OpenCore

Reference Manual (0.6.5.6)

[2021.01.31]

3 Setup

3.1 Directory Structure

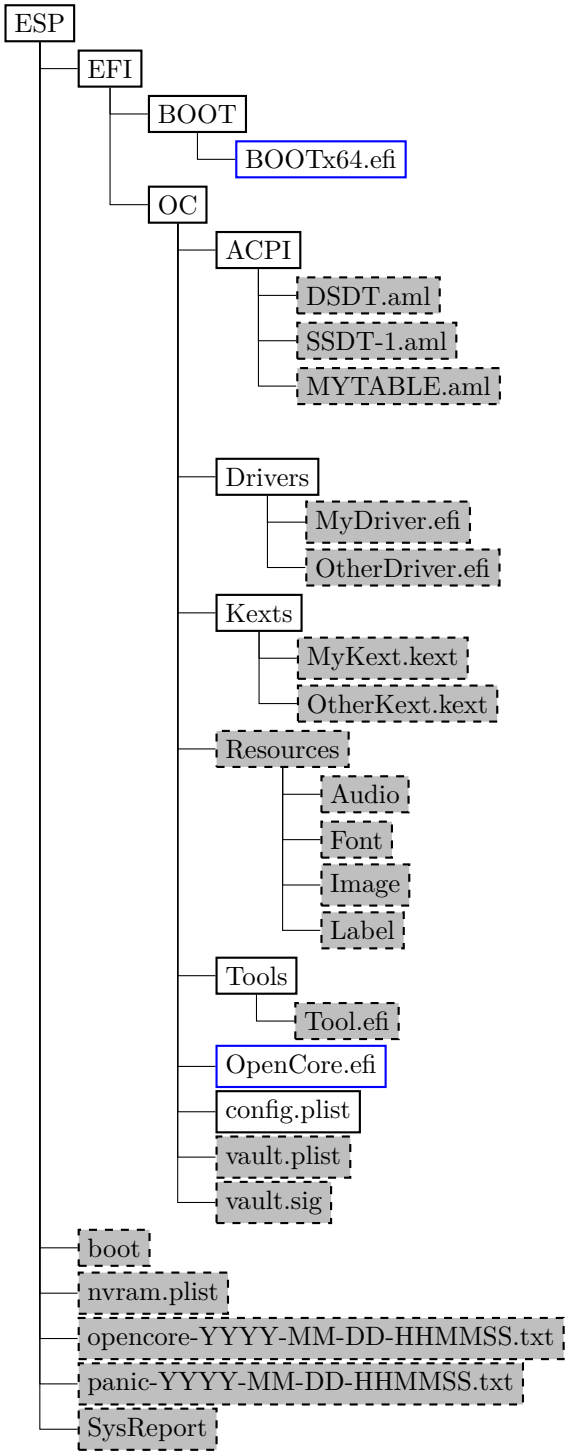

Figure 1. Directory Structure

When directory boot is used the directory structure used should follow the description on Directory Structure figure. Available entries include:

- `BOOTx64.efi` ~~and~~ or ~~`Bootstrap`~~`BOOTIa32.efi`
  Initial bootstrap loaders, which ~~loads~~ load `OpenCore.efi`~~unless it was already started as a driver~~. `BOOTx64.efi` is loaded by the firmware by default according to UEFI specification, ~~and `Bootstrap.efi` can be registered as a custom option~~ yet it can also be renamed and put to a custom location to let OpenCore coexist with operating

systems using `BOOTx64.efi` as their own loaders (e.g. Windows), see ~~`BootProtect`~~`LauncherOption` for more details.

- `boot`
  Duet bootstrap loader, which initialises UEFI environment on legacy BIOS firmware and loads `OpenCore.efi` similarly to other bootstrap loaders. Modern Duet bootstrap loader will default to `OpenCore.efi` on the same partition when present.
- `ACPI`
  Directory used for storing supplemental ACPI information for `ACPI` section.
- `Drivers`
  Directory used for storing supplemental `UEFI` drivers for `UEFI` section.
- `Kexts`
  Directory used for storing supplemental kernel information for `Kernel` section.
- `Resources`
  Directory used for storing media resources, such as audio files for screen reader support. See `UEFI Audio Properties` section for more details. This directory also contains image files for graphical user interface. See OpenCanopy section for more details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  Main booter ~~driver~~ application responsible for operating system loading. The directory `OpenCore.efi` resides is called the `root directory`. By default `root directory` is set to `EFI\OC`, however, when launching `OpenCore.efi` directly or through ~~`Bootstrap.efi`~~a custom launcher, other directories containing `OpenCore.efi` can also be supported.
- `config.plist`
  `OC Config`.
- `vault.plist`
  Hashes for all files potentially loadable by `OC Config`.
- `vault.sig`
  Signature for `vault.plist`.
- `SysReport`
  Directory containing system reports generated by `SysReport` option.
- `nvram.plist`
  OpenCore variable import file.
- `opencore-YYYY-MM-DD-HHMMSS.txt`
  OpenCore log file.
- `panic-YYYY-MM-DD-HHMMSS.txt`
  Kernel panic log file.

Note: It is not guaranteed that paths longer than `OC_STORAGE_SAFE_PATH_MAX` (128 characters including `0`-terminator) 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 regarding external resources such as 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. `OpenDuetPkg` is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system, `OpenDuetPkg` can be installed with a dedicated tool — BootInstall (bundled with OpenCore). Third-party utilities can be used to perform this on systems other than macOS.

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

Codestyle. The codebase follows EDK II codestyle with few changes and clarifications.

- Write inline documentation for the functions and variables only once: in headers, where a header prototype is available, and inline for `static` variables and functions.
- Use line length of 120 characters or less, preferably 100 characters.
- Use spaces after casts, e.g. `(VOID *)(UINTN) Variable`.
- Use two spaces to indent function arguments when splitting lines.
- Prefix public functions with either `Oc` or another distinct name.
- Do not prefix private `static` functions, but use `Internal` for `non-static`.
- Use SPDX license headers as shown in acidanthera/bugtracker#483.

3.5 Debugging

The codebase incorporates EDK II debugging and few custom features to improve the experience.

- Use module prefixes, 2-5 letters followed by a colon (`:`), for debug messages. For `OpenCorePkg` use `OC:`, for libraries and drivers use their own unique prefixes.
- Do not use dots (`.`) in the end of debug messages and separate `EFI_STATUS`, printed by `%r`, with a hyphen (e.g. `OCRAM: Allocation of %u bytes failed - %r\n`).
- Use `DEBUG_CODE_BEGIN ()` and `DEBUG_CODE_END ()` constructions to guard debug checks that may potentially reduce the performance of release builds and are otherwise unnecessary.
- Use `DEBUG` macro to print debug messages during normal functioning, and `RUNTIME_DEBUG` for debugging after `EXIT_BOOT_SERVICES`.
- Use `DEBUG_VERBOSE` debug level to leave debug messages for future debugging of the code, which are currently not necessary. By default `DEBUG_VERBOSE` messages are ignored even in `DEBUG` builds.
- Use `DEBUG_INFO` debug level for all non critical messages (including errors) and `DEBUG_BULK_INFO` for extensive messages that should not appear in NVRAM log that is heavily limited in size. These messages are ignored in `RELEASE` builds.
- Use `DEBUG_ERROR` to print critical human visible messages that may potentially halt the boot process, and `DEBUG_WARN` for all other human visible errors, `RELEASE` builds included.

When trying to find the problematic change it is useful to rely on `git-bisect` functionality. There also are some unofficial resources that provide per-commit binary builds of OpenCore, such as Dortania.

8. `Mask`
   Type: `plist data`
   Failsafe: Empty data
   Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Can be set to empty data to be ignored. Must equal to `Find` in size otherwise.

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

10. `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.

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

5.5 Quirks Properties

1. `AllowRelocationBlock`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Allows booting macOS through a relocation block.

   Relocation block is a scratch buffer allocated in lower 4 GB to be used for loading the kernel and related structures by EfiBoot on firmwares where lower memory is otherwise occupied by the (assumed to be) non-runtime data. Right before kernel startup the relocation block is copied back to lower addresses. Similarly all the other addresses pointing to relocation block are also carefully adjusted. Relocation block can be used when:

   - No better slide exists (all the memory is used)
   - `slide=0` is forced (by an argument or safe mode)
   - KASLR (slide) is unsupported (this is macOS 10.7 or older)

   This quirk requires `ProvideCustomSlide` to also be enabled and generally needs `AvoidRuntimeDefrag` to work correctly. Hibernation is not supported when booting with a relocation block (but relocation block is not always used when the quirk is enabled).

   Note: While this quirk is required to run older macOS versions on platforms with used lower memory it is not compatible with some hardware and macOS 11. In this case ~~you~~ one may try to use `EnableSafeModeSlide` instead.

2. `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 firmware that uses SMM backing for select services such as variable storage. SMM may try to access physical addresses, but they get moved by boot.efi.

   Note: Most types of firmware, apart from Apple and VMware, need this quirk.

3. `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

13. `LapicKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.6 (64-bit)
    Description: Disables kernel panic on LAPIC interrupts.

14. `LegacyCommpage`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.4 - 10.6
    Description: Replaces the default 64-bit commpage bcopy implementation with one that does not require SSSE3, useful for legacy platforms. This prevents a `commpage no match for last` panic due to no available 64-bit bcopy functions that do not require SSSE3.

15. `PanicNoKextDump`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.13 (not required for older)
    Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

16. `PowerTimeoutKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.15 (not required for older)
    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.

17. `SetApfsTrimTimeout`
    Type: `plist integer`
    Failsafe: `-1`
    Requirement: 10.14 (not required for older)
    Description: Set trim timeout in microseconds for APFS filesystems on SSDs.

    APFS filesystem is designed in a way that the space controlled via spaceman structure is either used or free. This may be different in other filesystems where the areas can be marked as used, free, and *unmapped*. All free space is trimmed (unmapped/deallocated) at macOS startup. The trimming procedure for NVMe drives happens in LBA ranges due to the nature of `DSM` command with up to 256 ranges per command. The more fragmented the memory on the drive is, the more commands are necessary to trim all the free space.

    Depending on the SSD controller and the drive fragmenation trim procedure may take considerable amount of time, causing noticeable boot slowdown APFS driver explicitly ignores previously unmapped areas and trims them on boot again and again. To workaround boot slowdown macOS driver introduced a timeout (`9.999999` seconds) that stops trim operation when it did not manage to complete in time. On many controllers, such as Samsung, where the deallocation is not very fast, the timeout is reached very quickly. Essentially it means that macOS will try to trim all the same lower blocks that have already been deallocated, but will never have enough time to deallocate higher blocks once the fragmentation increases. This means that trimming on these SSDs will be broken soon after the installation, causing extra wear to the flash.

    One way to workaround the problem is to increase the timeout to a very high value, which at the cost of slow boot times (extra minutes) will ensure that all the blocks are trimmed. For this one can set this option to a high value, e.g. `4294967295`.

    Another way is to utilise over-provisioning if it is supported or create a dedicated unmapped partition where the reserve blocks can be found by the controller. In this case the trim operation can also be disabled by setting a very low timeout. e.g. `999`. See more details in this article.

18. `ThirdPartyDrives`

- Mark the option as the default option to boot.
- Boot option through the picker or without it depending on the `ShowPicker` option.
- Show picker on failure otherwise.

Note 1: This process is meant to work reliably only when `RequestBootVarRouting` option is enabled or the firmware does not control UEFI boot options (`OpenDuetPkg` or custom BDS). Without ~~`BootProtect`~~`LauncherOption` it also is possible that other operating systems overwrite OpenCore, make sure to enable it when planning to use them.

Note 2: UEFI variable boot options' boot arguments will be removed if present as they may contain arguments compromising the operating system, which is undesired once secure boot is enabled.

Note 3: Some operating systems, namely Windows, will create their boot option and mark it as top most upon first boot or after NVRAM Reset. When this happens default boot entry choice will update till next manual reconfiguration.

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\debian\grubx64.efi` for Debian bootloader. This allows unusual boot paths to be automatically discovered by the boot picker. Designwise they are equivalent to predefined blessed path, such as `\System\Library\CoreServices\boot.efi` or `\EFI\Microsoft\Boot\bootmgfw.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. For tool examples check the UEFI section of this document.

8.3 Boot Properties

1. `ConsoleAttributes`
   Type: `plist integer`
   Failsafe: `0`
   Description: Sets specific attributes for console.

   Text renderer supports colour arguments as a sum of foreground and background colours 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.

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

   - `None` — Avoid hibernation (Recommended).
   - `Auto` — Use RTC and NVRAM detection.
   - `RTC` — Use RTC detection.
   - `NVRAM` — Use NVRAM detection.

3. `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. `Reset 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.

4. `LauncherOption`
   Type: `plist string`
   Failsafe: `Disabled`
   Description: Register launcher option in firmware preferences for persistence.

   Valid values:

   - `Disabled` — do nothing.
   - `Full` — create or update top-priority boot option in UEFI variable storage at bootloader startup. For this option to work `RequestBootVarRouting` is required to be enabled.

- `Short` — create a short boot option instead of a complete one. This variant is useful for some older firmwares, Insyde in particular, but possibly others, which cannot handle full device paths.

This option provides integration with third-party operating system installation and upgrade at the times they overwrite `\EFI\BOOT\BOOTx64.efi` file. By creating a custom option in this file path becomes no longer used for bootstrapping OpenCore. The path used for bootstrapping is specified in `LauncherPath` option.

Note 1: Some types of firmware may have faulty NVRAM, no boot option support, or other incompatibilities. While unlikely, the use of this option may even cause boot failures. This option should be used without any warranty exclusively on the boards known to be compatible. Check acidanthera/bugtracker#1222 for some known issues with Haswell and other boards.

Note 2: Be aware that while NVRAM reset executed from OpenCore should not erase the boot option created in `Bootstrap`, executing NVRAM reset prior to loading OpenCore will remove it. For significant implementation updates (e.g. in OpenCore 0.6.4) make sure to perform NVRAM reset with `Bootstrap` disabled before reenabling.

5. `LauncherPath`
   Type: `plist string`
   Failsafe: `Default`
   Description: Launch path for `LauncherOption`.

   `Default` stays for launched `OpenCore.efi`, any other path, e.g. `\EFI\Launcher.efi`, can be used to provide custom loaders, which are supposed to load `OpenCore.efi` themselves.

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

   Different pickers may be configured through the attribute mask containing OpenCore-reserved (`BIT0~BIT15`) and OEM-specific (`BIT16~BIT31`) values.

   Current OpenCore values include:

   - `0x0001` — `OC_ATTR_USE_VOLUME_ICON`, provides custom icons for boot entries:
     For `Tools` OpenCore will try to load a custom icon and fallback to the default icon:
     - `ResetNVRAM` — `Resources\Image\ResetNVRAM.icns` — `ResetNVRAM.icns` from icons directory.
     - `Tools\<TOOL_RELATIVE_PATH>.icns` — icon near the tool file with appended `.icns` extension.

     For custom boot `Entries` OpenCore will try to load a custom icon and fallback to the volume icon or the default icon:
     - `<ENTRY_PATH>.icns` — icon near the entry file with appended `.icns` extension.

     For all other entries OpenCore will try to load a volume icon and fallback to the default icon:
     - `.VolumeIcon.icns` file at `Preboot` volume directory for APFS (if present).
     - `.VolumeIcon.icns` file at `Preboot` root for APFS (otherwise).
     - `.VolumeIcon.icns` file at volume root for other filesystems.

     Volume icons can be set in Finder. Note, that enabling this may result in external and internal icons to be indistinguishable.
   - `0x0002` — `OC_ATTR_USE_DISK_LABEL_FILE`, provides custom rendered titles for boot entries:
     - `.disk_label` (`.disk_label_2x`) file near bootloader for all filesystems.
     - `<TOOL_NAME>.lbl` (`<TOOL_NAME>.l2x`) file near tool for `Tools`.

     Prerendered labels can be generated via `disklabel` utility or `bless` command. When disabled or missing text labels (`.contentDetails` or `.disk_label.contentDetails`) are to be rendered instead.
   - `0x0004` — `OC_ATTR_USE_GENERIC_LABEL_IMAGE`, provides predefined label images for boot entries without custom entries. May give less detail for the actual boot entry.
   - `0x0008` — `OC_ATTR_HIDE_THEMED_ICONS`, prefers builtin icons for certain icon categories to match the theme style. For example, this could force displaying the builtin Time Machine icon. Requires `OC_ATTR_USE_VOLUME_ICON`.
   - `0x0010` — `OC_ATTR_USE_POINTER_CONTROL`, enable pointer control in the picker when available. For example, this could make use of mouse or trackpad to control UI elements.

Development and debug kernels produce more helpful kernel panics. Consider downloading and installing `KernelDebugKit` from developer.apple.com when debugging a problem. To activate a development kernel the boot argument `kcsuffix=development` should be added. Use `uname -a` command to ensure that the current loaded kernel is a development (or a debug) kernel.

In case OpenCore kernel panic saving mechanism was not used, kernel panics may still be found in `/Library/Logs/DiagnosticReports` directory. Starting with macOS Catalina kernel panics are stored in JSON format, so they need to be preprocessed before passing to `kpdescribe.sh`:

```
cat Kernel.panic | grep macOSProcessedStackshotData |
  python -c 'import json,sys;print(json.load(sys.stdin)["macOSPanicString"])'
```

3. `DisableWatchDog`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Some types of firmware may not succeed in booting the operating system quickly, especially in debug mode, which results in the watchdog timer aborting the process. This option turns off the watchdog timer.

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

5. `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`.

6. `SerialInit`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform serial port initialisation.

   This option will perform serial port initialisation within OpenCore prior to enabling (any) debug logging. Serial port configuration is defined via PCDs at compile time in `gEfiMdeModulePkgTokenSpaceGuid` GUID. Default values as found in `MdeModulePkg.dec` are as follows:

   - `PcdSerialBaudRate` — Baud rate: `115200`.
   - `PcdSerialLineControl` — Line control: no parity, 8 data bits, 1 stop bit.

   See more details in `Debugging` section.

7. `SysReport`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Produce system report on ESP folder.

   This option will create a `SysReport` directory on ESP partition unless it is already present. The directory will contain ACPI~~and SMBIOSdumps.~~ , SMBIOS, and audio codec dumps. Audio codec dumps require an audio backend driver to be loaded.

   Note: For security reasons `SysReport` option is **not** available in `RELEASE` builds. Use a `DEBUG` build if this option is needed.

8. `Target`
   Type: `plist integer`
   Failsafe: `0`

Failsafe: `0`
Description: Apple Enclave Identifier.

Setting this value to any non-zero 64-bit integer will allow using personalised Apple Secure Boot identifiers. To use this setting, make sure to generate a random 64-bit number with a cryptographically secure random number generator. As an alternative, first 8 bytes of `SystemUUID` can be used for `ApECID`, this is found in macOS 11 for Macs without the T2 chip.

With this value set and `SecureBootModel` valid and not `Disabled` it is possible to achieve `Full Security` of Apple Secure Boot.

To start using personalised Apple Secure Boot, the operating system will have to be reinstalled or personalised. Unless the operating system is personalised, macOS DMG recovery cannot be loaded. If DMG recovery is missing, it can be downloaded with `macrecovery` utility and put to `com.apple.recovery.boot` as explained in Tips and Tricks section. Note that DMG loading needs to be set to `Signed` to use any DMG with Apple Secure Boot.

To personalise an existing operating system use `bless` command after loading to macOS DMG recovery. Mount the system volume partition, unless it has already been mounted, and execute the following command:

```
bless bless --folder "/Volumes/Macintosh HD/System/Library/CoreServices" \
  --bootefi --personalize
```

Before macOS 11, which introduced a dedicated `x86legacy` model for models without the T2 chip, personalised Apple Secure Boot may not work as expected. When reinstalling the operating system, macOS Installer from macOS 10.15 and older, will usually run out of free memory on the `/var/tmp` partition when trying to install macOS with the personalised Apple Secure Boot. Soon after downloading the macOS installer image an `Unable to verify macOS` error message will appear. To workaround this issue allocate a dedicated RAM disk of 2 MBs for macOS personalisation by entering the following commands in macOS recovery terminal before starting the installation:

```
disk=$(hdiutil attach -nomount ram://4096)
diskutil erasevolume HFS+ SecureBoot $disk
diskutil unmount $disk
mkdir /var/tmp/OSPersonalizationTemp
diskutil mount -mountpoint /var/tmp/OSPersonalizationTemp $disk
```

4. `AuthRestart`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable `VirtualSMC`-compatible authenticated restart.

   Authenticated restart is a way to reboot FileVault 2 enabled macOS without entering the password. A dedicated terminal command can be used to perform authenticated restarts: `sudo fdesetup authrestart`. It is also used when installing operating system updates.

   VirtualSMC performs authenticated restart by saving disk encryption key split in NVRAM and RTC, which despite being removed as soon as OpenCore starts, may be considered a security risk and thus is optional.

5. `BlacklistAppleUpdate`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Ignore boot options trying to update Apple peripheral firmware (e.g. `MultiUpdater.efi`).

   Note: This option exists due to some operating systems, namely macOS Big Sur, being incapable of disabling firmware updates with the NVRAM variable (`run-efi-updater`).

6. ~~`BootProtect`~~**~~Type~~**~~: `plist string`~~**~~Failsafe~~**~~: `None`~~**~~Description~~**~~: Attempt to provide bootloader persistence.~~

   ~~Valid values:~~

   - ~~`None` — do nothing.~~

- ~~`Bootstrap` — create or update top-priority `\EFI\OC\Bootstrap\Bootstrap.efi` boot option in UEFI variable storage at bootloader startup. For this option to work `RequestBootVarRouting` is required to be enabled.~~
- ~~`BootstrapShort` — create a short boot option instead of a complete one, otherwise equivalent to `Bootstrap`. This variant is useful for some older firmwares, Insyde in particular, but possibly others, which cannot handle full device paths.~~

~~This option provides integration with third-party operating system installation and upgrade at the times they overwrite `\EFI\BOOT\BOOTx64.efi` file. By creating a custom option in `Bootstrap` mode this file path becomes no longer used for bootstrapping OpenCore.~~

~~*Note 1*: Some types of firmware may have faulty NVRAM, no boot option support, or other incompatibilities. While unlikely, the use of this option may even cause boot failures. This option should be used without any warranty exclusively on the boards known to be compatible. Check acidanthera/bugtracker#1222 for some known issues with Haswell and other boards.~~

~~*Note 2*: Be aware that while NVRAM reset executed from OpenCore should not erase the boot option created in `Bootstrap`, executing NVRAM reset prior to loading OpenCore will remove it. For significant implementation updates (e.g. in OpenCore 0.6.4) make sure to perform NVRAM reset with `Bootstrap` disabled before reenabling.~~

7. `DmgLoading`
   Type: `plist string`
   Failsafe: `Signed`
   Description: Define Disk Image (DMG) loading policy used for macOS Recovery.

   Valid values:

   - `Disabled` — loading DMG images will fail. `Disabled` policy will still let macOS Recovery to load in most cases as there usually are `boot.efi` files compatible with Apple Secure Boot. Manually downloaded DMG images stored in `com.apple.recovery.boot` directories will not load, however.
   - `Signed` — only Apple-signed DMG images will load. Due to Apple Secure Boot design `Signed` policy will let any Apple-signed macOS Recovery to load regardless of Apple Secure Boot state, which may not always be desired.
   - `Any` — any DMG images will mount as normal filesystems. `Any` policy is strongly not recommended and will cause a boot failure when Apple Secure Boot is activated.

8. `EnablePassword`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable password protection to allow sensitive operations.

   Password protection ensures that sensitive operations such as booting a non-default operating system (e.g. macOS recovery or a tool), resetting NVRAM storage, trying to boot into a non-default mode (e.g. verbose mode or safe mode) are not allowed without explicit user authentication by a custom password. Currently password and salt are hashed with 5000000 iterations of SHA-512.

   Note: This functionality is currently in development and is not ready for daily usage.

9. `ExposeSensitiveData`
   Type: `plist integer`
   Failsafe: `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.
   - `0x08` — Expose OEM information as a set of UEFI variables.

   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
```

Warning: This feature is very dangerous as it passes unprotected data to 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.

   `*` value can be used 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: It is recommended to have this value enabled on most types of firmware but it is left configurable for firmware that may have issues with NVRAM variable storage garbage collection or similar.

To read NVRAM variable value from macOS, `nvram` could be used by concatenating GUID and name variables separated by a `:` 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~~ recommended 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`.
- `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`.

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.

6. `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 on some types of firmware.
   - `Create` — Replace the tables with newly allocated EfiReservedMemoryType at AllocateMaxAddress without any fallbacks.
   - `Overwrite` — Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
   - `Custom` — Write SMBIOS tables (`gEfiSmbios(3)TableGuid`) to `gOcCustomSmbios(3)TableGuid` to workaround firmware overwriting SMBIOS contents at ExitBootServices. Otherwise equivalent to `Create`. Requires patching AppleSmbios.kext and AppleACPIPlatform.kext to read from another GUID: `"EB9D2D31"` - `"EB9D2D35"` (in ASCII), done automatically by `CustomSMBIOSGuid` quirk.

   Note: A side effect of using `Custom` approach is making SMBIOS updates exclusive to macOS, avoiding a collision with existing Windows activation and custom OEM software but potentially breaking Apple-specific tools.

7. `Generic`
   Type: `plist dictionary`
   Description: Update all fields. This section is read only when `Automatic` is active.

8. `DataHub`
   Type: `plist dictionary`
   ~~Optional~~Ignored: When `Automatic` is `true`
   Description: Update Data Hub fields. This section is read only when `Automatic` is not active.

9. `Memory`
   Type: `plist dictionary`
   ~~Optional~~Ignored: When `CustomMemory` is `false`
   Description: Define custom memory configuration.

10. `PlatformNVRAM`
    Type: `plist dictionary`
    ~~Optional~~Ignored: When `Automatic` is `true`
    Description: Update platform NVRAM fields. This section is read only when `Automatic` is not active.

11. `SMBIOS`
    Type: `plist dictionary`
    ~~Optional~~Ignored: When `Automatic` is `true`
    Description: Update SMBIOS fields. This section is read only when `Automatic` is not active.

10.2 Generic Properties

1. `SpoofVendor`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Sets SMBIOS vendor fields to `Acidanthera`.

   It is dangerous to use Apple in SMBIOS vendor fields for reasons given in `SystemManufacturer` description. However, certain firmware may not provide valid values otherwise, which could break some software.

2. `AdviseWindows`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Forces Windows support in `FirmwareFeatures`.

   Added bits to `FirmwareFeatures`:

   - `FW_FEATURE_SUPPORTS_CSM_LEGACY_MODE` (`0x1`) - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being not the first partition on the disk.

- `FW_FEATURE_SUPPORTS_UEFI_WINDOWS_BOOT` (`0x20000000`) - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being the first partition on the disk.

3. `MaxBIOSVersion`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Sets `BIOSVersion` to `9999.999.999.999.999`, recommended for legacy Macs when using `Automatic` PlatformInfo to avoid BIOS updates in unofficially supported macOS versions.

4. `SystemMemoryStatus`
   Type: `plist string`
   Failsafe: `Auto`
   Description: Indicates whether system memory is upgradable in `PlatformFeature`. This controls the visibility of the Memory tab in About This Mac.

   Valid values:

   - `Auto` — use the original `PlatformFeature` value.
   - `Upgradable` — explicitly unset `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` (`0x2`) in `PlatformFeature`.
   - `Soldered` — explicitly set `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` (`0x2`) in `PlatformFeature`.

   Note: On certain Mac models (namely `MacBookPro10,x` and any `MacBookAir`), SPMemoryReporter.spreporter will ignore `PT_FEATURE_HAS_SOLDERED_SYSTEM_MEMORY` and assume that system memory is non-upgradable.

5. `ProcessorType`
   Type: `plist integer`
   Failsafe: `0` (Automatic)
   Description: Refer to SMBIOS `ProcessorType`.

6. `SystemProductName`
   Type: `plist string`
   Failsafe: ~~`MacPro6,1`~~OEM specified or not installed
   Description: Refer to SMBIOS `SystemProductName`.

7. `SystemSerialNumber`
   Type: `plist string`
   Failsafe: ~~`OPENCORE_SN1`~~OEM specified or not installed
   Description: Refer to SMBIOS `SystemSerialNumber`.

8. `SystemUUID`
   Type: `plist string`, GUID
   Failsafe: OEM specified or not installed
   Description: Refer to SMBIOS `SystemUUID`.

9. `MLB`
   Type: `plist string`
   Failsafe: ~~`OPENCORE_MLB_SN11`~~OEM specified or not installed
   Description: Refer to SMBIOS `BoardSerialNumber`.

10. `ROM`
    Type: `plist data`, 6 bytes
    Failsafe: ~~all zero~~OEM specified or not installed
    Description: Refer to `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14:ROM`.

10.3 DataHub Properties

1. `PlatformName`
   Type: `plist string`
   Failsafe: Not installed
   Description: Sets `name` in `gEfiMiscSubClassGuid`. Value found on Macs is `platform` in ASCII.

2. `SystemProductName`
   Type: `plist string`
   Failsafe: Not installed

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 Drivers

Depending on the firmware a different set of drivers may be required. Loading an incompatible driver may lead the system to unbootable state or even cause permanent firmware damage. Some of the known drivers are listed below:

```
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 and Applications

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, see more details in the Tools subsection of the configuration, most 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 partition 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: `/System/Library/CoreServices/BridgeVersion.bin` should be copied to `/Volumes/VOLNAME/DIR`.
Note 2: To be able to use `bless` disabling System Integrity Protection is necessary.
Note 3: To be able to boot Secure Boot might be disabled 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`).~~	Test keyboard input in `SimpleText` mode.
`KeyTester*`	
`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 firmware.
`PavpProvision`	Perform EPID provisioning (requires certificate data configuration).
`ResetSystem*`	Utility to perform system reset. Takes reset type as an argument: `ColdReset`, `Firmware`, `Shutdown`, `WarmReset`. Defaults to `ColdReset`.
`RtcRw*`	Utility to read and write RTC (CMOS) memory.
`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. Customised icons can be found over the internet (e.g. here or there).

OpenCanopy provides full support for `PickerAttributes` and offers a configurable builtin icon set. The default chosen icon set depends on the `DefaultBackgroundColor` variable value. For Light Gray `Old` icon set will be used, for other colours — the one without a prefix.

Predefined icons are put to `\EFI\OC\Resources\Image` directory. Full list of supported icons (in `.icns` format) is provided below. Missing optional icons will use the closest available icon. External entries will use `Ext`-prefixed icon if available (e.g. `OldExtHardDrive.icns`).

Note: In the following all dimensions are normative for the 1x scaling level and shall be scaled accordingly for other levels.

- `Cursor` — Mouse cursor (mandatory, up to 144x144).
- `Selected` — Selected item (mandatory, 144x144).
- `Selector` — Selecting item (mandatory, up to 144x40).
- `Left` — Scrolling left (mandatory, 40x40).
- `Right` — Scrolling right (mandatory, 40x40).
- `HardDrive` — Generic OS (mandatory, 128x128).
- `Background` — Centred background image.
- `Apple` — Apple OS (128x128).
- `AppleRecv` — Apple Recovery OS (128x128).
- `AppleTM` — Apple Time Machine (128x128).
- `Windows` — Windows (128x128).
- `Other` — Custom entry (see `Entries`, 128x128).
- `ResetNVRAM` — Reset NVRAM system action or tool (128x128).
- `Shell` — Entry with UEFI Shell name ~~(~~for e.g. `OpenShell` (128x128).
- `Tool` — Any other tool (128x128).

Predefined labels are put to `\EFI\OC\Resources\Label` directory. Each label has `.lbl` or `.l2x` suffix to represent the scaling level. Full list of labels is provided below. All labels are mandatory.

- `EFIBoot` — Generic OS.
- `Apple` — Apple OS.
- `AppleRecv` — Apple Recovery OS.
- `AppleTM` — Apple Time Machine.
- `Windows` — Windows.
- `Other` — Custom entry (see `Entries`).
- `ResetNVRAM` — Reset NVRAM system action or tool.
- `Shell` — Entry with UEFI Shell name (e.g. `OpenShell`).
- `Tool` — Any other tool.

Note: All labels must have a height of exactly 12 px. There is no limit for their width.

Label and icon generation can be performed with bundled utilities: `disklabel` and `icnspack`. ~~Please refer to sample data for the details about the dimensions.~~ Font is Helvetica 12 pt times scale factor.

Font format corresponds to AngelCode binary BMF. While there are many utilities to generate font files, currently it is recommended to use dpFontBaker to generate bitmap font (using CoreText produces best results) and fonverter to export it to binary format.

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`).
- Read-only and write-only NVRAM variables, enhancing the security of OpenCore, Lilu, and Lilu plugins, such as 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. `APFS`
   Type: `plist dict`

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 string`
   Failsafe: ~~empty string~~`Auto`
   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. Possible values include:

   - `Auto` — Enables chime when `StartupMute` NVRAM variable is not present or set to `00`.
   - `Enabled` — Enables chime unconditionally.
   - `Disabled` — Disables chime unconditionally.

   Note: `Enabled` can be used in separate from `StartupMute` NVRAM variable to avoid conflicts when the firmware is able to play boot chime.

7. `SetupDelay`
   Type: `plist integer`
   Failsafe: `0`
   Description: Audio codec reconfiguration delay in microseconds.

   Some codecs require a vendor-specific delay after the reconfiguration (e.g. volume setting). This option makes it configurable. In general the necessary delay may be as long as 0.5 seconds.

8. `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.

11.9 Input Properties

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

   Apparently some boards such as the 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 the 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.

Note: Some platforms may require different values, higher or lower. For example, when detecting key misses in OpenCanopy try increasing this value (e.g. to 10), and when detecting key stall, try decreasing this value. Since every platform is different it may be reasonable to check every value from 1 to 25.

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 OpenUsbKbDxe, this option should never be enabled.

5. KeySupportMode
Type: plist string
Failsafe: ~~empty string~~Auto
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`. The value of this property cannot be empty if `PointerSupport` is enabled.

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

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

   The recommended value is `50000` (`5` milliseconds) or slightly higher. Select ASUS Z87 boards use `60000` for the interface. Apple boards use `100000`. In case of issues, this option can be left as `0`.

11.10 Output Properties

1. `TextRenderer`
   Type: `plist string`
   Failsafe: `BuiltinGraphics`
   Description: Chooses renderer for text going through standard console output.

   Currently two renderers are supported: `Builtin` and `System`. `System` renderer uses firmware services for text rendering. `Builtin` bypassing firmware services and performs text rendering on its own. Different renderers support a different set of options. It is recommended to use `Builtin` renderer, as it supports HiDPI mode and uses full screen resolution.

   UEFI firmware generally supports `ConsoleControl` with two rendering modes: `Graphics` and `Text`. Some types of firmware do not support `ConsoleControl` and rendering modes. OpenCore and macOS expect text to only be shown in `Graphics` mode and graphics to be drawn in any mode. Since this is not required by UEFI specification, exact behaviour varies.

   Valid values are combinations of text renderer and rendering mode:

   - `BuiltinGraphics` — Switch to `Graphics` mode and use `Builtin` renderer with custom `ConsoleControl`.
   - `BuiltinText` — Switch to `Text` mode and use `Builtin` renderer with custom `ConsoleControl`.
   - `SystemGraphics` — Switch to `Graphics` mode and use `System` renderer with custom `ConsoleControl`.
   - `SystemText` — Switch to `Text` mode and use `System` renderer with custom `ConsoleControl`.
   - `SystemGeneric` — Use `System` renderer with system `ConsoleControl` assuming it behaves correctly.

   The use of `BuiltinGraphics` is generally straightforward. For most platforms it is necessary to enable `ProvideConsoleGop`, set `Resolution` to `Max`. `BuiltinText` variant is an alternative `BuiltinGraphics` for some very old and buggy laptop firmware, which can only draw in `Text` mode.

   The use of `System` protocols is more complicated. In general the preferred setting is `SystemGraphics` or `SystemText`. Enabling `ProvideConsoleGop`, setting `Resolution` to `Max`, enabling `ReplaceTabWithSpace` is useful on almost all platforms. `SanitiseClearScreen`, `IgnoreTextInGraphics`, and `ClearScreenOnModeSwitch` are more specific, and their use depends on the firmware.

   Note: Some Macs, namely `MacPro5,1`, may have broken console output with newer GPUs, and thus only `BuiltinGraphics` may work for them.

2. `ConsoleMode`
   Type: `plist string`
   Failsafe: Empty string
   Description: Sets console output mode as specified with the `WxH` (e.g. `80x24`) formatted string.

This protocol replaces legacy `VirtualSmc` UEFI driver, and is compatible with any SMC kernel extension. However, in case `FakeSMC` kernel extension is used, manual NVRAM key variable addition may be needed.

12. `AppleUserInterfaceTheme`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Reinstalls Apple User Interface Theme protocol with a builtin version.

13. `DataHub`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Reinstalls Data Hub protocol with a builtin version. This will delete all previous properties if the protocol was already installed.

14. `DeviceProperties`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Reinstalls Device Property protocol with a builtin version. This will delete all previous properties if it was already installed. This may be used to ensure full compatibility on VMs or legacy Macs.

15. `FirmwareVolume`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Forcibly wraps Firmware Volume protocols or installs new to support custom cursor images for File Vault 2. Should be set to `true` to ensure File Vault 2 compatibility on everything but VMs and legacy Macs.

    Note: Several virtual machines including VMware may have corrupted cursor image in HiDPI mode and thus may also require this setting to be enabled.

16. `HashServices`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Forcibly reinstalls Hash Services protocols with builtin versions. Should be set to `true` to ensure File Vault 2 compatibility on platforms providing broken SHA-1 hashing. Can be diagnosed by invalid cursor size with `UIScale` set to `02`, in general platforms prior to APTIO V (Haswell and older) are affected.

17. `OSInfo`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Forcibly reinstalls OS Info protocol with builtin versions. This protocol is generally used to receive notifications from macOS bootloader, by the firmware or by other applications.

18. `UnicodeCollation`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Forcibly reinstalls unicode collation services with builtin version. Should be set to `true` to ensure UEFI Shell compatibility on platforms providing broken unicode collation. In general legacy Insyde and APTIO platforms on Ivy Bridge and earlier are affected.

11.12 Quirks Properties

1. `DisableSecurityPolicy`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disable platform security policy.

   Note: This setting disables various security features of the firmware, defeating the purpose of any kind of Secure Boot. Do NOT enable if you use UEFI Secure Boot.

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

12 Troubleshooting

12.1 Legacy Apple OS

Older operating systems may be more complicated to install, but sometimes can be necessary to use for all kinds of reasons. While a compatible board identifier and CPUID are the obvious requirements for proper functioning of an older operating system, there are many other less obvious things to consider. This section tries to cover a common set of issues relevant to installing older macOS operating systems.

While newer operating systems can be downloaded over the internet, older operating systems did not have installation media for every minor release, so to get a compatible distribution one may have to download a device-specific image and mod it if necessary. To get the list of the bundled device-specific builds for legacy operating systems one can visit this archived Apple Support article. Since it is not always accurate, the latest versions are listed below.

12.1.1 macOS 10.8 and 10.9

- Disk images on these systems use Apple Partitioning Scheme and ~~will require the proprietary~~ require ~~PartitionDxe~~OpenPartit driver to run DMG recovery and installation (included in OpenDuet). It is possible to set `DmgLoading` to `Disabled` to run the recovery without DMG loading avoiding the need for ~~PartitionDxe~~OpenPartitionDxe.
- Cached kernel images often do not contain family drivers for networking (`IONetworkingFamily`) or audio (`IOAudioFamily`) requiring the use of `Force` loading in order to inject networking or audio drivers.

12.1.2 macOS 10.7

- All previous issues apply.
- `SSSE3` support (not to be confused with `SSE3` support) is a hard requirement for macOS 10.7 kernel.
- Many kexts, including `Lilu` when 32-bit kernel is used and a lot of `Lilu` plugins, are unsupported on macOS 10.7 and older as they require newer kernel APIs, which are not part of the macOS 10.7 SDK.
- Prior to macOS 10.8 KASLR sliding is not supported, which will result in memory allocation failures on firmware that utilise lower memory for their own purposes. Refer to acidanthera/bugtracker#1125 for tracking.

12.1.3 macOS 10.6

- All previous issues apply.
- `SSSE3` support is a requirement for macOS 10.6 kernel with 64-bit userspace enabled. This limitation can mostly be lifted by enabling the `LegacyCommpage` quirk.
- Last released installer images for macOS 10.6 are macOS 10.6.7 builds `10J3250` (for `MacBookPro8,x`) and `10J4139` (for `iMac12,x`), without Xcode). These images are limited to their target model identifiers and have no `-no_compat_check` boot argument support. Modified images (with `ACDT` suffix) without model restrictions can be found here (MEGA Mirror), assuming macOS 10.6 is legally owned. Read `DIGEST.txt` for more details. Note that these are the earliest tested versions of macOS 10.6 with OpenCore.

Model checking may also be erased by editing `OSInstall.mpkg` with e.g. `Flat Package Editor` by making `Distribution` script to always return `true` in `hwbeModelCheck` function. Since updating the only file in the image and not corrupting other files can be difficult and may cause slow booting due to kernel cache date changes, it is recommended to script image rebuilding as shown below:

```
#!/bin/bash
# Original.dmg is original image, OSInstall.mpkg is patched package
mkdir RO
hdiutil mount Original.dmg -noverify -noautoopen -noautoopenrw -noautofsck -mountpoint RO
cp RO/.DS_Store DS_STORE
hdiutil detach RO -force
rm -rf RO
hdiutil convert Original.dmg -format UDRW -o ReadWrite.dmg
mkdir RW
xattr -c OSInstall.mpkg
```

```
hdiutil mount ReadWrite.dmg -noverify -noautoopen -noautoopenrw -noautofsck -mountpoint RW
cp OSInstall.mpkg RW/System/Installation/Packages/OSInstall.mpkg
killall Finder fseventsd
rm -rf RW/.fseventsd
cp DS_STORE RW/.DS_Store
hdiutil detach RW -force
rm -rf DS_STORE RW
hdiutil convert ReadWrite.dmg -format UDZO -o ReadOnly.dmg
```

12.1.4 macOS 10.5

- All previous issues apply.
- This macOS version does not support `x86_64` kernel and requires `i386` kernel extensions and patches.
- This macOS version uses the first (V1) version of `prelinkedkernel`, which has kext symbol tables corrupted by the kext tools. This nuance renders `prelinkedkernel` kext injection impossible in OpenCore. `Mkext` kext injection will still work without noticeable performance drain and will be chosen automatically when `KernelCache` is set to `Auto`.
- Last released installer image for macOS 10.5 is macOS 10.5.7 build `9J3050` (for `MacBookPro5,3`). Unlike the others, this image is not limited to the target model identifiers and can be used as is. The original `9J3050` image can be found here (MEGA Mirror), assuming macOS 10.5 is legally owned. Read `DIGEST.txt` for more details. Note that this is the earliest tested version of macOS 10.5 with OpenCore.

12.1.5 macOS 10.4

- All previous issues apply.
- This macOS version has a hard requirement to access all the optional packages on the second DVD disk installation media, requiring either two disks or USB media installation.
- Last released installer images for macOS 10.4 are macOS 10.4.10 builds `8R4061a` (for `MacBookPro3,1`) and `8R4088` (for `iMac7,1`)). These images are limited to their target model identifiers as on newer macOS versions. Modified `8R4088` images (with `ACDT` suffix) without model restrictions can be found here (MEGA Mirror), assuming macOS 10.4 is legally owned. Read `DIGEST.txt` for more details. Note that these are the earliest tested versions of macOS 10.4 with OpenCore.

12.2 UEFI Secure Boot

OpenCore is designed to provide a secure boot chain between firmware and operating system. On most x86 platforms trusted loading is implemented via UEFI Secure Boot model. Not only OpenCore fully supports this model, but it also extends its capabilities to ensure sealed configuration via vaulting and provide trusted loading to the operating systems using custom verification, such as Apple Secure Boot. Proper secure boot chain requires several steps and careful configuration of select settings as explained below:

1. Enable Apple Secure Boot by setting `SecureBootModel` to run macOS. Note, that not every macOS is compatible with Apple Secure Boot and there are several other restrictions as explained in Apple Secure Boot section.
2. Disable DMG loading by setting `DmgLoading` to `Disabled` if users have concerns of loading old vulnerable DMG recoveries. This is **not** required, but recommended. For the actual tradeoffs see the details in DMG loading section.
3. Make sure that APFS JumpStart functionality restricts the loading of old vulnerable drivers by setting `MinDate` and `MinVersion` to `0`. More details are provided in APFS JumpStart section. An alternative is to install `apfs.efi` driver manually.
4. Make sure that `Force` driver loading is not needed and all the operating systems are still bootable.
5. Make sure that `ScanPolicy` restricts loading from undesired devices. It is a good idea to prohibit all removable drivers or unknown filesystems.

6. Sign all the installed drivers and tools with the private key. Do not sign tools that provide administrative access to the computer, such as UEFI Shell.
7. Vault the configuration as explained Vaulting section.
8. Sign all OpenCore binaries (`BOOTX64.efi`, `BOOTIa32.efi`, ~~`Bootstrap.efi`,~~ `OpenCore.efi`, custom launchers) used on this system with the same private key.
9. Sign all third-party operating system (not made by Microsoft or Apple) bootloaders if needed. For Linux there is an option to install Microsoft-signed Shim bootloader as explained on e.g. Debian Wiki.
10. Enable UEFI Secure Boot in firmware preferences and install the certificate with a private key. Details on how to generate a certificate can be found in various articles, such as this one, and are out of the scope of this document. If Windows is needed one will also need to add the Microsoft Windows Production CA 2011. To launch option ROMs or to use signed Linux drivers, Microsoft UEFI Driver Signing CA will also be needed.
11. Password-protect changing firmware settings to ensure that UEFI Secure Boot cannot be disabled without the user's knowledge.

12.3 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, such as Windows 7, might work with some extra precautions. Things to consider:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- 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 setting SystemUUID to the original firmware UUID. Be aware that it may be invalid on old firmware, i.e., not random. If there still are issues, consider using HWID or KMS38 license or making the use `Custom` `UpdateSMBIOSMode`. Other 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 Windows support software from Boot Camp is required. 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 7-Zip may be downloaded and installed 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`. If there is a previous version of Boot Camp installed it should be removed 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, the rest may still have to be addressed 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 is usually not needed).
- To access Apple filesystems such as HFS+ and APFS, separate software may need to be installed. Some of the known utilities 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.