OpenCore

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

[2019.11.01]

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 Some of these features were originally implemented as a part of AptioMemoryFix.efi, which is no longer maintained.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 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. `MmioWhitelist`
   Type: `plist array`
   Description: Designed to be filled with `plist dict` values, describing addresses critical for particular firmware functioning when `DevirtualiseMmio` quirk is in use. See MmioWhitelist Properties section below.

2. `Quirks`
   Type: `plist dict`
   Description: Apply individual booter quirks described in Quirks Properties section below.

5.3 MmioWhitelist Properties

1. `Address`
   Type: `plist integer`
   Failsafe: `0`
   Description: Exceptional MMIO address, which memory descriptor should be left virtualised (unchanged) by `DevirtualiseMmio`. This means that the firmware will be able to directly communicate with this memory region during operating system functioning, because the region this value is in will be assigned a virtual address.

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. 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.

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

3. `AppleXcpmExtraMsrs`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables multiple MSR access critical for select CPUs, which have no native XCPM support.

   This is normally used in conjunction with `Emulate` section on Haswell-E, Broadwell-E, Skylake-X, and similar CPUs. More details on the XCPM patches are outlined in acidanthera/bugtracker#365.

   Note: Additional not provided patches will be required for Ivy Bridge or Pentium CPUs. It is recommended to use `AppleIntelCpuPowerManagement.kext` for the former.

4. `CustomSMBIOSGuid`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Performs GUID patching for `UpdateSMBIOSMode` `Custom` mode. Usually relevant for Dell laptops.

5. `DisableIoMapper`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables `IOMapper` support in XNU (VT-d), which may conflict with the firmware implementation.

   Note: This option is a preferred alternative to dropping `DMAR` ACPI table and disabling VT-d in firmware preferences, which does not break VT-d support in other systems in case they need it.

6. `ExternalDiskIcons`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

   Note: This option should avoided whenever possible. Modern firmwares usually have compatible AHCI controllers.

7. `LapicKernelPanic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables kernel panic on LAPIC interrupts.

8. `PanicNoKextDump`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

9. `PowerTimeoutKernelPanic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disables kernel panic on setPowerState timeout.

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

10. `ThirdPartyTrim`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Patch IOAHCIBlockStorage.kext to force TRIM command support on AHCI SSDs.

    Note: This option should avoided whenever possible. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called `trimforce`. Starting from 10.15 this utility creates `EnableTRIM` variable in `APPLE_BOOT_VARIABLE_GUID` namespace with `01 00 00 00` value.

11. `XhciPortLimit`
    Type: `plist boolean`
    Failsafe: `false`

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\Boot\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 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. Use 0 to disable timer.

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 `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.

File logging will create a file named `opencore-YYYY-MM-DD-HHMMSS.txt` at EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmwares are not reliable, and may corrupt data when writing files through UEFI. Log is attempted to be written in the safest manner, and thus is very slow. Ensure that `DisableWatchDog` is set to `true` when you use a slow drive.

8.5 Security Properties

1. `AllowNvramReset`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Allow `CMD+OPT+P+R` handling and enable showing `NVRAM Reset` entry in boot picker.

2. `ExposeSensitiveData`
   Type: `plist integer`
   Failsafe: ~~2~~0x6
   Description: Sensitive data exposure bitmask (sum) to operating system.

   - `0x01` — Expose printable booter path as an UEFI variable.
   - `0x02` — Expose OpenCore version as an UEFI variable.
   - `0x04` — Expose OpenCore version in boot picker menu title.

   Exposed booter path points to OpenCore.efi or its booter depending on the load order. To obtain booter path use the following command in macOS:

   ```
   nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path
   ```

   To use booter path for mounting booter volume use the following command in macOS:

   ```
   u=$(nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path | sed 's/.*GPT,\([^,]*\),.*/\1/'); \
     if [ "$u" != "" ]; then sudo diskutil mount $u ; fi
   ```

   To obtain OpenCore version use the following command in macOS:

   ```
   nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:opencore-version
   ```

3. `HaltLevel`
   Type: `plist integer`, 64 bit
   Failsafe: `0x80000000` (`DEBUG_ERROR`)
   Description: EDK II debug level bitmask (sum) causing CPU to halt (stop execution) after obtaining a message of `HaltLevel`. Possible values match `DisplayLevel` values.

4. `RequireSignature`
   Type: `plist boolean`
   Failsafe: `true`
   Description: Require `vault.sig` signature file for `vault.plist` in `OC` directory.

   This file should contain a raw 256 byte RSA-2048 signature from SHA-256 hash of `vault.plist`. The signature is verified against the public key embedded into `OpenCore.efi`.

   To embed the public key you should do either of the following:

   - Provide public key during the `OpenCore.efi` compilation in `OpenCoreVault.c` file.
   - Binary patch `OpenCore.efi` replacing zeroes with the public key between `=BEGIN OC VAULT=` and `==END OC VAULT==` ASCII markers.

   RSA public key 520 byte format description can be found in Chromium OS documentation. To convert public key from X.509 certificate or from PEM file use RsaTool.

   Note: `vault.sig` is used regardless of this option when public key is embedded into `OpenCore.efi`. Setting it to `true` will only ensure configuration sanity, and abort the boot process when public key is not set but was supposed to be used for verification.

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 ~~`KeySypport`~~`KeySupport`, 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
```

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. `ReconnectOnResChange`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reconnect console controllers after changing screen resolution.

   On some firmwares when screen resolution is changed via GOP, it is required to reconnect the controllers, which produce the console protocols (simple text out). Otherwise they will not produce text based on the new resolution.

   Note: On several boards this logic may result in black screen when launching OpenCore from Shell and thus it is optional. In versions prior to 0.5.2 this option was mandatory and not configurable. Please do not use this unless required.

8. `ReleaseUsbOwnership`
   Type: `plist boolean`
   Failsafe: `false`

12 Troubleshooting

12.1 Windows support

Can I install Windows?

While no official Windows support is provided, 64-bit UEFI Windows installations (Windows 8 and above) prepared with Boot Camp are supposed to work. Third-party UEFI installations as well as systems partially supporting UEFI boot, like Windows 7, might work with some extra precautions. Things to keep in mind:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- To install Windows, macOS, and OpenCore on the same drive you can specify Windows bootloader path (`\EFI\Microsoft\Boot\bootmgfw.efi`) in `BlessOverride` section.
- All the modifications applied (to ACPI, NVRAM, SMBIOS, etc.) are supposed to be operating system agnostic, i.e. apply equally regardless of the OS booted. This enables Boot Camp software experience on Windows.
- macOS requires the first partition to be EFI System Partition, and does not support the default Windows layout. While OpenCore does have a workaround for this, it is highly recommend not to rely on it and install properly.
- Windows may need to be reactivated. To avoid it consider ~~leaving SystemUUID field empty, so that~~ setting SystemUUID to the original firmware UUID~~is used~~. Be warned, on old firmwares it may be invalid, i.e. not random. In case you still have issues, consider using HWID or KMS38 license. The nuances of Windows activation are out of the scope of this document and can be found online.

What additional software do I need?

To enable operating system switching and install relevant drivers in the majority of cases you will need Windows support software from Boot Camp. For simplicity of the download process or when configuring an already installed Windows version a third-party utility, Brigadier, can be used successfully. Note, that you may have to download and install 7-Zip prior to using Brigadier.

Remember to always use the latest version of Windows support software from Boot Camp, as versions prior to 6.1 do not support APFS, and thus will not function correctly. To download newest software pass most recent Mac model to Brigadier, for example `./brigadier.exe -m iMac19,1`. To install Boot Camp on an unsupported Mac model afterwards run PowerShell as Administrator and enter `msiexec /i BootCamp.msi`. In case you already have a previous version of Boot Camp installed you will have to remove it first by running `msiexec /x BootCamp.msi` command. `BootCamp.msi` file is located in `BootCamp/Drivers/Apple` directory and can be reached through Windows Explorer.

While Windows support software from Boot Camp solves most of compatibility problems, sometimes you may have to address some of them manually:

- To invert mouse wheel scroll direction `FlipFlopWheel` must be set to `1` as explained on SuperUser.
- `RealTimeIsUniversal` must be set to `1` to avoid time desync between Windows and macOS as explained on SuperUser (this one is usually not needed).
- To access Apple filesystems like HFS and APFS separate software may need to be installed. Some of the known tools are: Apple HFS+ driver (hack for Windows 10), HFSExplorer, MacDrive, Paragon APFS, Paragon HFS+, TransMac, etc. Remember to never ever attempt to modify Apple file systems from Windows as this often leads to irrecoverable data loss.

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

Boot Camp control panel uses GPT partition table to obtain each boot option name. After installing Windows separately you will have to relabel the partition manually. This can be done with many tools including open-source gdisk utility. Reference example:

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

Command (? for help): p
Disk \\.\physicaldrive0: 419430400 sectors, 200.0 GiB
Sector size (logical): 512 bytes
Disk identifier (GUID): DEC57EB1-B3B5-49B2-95F5-3B8C4D3E4E12
```

- 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