



# OpenCore

Reference Manual (0.5.~~2~~.3)

[2019.11.01]

The addresses written here must be part of the memory map, have `EfiMemoryMappedIO` type and `EFI_MEMORY_RUNTIME` attribute (highest bit) set. To find the list of the candidates the debug log can be used.

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 address will be devirtualised unless set to `true`.

## 5.4 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 [without additional measures](#). In general this frees from 64 to 256 megabytes of memory (present in the debug log), and on some platforms it is the only way to boot macOS, which otherwise fails with allocation error at bootloader stage.

This option is generally useful on all firmwares except some very old ones, like Sandy Bridge. On select firmwares it may require a list of exceptional addresses that still need to get their virtual addresses for proper NVRAM and hibernation functioning. Use `MmioWhitelist` section to do this.

3. `DisableSingleUser`

**Type:** `plist boolean`

**Failsafe:** `false`

**Description:** Disable single user mode.

This is a security option allowing one to restrict single user mode usage by ignoring `CMD+S` hotkey and `-s` boot argument. The behaviour with this quirk enabled is supposed to match T2-based model behaviour. Read this [article](#) to understand how to use single user mode with this quirk enabled.

4. `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.

5. `DiscardHibernateMap`

**Type:** `plist boolean`

**Failsafe:** Empty string

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

5. `MaxKernel`

**Type:** plist string

**Failsafe:** Empty string

**Description:** Adds kernel driver on 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 for 10.14.6. Kernel version interpretation is implemented as follows:

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

Kernel version comparison is implemented as follows:

$$\begin{aligned} \alpha &= \begin{cases} ParseDarwinVersion(\text{MinKernel}), & \text{If MinKernel is valid} \\ 0 & \text{Otherwise} \end{cases} \\ \beta &= \begin{cases} ParseDarwinVersion(\text{MaxKernel}), & \text{If MaxKernel is valid} \\ \infty & \text{Otherwise} \end{cases} \\ \gamma &= \begin{cases} ParseDarwinVersion(\text{FindDarwinVersion}()), & \text{If valid "Darwin Kernel Version" is found} \\ \infty & \text{Otherwise} \end{cases} \\ f(\alpha, \beta, \gamma) &= \alpha \leq \gamma \leq \beta \end{aligned}$$

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 "Darwin Kernel Version  $\kappa.\lambda.\mu$ " 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`).

# 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.
- **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 **KeySupportKeySupport**, 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
```

- 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~~ differently sized 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`