OpenCore

Reference Manual (0.5.~~6~~.7)

[2020.04.03]

Contents

- `Resources`
  Directory used for storing media resources, such as audio files for screen reader support. See `UEFI Audio Properties` section for more details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  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`.
- `nvram.plist`
  OpenCore variable import file.
- `opencore-YYYY-MM-DD-HHMMSS.txt`
  OpenCore log file.

Note: It is not guaranteed that paths longer than `OC_STORAGE_SAFE_PATH_MAX` (128 characters including 0-termnator) will be accessible within OpenCore.

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 and MacInfoPkg.

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/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/MdeModulePkg
-I/UefiPackages/MdeModulePkg/Include
-I/UefiPackages/MdeModulePkg/Include/X64
-I/UefiPackages/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
-I/UefiPackages/MacInfoPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-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
-DNO_MSABI_VA_FUNCS=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.

3.4 Coding conventions

Just like any other project we have conventions that we follow during the development. All third-party contributors are highly recommended to read and follow the conventions listed below before submitting their patches. In general it is also recommended to firstly discuss the issue in Acidanthera Bugtracker before sending the patch to ensure no double work and to avoid your patch being rejected.

Organisation. The codebase is structured in multiple repositories which contain separate EDK II packages. `AppleSupportPkg` and `OpenCorePkg` are primary packages, and `EfiPkg`, `MacInfoPkg.dsc`) are dependent packages.

- Whenever changes are required in multiple repositories, separate pull requests should be sent to each.
- Committing the changes should happen firstly to dependent repositories, secondly to primary repositories to avoid automatic build errors.
- Each unique commit should compile with `XCODE5` and preferably with other toolchains. In the majority of the cases it can be checked by accessing the CI interface. Ensuring that static analysis finds no warnings is preferred.

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: `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.

`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 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`~~`OpenRuntime.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: `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. Examples of such hardware are Ivy Bridge laptops with Insyde firmware, like Acer V3-571G. Do not use this unless you fully understand the consequences.

6. `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.

7. `EnableWriteUnprotector`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Permit write access to UEFI runtime services code.

   This option bypasses `RX̂` 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`~~`OpenRuntime.efi`.

   Note: ~~The necessity of this quirk is determined by early boot crashes of the firmware~~ This quirk may potentially weaken firmware security, please use `RebuildAppleMemoryMap` if your firmware supports memory attributes table (MAT).

8. `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.

   Note: The necessity of this quirk is determined by early boot crashes of the firmware. Do not use this unless you fully understand the consequences.

9. `ProtectCsmRegion`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Protect CSM region areas from relocation.

   Ensure that CSM memory regions are marked as ACPI NVS to prevent ~~`boot.efi`~~ `boot.efi` or XNU from relocating or using them.

   Note: The necessity of this quirk is determined by artifacts and sleep wake issues. As `AvoidRuntimeDefrag` resolves a similar problem, no known firmwares should need this quirk. Do not use this unless you fully understand the consequences.

10. `ProtectSecureBoot`
    Type: `plist boolean`

Failsafe: `false`
Description: Protect UEFI Secure Boot variables from being written.

Reports security violation during attempts to write to `db`, `dbx`, `PK`, and `KEK` variables from the operating system.

Note: This quirk mainly attempts to avoid issues with NVRAM implementations with problematic defragmentation, such as select Insyde or `MacPro5,1`.

11. `ProtectUefiServices`
Type: `plist boolean`
Failsafe: `false`
Description: Protect UEFI services from being overridden by the firmware.

Some modern firmwares including both hardware and virtual machines, like VMware, may update pointers to UEFI services during driver loading and related actions. Consequentially this directly breaks other quirks that affect memory management, like `DevirtualiseMmio`, `ProtectCsmRegion`, or `ShrinkMemoryMap`, and may also break other quirks depending on the effects of these.

Note: On VMware the need for this quirk may be diagnosed by "Your Mac OS guest might run unreliably with more than one virtual core." message.

12. `ProvideCustomSlide`
Type: `plist boolean`
Failsafe: `false`
Description: Provide custom KASLR slide on low memory.

This option performs memory map analysis of your firmware and checks whether all slides (from `1` to `255`) can be used. As `boot.efi` generates this value randomly with `rdrand` or pseudo randomly `rdtsc`, there is a chance of boot failure when it chooses a conflicting slide. In case potential conflicts exist, this option forces macOS to use a pseudo random value among the available ones. This also ensures that `slide=` argument is never passed to the operating system for security reasons.

Note: The necessity of this quirk is determined by `OCABC: Only N/256 slide values are usable!` message in the debug log. If the message is present, this option is to be enabled.

13. `RebuildAppleMemoryMap`
Type: `plist boolean`
Failsafe: `false`
Description: Generate Memory Map compatible with macOS.

Apple kernel has several limitations in parsing UEFI memory map:

- Memory map size must not exceed 4096 bytes as Apple kernel maps it as a single 4K page. Since some firmwares have very large memory maps (approximately over 100 entries) Apple kernel will crash at boot.
- Memory attributes table is ignored. `EfiRuntimeServicesCode` memory statically gets `RX` permissions, and all other memory types get `RW` permissions. Since some firmware drivers may write to global variables at runtime, Apple kernel will crash at calling UEFI runtime services, unless driver `.data` section has `EfiRuntimeServicesData` type.

To workaround these limitations this quirk applies memory attributes table permissions to memory map passed to Apple kernel and optionally attempts to unify contiguous slots of similar types if the resulting memory map exceeds 4 KB.

Note: The necessity of this quirk is determined by early boot failures. This quirk replaces `EnableWriteUnprotector` on firmwares supporting memory attributes table (MAT).

14. `SetupVirtualMap`
Type: `plist boolean`
Failsafe: `false`
Description: Setup virtual memory at `SetVirtualAddresses`.

Select firmwares access memory by virtual addresses after `SetVirtualAddresses` call, which results in early boot crashes. This quirk workarounds the problem by performing early boot identity mapping of assigned virtual addresses to physical memory.

Note: The necessity of this quirk is determined by early boot failures. Currently new firmwares with memory protection support (like OVMF) do not support this quirk due to acidanthera/bugtracker#719.

15. ~~`ShrinkMemoryMap`~~**~~Type~~**~~: `plist boolean`~~**~~Failsafe~~**~~: `false`~~**~~Description~~**~~: Attempt to join similar memory map entries.~~

    ~~Select firmwares have very large memory maps, which do not fit Apple kernel, permitting up to `64` slots for runtime memory. This quirk attempts to unify contiguous slots of similar types to prevent boot failures.~~

    ~~*Note*: The necessity of this quirk is determined by early boot failures. It is rare to need this quirk on Haswell or newer. Do not use unless you fully understand the consequences.~~

16. `SignalAppleOS`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Report macOS being loaded through OS Info for any OS.

    This quirk is useful on Mac firmwares, which behave differently in different OS. For example, it is supposed to enable Intel GPU in Windows and Linux in some dual-GPU MacBook models.

17. `SyncRuntimePermissions`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Update memory permissions for `OpenRuntime` to function.

    Some firmwares may incorrectly mark `OpenRuntime` as not executable, this quirks updates memory map and memory attributes table to correct this.

    Note: The necessity of this quirk is determined by early boot failures either in macOS or in Linux/Windows. In general only firmwares released in 2018 or later are affected.

6 DeviceProperties

6.1 Introduction

Device configuration is provided to macOS with a dedicated buffer, called ~~EfiDevicePropertyDatabase~~EfiDevicePathPropertyDat
This buffer is a serialised map of DevicePaths to a map of property names and their values.

Property data can be debugged with gfxutil. To obtain current property data use the following command in macOS:

```
ioreg -lw0 -p IODeviceTree -n efi -r -x | grep device-properties |
  sed 's/.*<//;s/>.*//' > /tmp/device-properties.hex &&
  gfxutil /tmp/device-properties.hex /tmp/device-properties.plist &&
  cat /tmp/device-properties.plist
```

6.2 Properties

1. `Add`
   Type: `plist dict`
   Description: Sets device properties from a map (`plist dict`) of deivce paths to a map (`plist dict`) of variable names and their values in `plist metadata` format. Device paths must be provided in canonic string format (e.g. `PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x0)`). Properties will only be set if not present and not blocked.

   Note: Currently properties may only be (formerly) added by the original driver, so unless a separate driver was installed, there is no reason to block the variables.

2. `Block`
   Type: `plist dict`
   Description: Removes device properties from a map (`plist dict`) of deivce paths to an array (`plist array`) of variable names in `plist string` format.

6.3 Common Properties

Some known properties include:

- `device-id`
  User-specified device identifier used for I/O Kit matching. Has 4 byte data type.
- `vendor-id`
  User-specified vendor identifier used for I/O Kit matching. Has 4 byte data type.
- `AAPL,ig-platform-id`
  Intel GPU framebuffer identifier used for framebuffer selection on Ivy Bridge and newer. Has 4 byte data type.
- `AAPL,snb-platform-id`
  Intel GPU framebuffer identifier used for framebuffer selection on Sandy Bridge. Has 4 byte data type.
- `layout-id`
  Audio layout used for AppleHDA layout selection. Has 4 byte data type.

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

   - `None` — Avoid hibernation for your own good.
   - `Auto` — Use RTC and NVRAM detection.
   - `RTC` — Use RTC detection.
   - `NVRAM` — Use NVRAM detection.

2. `HideAuxiliary`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Hides auxiliary entries from picker menu by default.

   An entry is considered auxiliary when at least one of the following applies:

- Entry is macOS recovery.
- Entry is macOS Time Machine.
- Entry is explicitly marked as `Auxiliary`.
- Entry is system (e.g. `Clean NVRAM`).

To see all entries picker menu needs to be reloaded in extended mode by pressing `Spacebar` key. Hiding auxiliary entries may increase boot performance for multidisk systems.

3. `HideSelf`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Hides own boot entry from boot picker. This may potentially hide other entries, for instance, when another UEFI OS is installed on the same volume and driver boot is used.

4. `PickerAttributes`
   Type: `plist integer`
   Failsafe: `0`
   Description: Sets specific attributes for picker.

   Builtin picker supports colour arguments as a sum of foreground and background colors according to UEFI specification. The value of black background and black foreground (`0`) is reserved. List of colour names:

   - `0x00` — `EFI_BLACK`
   - `0x01` — `EFI_BLUE`
   - `0x02` — `EFI_GREEN`
   - `0x03` — `EFI_CYAN`
   - `0x04` — `EFI_RED`
   - `0x05` — `EFI_MAGENTA`
   - `0x06` — `EFI_BROWN`
   - `0x07` — `EFI_LIGHTGRAY`
   - `0x08` — `EFI_DARKGRAY`
   - `0x09` — `EFI_LIGHTBLUE`
   - `0x0A` — `EFI_LIGHTGREEN`
   - `0x0B` — `EFI_LIGHTCYAN`
   - `0x0C` — `EFI_LIGHTRED`
   - `0x0D` — `EFI_LIGHTMAGENTA`
   - `0x0E` — `EFI_YELLOW`
   - `0x0F` — `EFI_WHITE`
   - `0x00` — `EFI_BACKGROUND_BLACK`
   - `0x10` — `EFI_BACKGROUND_BLUE`
   - `0x20` — `EFI_BACKGROUND_GREEN`
   - `0x30` — `EFI_BACKGROUND_CYAN`
   - `0x40` — `EFI_BACKGROUND_RED`
   - `0x50` — `EFI_BACKGROUND_MAGENTA`
   - `0x60` — `EFI_BACKGROUND_BROWN`
   - `0x70` — `EFI_BACKGROUND_LIGHTGRAY`

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

5. `PickerAudioAssist`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable screen reader by default in boot picker.

   For macOS bootloader screen reader preference is set in `preferences.efires` archive in `isVOEnabled.int32` file and is controlled by the operating system. For OpenCore screen reader support this option is an independent equivalent. Toggling screen reader support in both OpenCore boot picker and macOS bootloader FileVault 2 login window can also be done with `Command` + `F5` key combination.

   Note: screen reader requires working audio support, see `UEFI Audio Properties` section for more details.

6. `PollAppleHotKeys`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable `modifier hotkey` handling in boot picker.

   In addition to `action hotkeys`, which are partially described in `PickerMode` section and are normally handled by Apple BDS, there exist modifier keys, which are handled by operating system bootloader, namely `boot.efi`. These keys allow to change operating system behaviour by providing different boot modes.

   On some firmwares it may be problematic to use modifier keys due to driver incompatibilities. To workaround this problem this option allows registering select hotkeys in a more permissive manner from within boot picker. Such extensions include the support of tapping on keys in addition to holding and pressing `Shift` along with other keys instead of just `Shift` alone, which is not detectible on many PS/2 keyboards. This list of known `modifier hotkeys` includes:

   - `CMD+C+MINUS` — disable board compatibility checking.
   - `CMD+K` — boot release kernel, similar to `kcsuffix=release`.
   - `CMD+S` — single user mode.
   - `CMD+S+MINUS` — disable KASLR slide, requires disabled SIP.
   - `CMD+V` — verbose mode.
   - `Shift` — safe mode.

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

8. `TakeoffDelay`
   Type: `plist integer`, 32 bit
   Failsafe: `0`
   Description: Delay in microseconds performed before handling picker startup and `action hotkeys`.

   Introducing a delay may give extra time to hold the right `action hotkey` sequence to e.g. boot to recovery mode. On some platforms setting this option to at least `5000-10000` microseconds may be necessary to access `action hotkeys` at all due to the nature of the keyboard driver.

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. `PickerMode`
    Type: `plist string`
    Failsafe: `Builtin`
    Description: Choose boot picker used for boot management.

    Picker describes underlying boot management with an optional user interface responsible for handling boot options. The following values are supported:

    - `Builtin` — boot management is handled by OpenCore, a simple text only user interface is used.
    - `External` — an external boot management protocol is used if available. Otherwise `Builtin` mode is used.
    - `Apple` — Apple boot management is used if available. Otherwise `Builtin` mode is used.

    Upon success `External` mode will entirely disable all boot management in OpenCore except policy enforcement. In `Apple` mode it may additionally bypass policy enforcement. ~~To implement `External` mode~~ See OpenCanopy plugin for an example of a custom user interface~~may utilise `OcBootManagementLib`. Reference example of external graphics interface is provided in 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 in general can be accessed by holding `action hotkeys` during boot process. Currently the following actions are considered:

- `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 1: Activated `KeySupport`, ~~`AppleUsbKbDxe`~~`OpenUsbKbDxe`, or similar driver is required for key handling to work. On many firmwares it is not possible to get all the keys function.

Note 2: In addition to `OPT` OpenCore supports `Escape` key to display picker when `ShowPicker` is disabled. This key exists for `Apple` picker mode and 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.

Note 3: On Macs with problematic GOP it may be difficult to access Apple BootPicker. To workaround this problem even without loading OpenCore `BootKicker` utility can be blessed.

8.4 Debug Properties

1. `AppleDebug`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable `boot.efi` debug log saving to OpenCore log.

   Note: This option only applies to 10.15.4 and newer.

2. `DisableWatchDog`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Select firmwares may not succeed in quickly booting the operating system, especially in debug mode, which results in watch dog timer aborting the process. This option turns off watch dog timer.

3. `DisplayDelay`
   Type: `plist integer`
   Failsafe: `0`
   Description: Delay in microseconds performed after every printed line visible onscreen (i.e. console).

4. `DisplayLevel`
   Type: `plist integer`, 64 bit
   Failsafe: `0`
   Description: EDK II debug level bitmask (sum) showed onscreen. Unless `Target` enables console (onscreen) printing, onscreen debug output will not be visible. The following levels are supported (discover more in DebugLib.h):

   - `0x00000002` (bit `1`) — `DEBUG_WARN` in `DEBUG`, `NOOPT`, `RELEASE`.
   - `0x00000040` (bit `6`) — `DEBUG_INFO` in `DEBUG`, `NOOPT`.
   - `0x00400000` (bit `22`) — `DEBUG_VERBOSE` in custom builds.
   - `0x80000000` (bit `31`) — `DEBUG_ERROR` in `DEBUG`, `NOOPT`, `RELEASE`.

5. `Target`
   Type: `plist integer`
   Failsafe: `0`
   Description: A bitmask (sum) of enabled logging targets. By default all the logging output is hidden, so this option is required to be set when debugging is necessary.

   The following logging targets are supported:

- 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. Auxiliary
   Type: plist boolean
   Failsafe: false
   Description: This entry will not be listed by default when HideAuxiliary is set to true.

3. 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.

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

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

6. 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: ~~Shell~~OpenShell.efi.

9 NVRAM

9.1 Introduction

Has `plist dict` type and allows to set volatile UEFI variables commonly referred as NVRAM variables. Refer to `man nvram` for more details. macOS extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication, and thus supplying several NVRAM is required for proper macOS functioning.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' NVRAM variable belongs to. macOS uses several GUIDs, including but not limited to:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14` (`APPLE_VENDOR_VARIABLE_GUID`)
- `7C436110-AB2A-4BBB-A880-FE41995C9F82` (`APPLE_BOOT_VARIABLE_GUID`)
- `8BE4DF61-93CA-11D2-AA0D-00E098032B8C` (`EFI_GLOBAL_VARIABLE_GUID`)
- `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` (`OC_VENDOR_VARIABLE_GUID`)

Note: Some of the variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Please ensure that variables of this section never collide with them, as behaviour is undefined otherwise.

For proper macOS functioning it is often required to use `OC_FIRMWARE_RUNTIME` protocol implementation currently offered as a part of ~~`FwRuntimeServices`~~`OpenRuntime` driver. While it brings any benefits, there are certain limitations which arise depending on the use.

1. Not all tools may be aware of protected namespaces.
   When `RequestBootVarRouting` is used `Boot`-prefixed variable access is restricted and protected in a separate namespace. To access the original variables tools have to be aware of `OC_FIRMWARE_RUNTIME` logic.
2. Assigned NVRAM variables are not always allowed to exceed 512 bytes.
   This is true for `Boot`-prefixed variables when `RequestBootVarFallback` is used, and for overwriting volatile variables with non-volatile on UEFI 2.8 non-conformant firmwares.

9.2 Properties

1. `Add`
   Type: `plist dict`
   Description: Sets NVRAM variables from a map (`plist dict`) of GUIDs to a map (`plist dict`) of variable names and their values in `plist metadata` format. GUIDs must be provided in canonic string format in upper or lower case (e.g. `8BE4DF61-93CA-11D2-AA0D-00E098032B8C`).

   Created variables get `EFI_VARIABLE_BOOTSERVICE_ACCESS` and `EFI_VARIABLE_RUNTIME_ACCESS` attributes set. Variables will only be set if not present and not blocked. To overwrite a variable add it to `Block` section. This approach enables to provide default values till the operating system takes the lead.

   Note: If `plist key` does not conform to GUID format, behaviour is undefined.
2. `Block`
   Type: `plist dict`
   Description: Removes NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.
3. `LegacyEnable`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enables loading of NVRAM variable file named `nvram.plist` from EFI volume root.

   This file must have root `plist dictionary` type and contain two fields:

   - `Version` — `plist integer`, file version, must be set to 1.
   - `Add` — `plist dictionary`, equivalent to `Add` from `config.plist`.

   Variable loading happens prior to `Block` (and `Add`) phases. Unless `LegacyOverwrite` is enabled, it will not overwrite any existing variable. Variables allowed to be set must be specified in `LegacySchema`. Third-party scripts may be used to create `nvram.plist` file. An example of such script can be found in `Utilities`. The use of

third-party scripts may require `ExposeSensitiveData` set to `0x3` to provide `boot-path` variable with OpenCore EFI partition UUID.

WARNING: This feature is very dangerous as it passes unprotected data to your firmware variable services. Use it only when no hardware NVRAM implementation is provided by the firmware or it is incompatible.

4. `LegacyOverwrite`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Permits overwriting firmware variables from `nvram.plist`.

   Note: Only variables accessible from the operating system will be overwritten.

5. `LegacySchema`
   Type: `plist dict`
   Description: Allows setting select NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

   You can use `*` value to accept all variables for select GUID.

   WARNING: Choose variables very carefully, as nvram.plist is not vaulted. For instance, do not put `boot-args` or `csr-active-config`, as this can bypass SIP.

6. `WriteFlash`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enables writing to flash memory for all added variables.

   Note: This value is recommended to be enabled on most firmwares, but is left configurable for firmwares that may have issues with NVRAM variable storage garbage collection or alike.

To read NVRAM variable value from macOS one could use `nvram` by concatenating variable GUID and name separated by `:` symbol. For example, `nvram 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args`.

A continuously updated variable list can be found in a corresponding document: NVRAM Variables.

9.3 Mandatory Variables

Warning: These variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Using `PlatformInfo` is the recommend way of setting these variables.

The following variables are mandatory for macOS functioning:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeatures`
  32-bit `FirmwareFeatures`. Present on all Macs to avoid extra parsing of SMBIOS tables
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:FirmwareFeaturesMask`
  32-bit `FirmwareFeaturesMask`. Present on all Macs to avoid extra parsing of SMBIOS tables.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:MLB`
  `BoardSerialNumber`. Present on newer Macs (2013+ at least) to avoid extra parsing of SMBIOS tables, especially in ~~boot. efi.~~ `boot.efi`.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM`
  Primary network adapter MAC address or replacement value. Present on newer Macs (2013+ at least) to avoid accessing special memory region, especially in ~~boot. efi.~~ `boot.efi`.

9.4 Recommended Variables

The following variables are recommended for faster startup or other improvements:

- `7C436110-AB2A-4BBB-A880-FE41995C9F82:csr-active-config`
  32-bit System Integrity Protection bitmask. Declared in XNU source code in csr.h.
- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeatures`
  Combined `FirmwareFeatures` and `ExtendedFirmwareFeatures`. Present on newer Macs to avoid extra parsing of SMBIOS tables

- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ExtendedFirmwareFeaturesMask
  Combined FirmwareFeaturesMask and ExtendedFirmwareFeaturesMask. Present on newer Macs to avoid extra parsing of SMBIOS tables.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_BID
  Hardware BoardProduct (e.g. Mac-35C1E88140C3E6CF). Not present on real Macs, but used to avoid extra parsing of SMBIOS tables, especially in ~~boot. efi.~~ boot.efi.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_MLB
  Hardware BoardSerialNumber. Override for MLB. Present on newer Macs (2013+ at least).
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:HW_ROM
  Hardware ROM. Override for ROM. Present on newer Macs (2013+ at least).
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:prev-lang:kbd
  ASCII string defining default keyboard layout. Format is lang-COUNTRY:keyboard, e.g. ru-RU:252 for Russian locale and ABC keyboard. Also accepts short forms: ru:252 or ru:0 (U.S. keyboard, compatible with 10.9). Full decoded keyboard list from AppleKeyboardLayouts-L.dat can be found here. Using non-latin keyboard on 10.14 will not enable ABC keyboard, unlike previous and subsequent macOS versions, and is thus not recommended in case you need 10.14.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:security-mode
  ASCII string defining FireWire security mode. Legacy, can be found in IOFireWireFamily source code in IOFireWireController.cpp. It is recommended not to set this variable, which may speedup system startup. Setting to full is equivalent to not setting the variable and none disables FireWire security.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:UIScale
  One-byte data defining ~~boot.efi~~ boot.efi user interface scaling. Should be **01** for normal screens and **02** for HiDPI screens.
- 4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:DefaultBackgroundColor
  Four-byte RGBA data defining ~~boot.efi~~ boot.efi user interface background colour. Standard colours include **BF BF BF 00** (Light Gray) and **00 00 00 00** (Syrah Black). Other colours may be set at user's preference.

9.5 Other Variables

The following variables may be useful for certain configurations or troubleshooting:

- 7C436110-AB2A-4BBB-A880-FE41995C9F82:boot-args
  Kernel arguments, used to pass configuration to Apple kernel and drivers. There are many arguments, which may be found by looking for the use of PE_parse_boot_argn function in the kernel or driver code. Some of the known boot arguments include:
  - acpi_layer=0xFFFFFFFF
  - acpi_level=0xFFFF5F (implies ACPI_ALL_COMPONENTS)
  - batman=VALUE (AppleSmartBatteryManager debug mask)
  - batman-nosmc=1 (disable AppleSmartBatteryManager SMC interface)
  - cpus=VALUE (maximum number of CPUs used)
  - debug=VALUE (debug mask)
  - io=VALUE (IOKit debug mask)
  - keepsyms=1 (show panic log debug symbols)
  - kextlog=VALUE (kernel extension loading debug mask)
  - nv_disable=1 (disables NVIDIA GPU acceleration)
  - nvda_drv=1 (legacy way to enable NVIDIA web driver, removed in 10.12)
  - npci=0x2000 (legacy, disables kIOPCIConfiguratorPFM64)
  - lapic_dont_panic=1
  - slide=VALUE (manually set KASLR slide)
  - smcdebug=VALUE (AppleSMC debug mask)
  - -amd_no_dgpu_accel (alternative to WhateverGreen's -radvesa for new GPUs)
  - -nehalem_error_disable
  - -no_compat_check (disable model checking)
  - -s (single mode)
  - -v (verbose mode)
  - -x (safe mode)

  There are multiple external places summarising macOS argument lists: example 1, example 2.
- 7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg

    - 2 — AppleLoggingStdErrSet/AppleLoggingStdErrPrint (StdErr or serial?)
    - 4 — AppleLoggingFileSet/AppleLoggingFilePrint (BOOTER.LOG/BOOTER.OLD file on EFI partition)
  - `debug=VALUE` — deprecated starting from 10.15.
    - 1 — enables print something to BOOTER.LOG (stripped code implies there may be a crash)
    - 2 — enables perf logging to /efi/debug-log in the device three
    - 4 — enables timestamp printing for styled printf calls
  - `level=VALUE` — deprecated starting from 10.15. Verbosity level of DEBUG output. Everything but `0x80000000` is stripped from the binary, and this is the default value.

  Note: To ~~quickly~~ see verbose output from `boot.efi` on ~~versions~~ modern macOS versions enable `AppleDebug` option. This will save the log to general OpenCore log. For versions before 10.15.4 set `bootercfg` to `log=1`. This will print verbose output onscreen.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:bootercfg-once`
  Booter arguments override removed after first launch. Otherwise equivalent to `bootercfg`.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:efiboot-perf-record`
  Enable performance log saving in `boot.efi`. Performance log is saved to physical memory and is pointed by `efiboot-perf-record-data` and `efiboot-perf-record-size` variables. Starting from 10.15.4 it can also be saved to OpenCore log by `AppleDebug` option.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:fmm-computer-name`
  Current saved host name. ASCII string.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:nvda_drv`
  NVIDIA Web Driver control variable. Takes ASCII digit `1` or `0` to enable or disable installed driver.
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:StartupMute`
  Mute startup chime sound in firmware audio support. 8-bit integer. The value of `0x00` means unmuted. Missing variable or any other value means muted. This variable only affects Gibraltar machines (T2).
- `7C436110-AB2A-4BBB-A880-FE41995C9F82:SystemAudioVolume`
  System audio volume level for firmware audio support. 8-bit integer. The bit of `0x80` means muted. Lower bits are used to encode volume range specific to installed audio codec. The value is capped by `MaximumBootBeepVolume` AppleHDA layout value to avoid too loud audio playback in the firmware.

10 PlatformInfo

Platform information is comprised of several identification fields generated or filled manually to be compatible with macOS services. The base part of the configuration may be obtained from `MacInfoPkg` package, which itself generates a set of interfaces based on a database in YAML format. These fields are written to three select destinations:

- SMBIOS
- Data Hub
- NVRAM

Most of the fields specify the overrides in SMBIOS, and their field names conform to EDK2 SmBios.h header file. However, several important fields reside in Data Hub and NVRAM. Some of the values can be found in more than one field and/or destination, so there are two ways to control their update process: manual, where one specifies all the values (the default), and semi-automatic, where (`Automatic`) only select values are specified, and later used for system configuration.

To inspect SMBIOS contents dmidecode utility can be used. Version with macOS specific enhancements can be downloaded from Acidanthera/dmidecode.

10.1 Properties

1. `Automatic`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Generate PlatformInfo based on `Generic` section instead of using values from `DataHub`, `NVRAM`, and `SMBIOS` sections.

   Enabling this option is useful when `Generic` section is flexible enough. When enabled `SMBIOS`, `DataHub`, and `PlatformNVRAM` data is unused.

2. `UpdateDataHub`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update Data Hub fields. These fields are read from `Generic` or `DataHub` sections depending on `Automatic` value.

3. `UpdateNVRAM`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update NVRAM fields related to platform information.

   These fields are read from `Generic` or `PlatformNVRAM` sections depending on `Automatic` value. All the other fields are to be specified with `NVRAM` section.

   If `UpdateNVRAM` is set to `false` the aforementioned variables can be updated with `NVRAM` section. If `UpdateNVRAM` is set to `true` the behaviour is undefined when any of the fields are present in `NVRAM` section.

4. `UpdateSMBIOS`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Update SMBIOS fields. These fields are read from `Generic` or `SMBIOS` sections depending on `Automatic` value.

5. `UpdateSMBIOSMode`
   Type: `plist string`
   Failsafe: `Create`
   Description: Update SMBIOS fields approach:

   - `TryOverwrite` — `Overwrite` if new size is <= than the page-aligned original and there are no issues with legacy region unlock. `Create` otherwise. Has issues with some firmwares.
   - `Create` — Replace the tables with newly allocated EfiReservedMemoryType at AllocateMaxAddress without any fallbacks.

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 PropertiesDrivers

AudioType: plist dictFailsafe: NoneDescription: Configure audio backend support described in section below.

Audio support provides a way for upstream protocols to interact with the selected hardware and audio resources. All audio resources should reside in \EFI\OC\Resources\Audio directory. Currently the only supported audio file format is WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: audio type_audio localisation_audio path.wav. For unlocalised files filename does not include the language code and looks as follows: audio type_audio path.wav.

- Audio type can be OCEFIAudio for OpenCore audio files or AXEFIAudio for macOS bootloader audio files.
- Audio localisation is a two letter language code (e.g. en) with an exception for Chinese, Spanish, and Portuguese. Refer to for the list of all supported localisations.
- Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to . For OpenCore audio paths refer to . The only exception is OpenCore boot chime file, which is OCEFIAudio_VoiceOver_Boot.wav.

Audio localisation is determined separately for macOS bootloader and OpenCore. For macOS bootloader it is set in preferences.efires archive in systemLanguage.utf8 file and is controlled by the operating system. For OpenCore the value of prev-lang:kbd variable is used. When native audio localisation of a particular file is missing, English language (en) localisation is used. Sample audio files can be found in .

ConnectDriversType: plist booleanFailsafe: falseDescription: Perform UEFI controller connection after driver loading.

This option is useful for loading drivers following UEFI driver model as they may not start by themselves. Examples of such drivers are filesystem or audio drivers. While effective, this option may not be necessary for drivers performing automatic connection, and may slightly slowdown the boot.

Note: Some firmwares, made by Apple in particular, only connect the boot drive to speedup the boot process. Enable this option to be able to see all the boot options when having multiple drives.

DriversType: plist arrayFailsafe: NoneDescription: 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 includeare listed below:

ApfsDriverLoader	APFS file system bootstrap driver adding the support of embedded APFS drivers in bootable APFS containers in UEFI firmwares.
AudioDxe	HDA audio support driver in UEFI firmwares for most Intel and some other analog audio controllers. Refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
ExFatDxe	Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmwares. For Sandy Bridge and earlier CPUs ExFatDxeLegacy driver should be used due to the lack of RDRAND instruction support.
HfsPlus	Proprietary HFS file system driver with bless support commonly found in Apple firmwares. For Sandy Bridge and earlier CPUs HfsPlusLegacy driver should be used due to the lack of RDRAND instruction support.
HiiDatabase*	HII services support driver from MdeModulePkg. This driver is included in most firmwares starting with Ivy Bridge generation. Some applications with the GUI like UEFI Shell may need this driver to work properly.
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.
OpenCanopy*	OpenCore plugin implementing graphical interface.
OpenRuntime*	OpenCore plugin implementing OC_FIRMWARE_RUNTIME protocol.
OpenUsbKbDxe*	USB keyboard driver adding the support of AppleKeyMapAggregator protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin KeySupport, which may work better or worse depending on the firmware.
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.

Driver marked with * are bundled with OpenCore. 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
```

11.3 Tools

Standalone tools may help to debug firmware and hardware. Some of the known tools are listed below. While some tools can be launched from within OpenCore many should be run separately either directly or from `Shell`.

To boot into OpenShell or any other tool directly save `OpenShell.efi` under the name of `EFI\BOOT\BOOTX64.EFI` on a FAT32 partition. In general it is unimportant whether the partitition scheme is `GPT` or `MBR`.

While the previous approach works both on Macs and other computers, an alternative Mac-only approach to bless the tool on an HFS+ or APFS volume:

```
sudo bless --verbose --file /Volumes/VOLNAME/DIR/OpenShell.efi \
  --folder /Volumes/VOLNAME/DIR/ --setBoot
```

Listing 3: Blessing tool

Note 1: You may have to copy `/System/Library/CoreServices/BridgeVersion.bin` to `/Volumes/VOLNAME/DIR`.
Note 2: To be able to use `bless` you may have to disable System Integrity Protection.
Note 3: To be able to boot you may have to disable Secure Boot if present.

Some of the known tools are listed below (builtin tools are marked with *):

`BootKicker`*	Enter Apple BootPicker menu (exclusive for Macs with compatible GPUs).
`ChipTune`*	Test BeepGen protocol and generate audio signals of different style and length.
`CleanNvram`*	Reset NVRAM alternative bundled as a standalone tool.
`GopStop`*	Test GraphicsOutput protocol with a simple scenario.
`HdaCodecDump`*	Parse and dump High Definition Audio codec information (requires `AudioDxe`).
`KeyTester`*	Test keyboard input in `SimpleText` mode.
`MemTest86`	Memory testing utility.
`OpenControl`*	Unlock and lock back NVRAM protection for other tools to be able to get full NVRAM access when launching from OpenCore.
`OpenShell`*	OpenCore-configured `UEFI Shell` for compatibility with a broad range of firmwares.
`PavpProvision`	Perform EPID provisioning (requires certificate data configuration).
`VerifyMsrE2`*	Check `CFG Lock` (MSR `0xE2` write protection) consistency across all cores.

11.4 OpenCanopy

OpenCanopy is a graphical OpenCore user interface that runs in `External` `PickerMode` and relies on OpenCorePkg `OcBootManagementLib` similar to the builtin text interface.

OpenCanopy requires graphical resources located in `Resources` directory to run. Sample resources (fonts and images) can be found in OcBinaryData repository.

Note: OpenCanopy is currently considered experimental and is not recommended for everyday use. Refer to acidanthera/bugtracker#759 for more details regarding the current limitations.

11.5 OpenRuntime

`OpenRuntime` is an OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol. This protocol implements multiple features required for OpenCore that are otherwise not possible to implement in OpenCore itself as they are needed to work in runtime, i.e. during operating system functioning. Feature highlights:

- NVRAM namespaces, allowing to isolate operating systems from accessing select variables (e.g. `RequestBootVarRouting` or `ProtectSecureBoot`).
- NVRAM proxying, allowing to manipulate multiple variables on variable updates (e.g. `RequestBootVarFallback`).

- Read-only and write-only NVRAM variables, enhancing the security of OpenCore, Lilu, and Lilu plugins, like VirtualSMC, which implements `AuthRestart` support.
- NVRAM isolation, allowing to protect all variables from being written from an untrusted operating system (e.g. `DisableVariableWrite`).
- UEFI Runtime Services memory protection management to workaround read-only mapping (e.g. `EnableWriteUnprotector`).

11.6 Properties

1. `Audio`
   Type: `plist dict`
   Failsafe: None
   Description: Configure audio backend support described in Audio Properties section below.

   Audio support provides a way for upstream protocols to interact with the selected hardware and audio resources. All audio resources should reside in `\EFI\OC\Resources\Audio` directory. Currently the only supported audio file format is WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

   Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: `[audio type]_[audio localisation]_[audio path].wav`. For unlocalised files filename does not include the language code and looks as follows: `[audio type]_[audio path].wav`.

   - Audio type can be `OCEFIAudio` for OpenCore audio files or `AXEFIAudio` for macOS bootloader audio files.
   - Audio localisation is a two letter language code (e.g. `en`) with an exception for Chinese, Spanish, and Portuguese. Refer to `APPLE_VOICE_OVER_LANGUAGE_CODE` definition for the list of all supported localisations.
   - Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to `APPLE_VOICE_OVER_AUDIO_FILE` definition. For OpenCore audio paths refer to `OC_VOICE_OVER_AUDIO_FILE` definition. The only exception is OpenCore boot chime file, which is `OCEFIAudio_VoiceOver_Boot.wav`.

   Audio localisation is determined separately for macOS bootloader and OpenCore. For macOS bootloader it is set in `preferences.efires` archive in `systemLanguage.utf8` file and is controlled by the operating system. For OpenCore the value of `prev-lang:kbd` variable is used. When native audio localisation of a particular file is missing, English language (`en`) localisation is used. Sample audio files can be found in OcBinaryData repository.

2. `ConnectDrivers`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform UEFI controller connection after driver loading.

   This option is useful for loading drivers following UEFI driver model as they may not start by themselves. Examples of such drivers are filesystem or audio drivers. While effective, this option may not be necessary for drivers performing automatic connection, and may slightly slowdown the boot.

   Note: Some firmwares, made by Apple in particular, only connect the boot drive to speedup the boot process. Enable this option to be able to see all the boot options when having multiple drives.

3. `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.

4. `Input`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual settings designed for input (keyboard and mouse) in Input Properties section below.

5. `Output`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual settings designed for output (text and graphics) in Output Properties section below.

6. `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.

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

11.7 Audio Properties

1. `AudioCodec`
   Type: `plist integer`
   Failsafe: ~~empty string~~0
   Description: Codec address on the specified audio controller for audio support.

   Normally this contains first audio codec address on the builtin analog audio controller (`HDEF`). Audio codec addresses, e.g. `2`, can be found in the debug log (marked in bold):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   As an alternative this value can be obtained from `IOHDACodecDevice` class in I/O Registry containing it in `IOHDACodecAddress` field.

2. `AudioDevice`
   Type: `plist string`
   Failsafe: ~~0~~empty string
   Description: Device path of the specified audio controller for audio support.

   Normally this contains builtin analog audio controller (`HDEF`) device path, e.g. `PciRoot(0x0)/Pci(0x1b,0x0)`. The list of recognised audio controllers can be found in the debug log (marked in bold):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   As an alternative `gfxutil -f HDEF` command can be used in macOS. Specifying empty device path will result in the first available audio controller to be used.

3. `AudioOut`
   Type: `plist integer`
   Failsafe: `0`
   Description: Index of the output port of the specified codec starting from 0.

   Normally this contains the index of the green out of the builtin analog audio controller (`HDEF`). The number of output nodes (`N`) in the debug log (marked in bold):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   The quickest way to find the right port is to bruteforce the values from `0` to `N - 1`.

4. `AudioSupport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Activate audio support by connecting to a backend driver.

   Enabling this setting routes audio playback from builtin protocols to a dedicated audio port (`AudioOut`) of the specified codec (`AudioCodec`) located on the audio controller (`AudioDevice`).

5. `MinimumVolume`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal heard volume level from `0` to `100`.

   Screen reader will use this volume level, when the calculated volume level is less than `MinimumVolume`. Boot chime sound will not play if the calculated volume level is less than `MinimumVolume`.

6. `PlayChime`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Play chime sound at startup.

   Enabling this setting plays boot chime through builtin audio support. Volume level is determined by `MinimumVolume` and `VolumeAmplifier` settings and `SystemAudioVolume` NVRAM variable.

   Note: this setting is separate from `StartupMute` NVRAM variable to avoid conflicts when the firmware is able to play boot chime.

7. `VolumeAmplifier`
   Type: `plist integer`
   Failsafe: `0`
   Description: Multiplication coefficient for system volume to raw volume linear translation from `0` to `1000`.

   Volume level range read from `SystemAudioVolume` varies depending on the codec. To transform read value in `[0, 127]` range into raw volume range `[0, 100]` the read value is scaled to `VolumeAmplifier` percents:

$$RawVolume = MIN(\frac{SystemAudioVolume * VolumeAmplifier}{100}, 100)$$

   Note: the transformation used in macOS is not linear, but it is very close and this nuance is thus ignored.

11.8 Input Properties

1. `KeyFiltering`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable keyboard input sanity checking.

   Apparently some boards like GA Z77P-D3 may return uninitialised data in `EFI_INPUT_KEY` with all input protocols. This option discards keys that are neither ASCII, nor are defined in the UEFI specification (see tables 107 and 108 in version 2.8).

2. `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.

3. 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.

4. 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 (AptioInputFix), to fill AppleKeyMapAggregator database for input functioning. In case a separate driver is used, such as ~~AppleUsbKbDxe~~OpenUsbKbDxe, this option should never be enabled.

5. 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.

   Note: Currently V1, V2, and AMI unlike Auto only do filtering of the particular specified protocol. This may change in the future versions.

6. 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.

7. 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.

8. 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.

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

Note: This will fail when console handle has no GOP protocol. When the firmware does not provide it, it can be added with `ProvideConsoleGop` set to `true`.

4. `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: This option only applies to `System` renderer.

5. `DirectGopCacheMode`
   Type: `plist string`
   Failsafe: Empty string
   Description: Cache mode for builtin graphics output protocol framebuffer.

   Tuning cache mode may provide better rendering performance on some firmwares. Providing empty string leaves cache control settings to the firmware. Valid non-empty values are: `Uncacheable`, `WriteCombining`, and `WriteThrough`.

   Note: This option is not supported on most hardware (see acidanthera/bugtracker#755 for more details).

6. `DirectGopRendering`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Use builtin graphics output protocol renderer for console.

   On some firmwares this may provide better performance or even fix rendering issues, like on `MacPro5,1`. However, it is recommended not to use this option unless there is an obvious benefit as it may even result in slower scrolling.

7. `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: This option only applies to `System` renderer.

8. `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: This option only applies to `System` renderer.

9. `ProvideConsoleGop`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Ensure GOP (Graphics Output Protocol) on console handle.

   macOS bootloader requires GOP to be present on console handle, yet the exact location of GOP is not covered by the UEFI specification. This option will ensure GOP is installed on console handle if it is present.

   Note: This option will also replace broken GOP protocol on console handle, which may be the case on `MacPro5,1` with newer GPUs.

10. `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.

11. `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: This option only applies to `System` renderer. On all known affected systems `ConsoleMode` had to be set to empty string for this to work.

11.10 Protocols Properties

1. `AppleAudio`
Type: `plist boolean`
Failsafe: `false`
Description: Reinstalls Apple audio protocols with builtin versions.

Apple audio protocols allow macOS bootloader and OpenCore to play sounds and signals for screen reading or audible error reporting. Supported protocols are beep generation and VoiceOver. VoiceOver protocol is specific to Gibraltar machines (T2) and is not supported before macOS High Sierra (10.13). Instead older macOS versions use AppleHDA protocol, which is currently not implemented.

Only one set of audio protocols can be available at a time, so in order to get audio playback in OpenCore user interface on Mac system implementing some of these protocols this setting should be enabled.

Note: Backend audio driver needs to be configured in `UEFI Audio` section for these protocols to be able to stream audio.

2. `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.

Note: Some Macs, namely `MacPro5,1`, do have APFS compatibility, but their Apple Boot Policy protocol contains recovery detection issues, thus using this option is advised on them as well.

3. `AppleDebugLog`
Type: `plist boolean`
Failsafe: `false`
Description: Reinstalls Apple Debug Log protocol with a builtin version.

4. `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.

5. `AppleImageConversion`
Type: `plist boolean`
Failsafe: `false`
Description: Reinstalls Apple Image Conversion protocol with a builtin version.

6. `AppleKeyMap`
Type: `plist boolean`
Failsafe: `false`
Description: Reinstalls Apple Key Map protocols with builtin versions.

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.

2. `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.

3. `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.

4. `RequestBootVarFallback`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Request fallback of some `Boot` prefixed variables from `OC_VENDOR_VARIABLE_GUID` to `EFI_GLOBAL_VARIABLE_GUID`.

   This quirk requires `RequestBootVarRouting` to be enabled and therefore `OC_FIRMWARE_RUNTIME` protocol implemented in ~~`FwRuntimeServices`~~`OpenRuntime.efi`.

   By redirecting `Boot` prefixed variables to a separate GUID namespace we achieve multiple goals:

   - Operating systems are jailed and only controlled by OpenCore boot environment to enhance security.
   - Operating systems do not mess with OpenCore boot priority, and guarantee fluent updates and hibernation wakes for cases that require reboots with OpenCore in the middle.
   - Potentially incompatible boot entries, such as macOS entries, are not deleted or anyhow corrupted.

   However, some firmwares do their own boot option scanning upon startup by checking file presence on the available disks. Quite often this scanning includes non-standard locations, such as Windows Bootloader paths. Normally it is not an issue, but some firmwares, ASUS firmwares on APTIO V in particular, have bugs. For them scanning is implemented improperly, and firmware preferences may get accidentally corrupted due to `BootOrder` entry duplication (each option will be added twice) making it impossible to boot without cleaning NVRAM.

   To trigger the bug one should have some valid boot options (e.g. OpenCore) and then install Windows with `RequestBootVarRouting` enabled. As Windows bootloader option will not be created by Windows installer, the firmware will attempt to create it itself, and then corrupt its boot option list.

   This quirk forwards all UEFI specification valid boot options, that are not related to macOS, to the firmware into `BootF###` and `BootOrder` variables upon write. As the entries are added to the end of `BootOrder`, this does not break boot priority, but ensures that the firmware does not try to append a new option on its own after Windows installation for instance.

5. `RequestBootVarRouting`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Request redirect of all `Boot` prefixed variables from `EFI_GLOBAL_VARIABLE_GUID` to `OC_VENDOR_VARIABLE_GUID`.

   This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in ~~`FwRuntimeServices`~~`OpenRuntime.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.

6. `UnblockFsConnect`
   Type: `plist boolean`
   Failsafe: `false`

```
Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 419430366
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048         1023999   499.0 MiB   2700  Basic data partition
   2         1024000         1226751   99.0 MiB    EF00  EFI system partition
   3         1226752         1259519   16.0 MiB    0C01  Microsoft reserved ...
   4         1259520       419428351   199.4 GiB   0700  Basic data partition

Command (? for help): c
Partition number (1-4): 4
Enter name: BOOTCAMP

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING PARTITIONS!!

Do you want to proceed? (Y/N): Y
OK; writing new GUID partition table (GPT) to \\.\physicaldrive0.
Disk synchronization succeeded! The computer should now use the new partition table.
The operation has completed successfully.
```

Listing 4: Relabeling Windows volume

How 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 OpenCore 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 `RX`, and motherboard `GND` to USB UART `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. Apart from App Store and softwareupdate utility there also are third-party tools to download an offline image.

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.

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 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~~OpenRuntime 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