration

Warning: Tool developers modifying `config.plist` or any other OpenCore files must ensure that their tool checks for `opencore-version` NVRAM variable (see Debug Properties section below) and warn the user if the version listed is unsupported or prerelease. OpenCore configuration may change across the releases and the tool shall ensure that it carefully follows this document. Failure to do so may result in this tool to be considered as malware and blocked with all possible means.

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. Consider patching it if you have enough skills and no option is available. See `VerifyMsrE2` notes 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 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.

2. [DevirtualiseMmio](#)

Type: plist boolean

Failsafe: false

Description: Remove runtime attribute from select MMIO regions.

This option reduces stolen memory footprint from the memory map by removing runtime bit for known memory regions. This quirk may result in the increase of KASLR slides available, but is not necessarily compatible with the target board.

Note: This option is generally useful on APTIO V firmwares (Broadwell and newer).

3. `DisableVariableWrite`

Type: plist boolean

Failsafe: false

Description: Protect from macOS NVRAM write access.

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

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

4. `DiscardHibernateMap`

Type: plist boolean

Failsafe: false

Description: Reuse original hibernate memory map.

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

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

5. `EnableSafeModeSlide`

Type: plist boolean

Failsafe: false

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

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

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

6. `EnableWriteUnprotector`

Type: plist boolean

Failsafe: false

Description: Permit write access to UEFI runtime services code.

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

Note: The necessity of this quirk is determined by early boot crashes of the firmware.

7. `ForceExitBootServices`

Type: plist boolean

Failsafe: false

Description: Retry `ExitBootServices` with new memory map on failure.

Try to ensure that `ExitBootServices` call succeeds even with outdated `MemoryMap` key argument by obtaining current memory map and retrying `ExitBootServices` call.

Failsafe: Empty string

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

5. MatchKernel

Type: plist string

Failsafe: Empty string

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

6. PlistPath

Type: plist string

Failsafe: Empty string

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

7.4 Block Properties

1. Comment

Type: plist string

Failsafe: Empty string

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

2. Enabled

Type: plist boolean

Failsafe: false

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

3. Identifier

Type: plist string

Failsafe: Empty string

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

4. MatchKernel

Type: plist string

Failsafe: Empty string

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

7.5 Emulate Properties

1. Cpuid1Data

Type: plist data, 16 bytes

Failsafe: All zero

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

- 0x00000002 (bit 1) — `OC_SCAN_DEVICE_LOCK`, restricts scanning to only known device types defined as a part of this policy. This is not always possible to detect protocol tunneling, so be aware that on some systems it may be possible for e.g. USB HDDs to be recognised as SATA. Cases like this must be reported. Known device types are prefixed with `OC_SCAN_ALLOW_DEVICE_`.
- 0x00000100 (bit 8) — `OC_SCAN_ALLOW_FS_APFS`, allows scanning of APFS file system.
- 0x00000200 (bit 9) — `OC_SCAN_ALLOW_FS_HFS`, allows scanning of HFS file system.
- 0x00000400 (bit 10) — `OC_SCAN_ALLOW_FS_ESP`, allows scanning of EFI System Partition file system.
- [0x00000800 \(bit 11\) — `OC_SCAN_ALLOW_FS_NTFS`, allows scanning of NTFS \(Msft Basic Data\) file system.](#)
- [0x00001000 \(bit 12\) — `OC_SCAN_ALLOW_FS_EXT`, allows scanning of EXT \(Linux Root\) file system.](#)
- 0x00010000 (bit 16) — `OC_SCAN_ALLOW_DEVICE_SATA`, allow scanning SATA devices.
- 0x00020000 (bit 17) — `OC_SCAN_ALLOW_DEVICE_SASEX`, allow scanning SAS and Mac NVMe devices.
- 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. 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 entry will not be listed unless set to `true`.

3. Name

Type: plist string

Failsafe: Empty string

Description: Human readable entry name displayed in boot picker.

4. 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.efi`.

10.3 DataHub Properties

1. PlatformName
Type: plist string
Failsafe: Not installed
Description: Sets name in gEfiMiscSubClassGuid. Value found on Macs is platform in ASCII.
2. SystemProductName
Type: plist string
Failsafe: Not installed
Description: Sets Model in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS SystemProductName in Unicode.
3. SystemSerialNumber
Type: plist string
Failsafe: Not installed
Description: Sets SystemSerialNumber in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS SystemSerialNumber in Unicode.
4. SystemUUID
Type: plist string, GUID
Failsafe: Not installed
Description: Sets system-id in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS SystemUUID.
5. BoardProduct
Type: plist string
Failsafe: Not installed
Description: Sets board-id in gEfiMiscSubClassGuid. Value found on Macs is equal to SMBIOS BoardProduct in ASCII.
6. BoardRevision
Type: plist data, 1 byte
Failsafe: 0
Description: Sets board-rev in gEfiMiscSubClassGuid. Value found on Macs seems to correspond to internal board revision (e.g. 01).
7. StartupPowerEvents
Type: plist integer, 64-bit
Failsafe: 0
Description: Sets StartupPowerEvents in gEfiMiscSubClassGuid. Value found on Macs is power management state bitmask, normally 0. Known bits read by X86PlatformPlugin.kext:
 - 0x00000001 — Shutdown cause was a PWR0K event (Same as GEN_PMCON_2 bit 0)
 - 0x00000002 — Shutdown cause was a SYS_PWR0K event (Same as GEN_PMCON_2 bit 1)
 - 0x00000004 — Shutdown cause was a THRMTRIP# event (Same as GEN_PMCON_2 bit 3)
 - 0x00000008 — Rebooted due to a SYS_RESET# event (Same as GEN_PMCON_2 bit 4)
 - 0x00000010 — Power Failure (Same as GEN_PMCON_3 bit 1 PWR_FLR)
 - 0x00000020 — Loss of RTC Well Power (Same as GEN_PMCON_3 bit 2 RTC_PWR_STS)
 - 0x00000040 — General Reset Status (Same as GEN_PMCON_3 bit 9 GEN_RST_STS)
 - 0xfffff80 — SUS Well Power Loss (Same as GEN_PMCON_3 bit 14)
 - 0x00010000 — Wake cause was a ME Wake event (Same as PRSTS bit 0, ME_WAKE_STS)
 - 0x00020000 — Cold Reboot was ME Induced event (Same as PRSTS bit 1 ME_HRST_COLD_STS)
 - 0x00040000 — Warm Reboot was ME Induced event (Same as PRSTS bit 2 ME_HRST_WARM_STS)
 - 0x00080000 — Shutdown was ME Induced event (Same as PRSTS bit 3 ME_HOST_PWRDN)
 - 0x00100000 — Global reset ME Wachdog Timer event (Same as PRSTS bit 6)
 - 0x00200000 — Global reset PowerManagment Wachdog Timer event (Same as PRSTS bit 15)
8. InitialTSC
Type: plist integer, 64-bit
Failsafe: 0
Description: Sets InitialTSC in gEfiProcessorSubClassGuid. Sets initial TSC value, normally 0.

9. FSBFrequency

Type: plist integer, 64-bit
Failsafe: Automatic
Description: Sets FSBFrequency in gEfiProcessorSubClassGuid.

Sets CPU FSB frequency. [This value equals to CPU nominal frequency divided by CPU maximum bus ratio and is specified in Hz. Refer to MSR_NEHALEM_PLATFORM_INFO \(CEh\) MSR value to determine maximum bus ratio on modern Intel CPUs.](#)

[Note: This value is not used on Skylake and newer but is still provided to follow suit.](#)
10. ARTFrequency

Type: plist integer, 64-bit
Failsafe: ~~Not installed~~Automatic
Description: Sets ARTFrequency in gEfiProcessorSubClassGuid. ~~Sets-~~

[This value contains CPU ART frequency, ~~Skylake~~ also known as crystal clock frequency. Its existence is exclusive to Skylake generation and newer. The value is specified in Hz, and is normally 24 MHz for client Intel segment, 25 MHz for server Intel segment, and 19.2 MHz for Intel Atom CPUs. macOS till 10.15 inclusive assumes 24 MHz by default.](#)
11. DevicePathsSupported

Type: plist integer, 32-bit
Failsafe: Not installed
Description: Sets DevicePathsSupported in gEfiMiscSubClassGuid. Must be set to 1 for AppleACPIPlatform.kext to append SATA device paths to Boot#### and efi-boot-device-data variables. Set to 1 on all modern Macs.
12. SmcRevision

Type: plist data, 6 bytes
Failsafe: Not installed
Description: Sets REV in gEfiMiscSubClassGuid. Custom property read by VirtualSMC or FakeSMC to generate SMC REV key.
13. SmcBranch

Type: plist data, 8 bytes
Failsafe: Not installed
Description: Sets RBr in gEfiMiscSubClassGuid. Custom property read by VirtualSMC or FakeSMC to generate SMC RBr key.
14. SmcPlatform

Type: plist data, 8 bytes
Failsafe: Not installed
Description: Sets RP1t in gEfiMiscSubClassGuid. Custom property read by VirtualSMC or FakeSMC to generate SMC RP1t key.

10.4 PlatformNVRAM Properties

1. BID

Type: plist string
Failsafe: Not installed
Description: Specifies the value of NVRAM variable 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID.
2. ROM

Type: plist data, 6 bytes
Failsafe: Not installed
Description: Specifies the values of NVRAM variables 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM and 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM.
3. MLB

Type: plist string
Failsafe: Not installed

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.
- ~~— Apple-specific user interface support driver. This driver brings the support for FileVault 2 GUI, hotkey parsing (shift, cmd+v, etc.), language collation support, and certain other features important for normal macOS functioning. For hotkey support AppleKeyMapAggregator-compatible driver is required.—~~
- **AppleGenericInput** — user input driver adding the support of **AppleKeyMapAggregator** protocols on top of different UEFI input protocols. Additionally resolves mouse input issues on select firmwares. This is an alternative to **UsbKbDxe**, which may work better or worse depending on the firmware.
- **FwRuntimeServices** — `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.
- ~~— Unicode collation driver from **MdeModulePkg**. This driver is a lightweight alternative to **AppleUiSupport**, which contains no Apple-specific code, and only provides unicode collation support. The driver is not recommended for use on any hardware but few original Macs.—~~
- **EnhancedFatDxe** — FAT filesystem driver from **FatPkg**. This driver is embedded in all UEFI firmwares, and cannot be used from OpenCore. It is known that multiple firmwares have a bug in their FAT support implementation, which leads to corrupted filesystems on write attempt. Embedding this driver within the firmware may be required in case writing to EFI partition is needed during the boot process.
- **NvmExpressDxe** — NVMe support driver from **MdeModulePkg**. This driver is included in most firmwares starting with Broadwell generation. For Haswell and earlier embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
- **UsbKbDxe** — USB keyboard driver adding the support of **AppleKeyMapAggregator** protocols on top of a custom USB keyboard driver implementation. This is an alternative to **AptioInputFix**, which may work better or worse depending on the firmware.
- **VirtualSmc** — UEFI SMC driver, required for proper FileVault 2 functionality and potentially other macOS specifics. An alternative, named **SMCHelper**, is not compatible with **VirtualSmc** and OpenCore, which is unaware of its specific interfaces. In case **FakeSMC** kernel extension is used, manual NVRAM variable addition may be needed and **VirtualSmc** driver should still be used.
- **VBoxHfs** — HFS file system driver with bless support. This driver is an alternative to a closed source **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. Protocols

Type: plist dict

Failsafe: None

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

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

4. Quirks

Type: plist dict

Failsafe: None

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

11.3 Protocols Properties

1. AppleBootPolicy

Type: plist boolean

Failsafe: false

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

2. [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](#)

Failsafe: [false](#)

Description: [Reinstalls Apple Image Conversion protocol with a builtin version.](#)

4. [AppleKeyMap](#)

Type: [plist boolean](#)

Failsafe: [false](#)

Description: [Reinstalls Apple Key Map protocols with builtin versions.](#)

5. [AppleUserInterfaceTheme](#)

Type: [plist boolean](#)

Failsafe: [false](#)

Description: [Reinstalls Apple User Interface Theme protocol with a builtin version.](#)

6. ConsoleControl

Type: plist boolean

Failsafe: false

Description: Replaces Console Control protocol with a builtin version.

macOS bootloader requires console control protocol for text output, which some firmwares miss. This option is required to be set when the protocol is already available in the firmware, and other console control options

are used, such as `IgnoreTextInGraphics`, `SanitiseClearScreen`, and sometimes `ConsoleBehaviourOs` with `ConsoleBehaviourUi`).

7. `DataHub`
Type: plist boolean
Failsafe: false
Description: Reinstalls Data Hub protocol with a builtin version. This will drop all previous properties if the protocol was already installed.
8. `DeviceProperties`
Type: plist boolean
Failsafe: false
Description: Reinstalls Device Property protocol with a builtin version. This will drop all previous properties if it was already installed. This may be used to ensure full compatibility on VMs or legacy Macs.
9. [`FirmwareVolume`](#)
Type: [plist boolean](#)
Failsafe: [false](#)
Description: [Forcibly wraps Firmware Volume protocols or installs new to support custom cursor images for File Vault 2. Should be set to true to ensure File Vault 2 compatibility on everything but VMs and legacy Macs.](#)
10. [`HashServices`](#)
Type: [plist boolean](#)
Failsafe: [false](#)
Description: [Forcibly reinstalls Hash Services protocols with builtin versions. Should be set to true to ensure File Vault 2 compatibility on platforms providing broken SHA-1 hashing. Can be diagnosed by invalid cursor size with UIScale set to 02, in general platforms prior to APTIO V \(Haswell and older\) are affected.](#)
11. [`UnicodeCollation`](#)
Type: [plist boolean](#)
Failsafe: [false](#)
Description: [Forcibly reinstalls unicode collation services with builtin version. Should be set to true to ensure UEFI Shell compatibility on platforms providing broken unicode collation. In general legacy Insyde and APTIO platforms on Ivy Bridge and earlier are affected.](#)

11.4 Quirks Properties

1. `AvoidHighAlloc`
Type: plist boolean
Failsafe: false
Description: Advises allocators to avoid allocations above first 4 GBs of RAM.

This is a workaround for select board firmwares, namely GA-Z77P-D3 (rev. 1.1), failing to properly access higher memory in UEFI Boot Services. On these boards this quirk is required for booting entries that need to allocate large memory chunks, such as macOS DMG recovery entries. On unaffected boards it may cause boot failures, and thus strongly not recommended. For known issues refer to [`acidanthera/bugtracker#449`](#).

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

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

3. `IgnoreInvalidFlexRatio`
Type: plist boolean
Failsafe: false

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

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

4. `IgnoreTextInGraphics`

Type: plist boolean

Failsafe: false

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

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

5. [ReplaceTabWithSpace](#)

Type: plist boolean

Failsafe: false

Description: [Some firmwares do not print tab characters or even everything that follows them, causing difficulties or inability to use the UEFI Shell builtin text editor to edit property lists and other documents. This option makes the console output spaces instead of tabs.](#)

Note: [ConsoleControl may need to be set to true for this to work.](#)

6. `ProvideConsoleGop`

Type: plist boolean

Failsafe: false

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

7. `ReleaseUsbOwnership`

Type: plist boolean

Failsafe: false

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

8. `RequestBootVarRouting`

Type: plist boolean

Failsafe: false

Description: Request `redirectBoot` prefixed variables from `EFI_GLOBAL_VARIABLE_GUID` to `OC_VENDOR_VARIABLE_GUID`.

This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `FwRuntimeServices.efi`. The quirk lets default boot entry preservation at times when firmwares delete incompatible boot entries. Simply said, you are required to enable this quirk to be able to reliably use Startup Disk preference pane in a firmware that is not compatible with macOS boot entries by design.

9. `SanitiseClearScreen`

Type: plist boolean

Failsafe: false

Description: Some firmwares reset screen resolution to a failsafe value (like 1024x768) on the attempts to clear screen contents when large display (e.g. 2K or 4K) is used. This option attempts to apply a workaround.

Note: `ConsoleControl` may need to be set to `true` for this to work. On all known affected systems `ConsoleMode` had to be set to empty string for this to work.

10. [ClearScreenOnModeSwitch](#)

Type: plist boolean

Failsafe: false

Description: [Some firmwares clear only part of screen when switching from graphics to text mode, leaving a](#)

fragment of previously drawn image visible. This option fills the entire graphics screen with black color before switching to text mode.

Note: ConsoleControl should be set to true for this to work.

- 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.

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 ~~tool from~~ `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`
- `ProvideCustomSlide`
- `SetupVirtualMap`
- `ShrinkMemoryMap`