OpenCore

Reference Manual (0.6.5.6)

[2021.01.24]

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.

    Depending on the SSD controller trim procedure may take considerable amount of time, causing noticeable boot slowdown as this is when the APFS driver executes the trim operation. If the SSD supports over-provisioning or there exists a dedicated unused partition that can be used to reserve blocks, trim operation may not be necessary. On the other side if the SSD is slow, the default timeout (equals to `9.999999` seconds) may not be enough. See more details in this article.

    Set this value to `4294967295` for the maximum timeout to guarantee trim success or to `999` to essentially disable trim.

18. `ThirdPartyDrives`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.6 (not required for older)
    Description: Apply vendor patches to IOAHCIBlockStorage.kext to enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

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

19. `XhciPortLimit`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.11 (not required for older)

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`
   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:

   - `0x01` (bit `0`) — Enable logging, otherwise all log is discarded.
   - `0x02` (bit `1`) — Enable basic console (onscreen) logging.
   - `0x04` (bit `2`) — Enable logging to Data Hub.
   - `0x08` (bit `3`) — Enable serial port logging.
   - `0x10` (bit `4`) — Enable UEFI variable logging.
   - `0x20` (bit `5`) — Enable non-volatile UEFI variable logging.
   - `0x40` (bit `6`) — Enable logging to file.

   Console logging prints less than all the other variants. Depending on the build type (`RELEASE`, `DEBUG`, or `NOOPT`) different amount of logging may be read (from least to most).

   Data Hub log will not log kernel and kext patches. To obtain Data Hub log use the following command in macOS:

```
ioreg -lw0 -p IODeviceTree | grep boot-log | sort | sed 's/.*<\(.*\)>.*/\1/' | xxd -r -p
```

   UEFI variable log does not include some messages and has no performance data. For safety reasons log size is limited to 32 kilobytes. Some types of firmware may truncate it much earlier or drop completely if they have no memory. Using non-volatile flag will write the log to NVRAM flash after every printed line. To obtain UEFI variable log use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-log |
  awk '{gsub(/%0d%0a%00/,"");gsub(/%0d%0a/,"\n")}1'
```

   Warning: Some types of firmware appear to have flawed NVRAM garbage collection. This means that they may not be able to always free space after variable deletion. Do not use non-volatile NVRAM logging without extra need on such devices.

   While OpenCore boot log already contains basic version information with build type and date, this data may also be found in NVRAM in `opencore-version` variable even with boot log disabled.

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.

- `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`
   Description: Refer to SMBIOS `SystemProductName`.

7. `SystemSerialNumber`
   Type: `plist string`
   Failsafe: `OPENCORE_SN1`
   Description: Refer to SMBIOS `SystemSerialNumber`.

8. `SystemUUID`
   Type: `plist string`, GUID
   Failsafe: OEM specified
   Description: Refer to SMBIOS `SystemUUID`.

9. `MLB`
   Type: `plist string`
   Failsafe: `OPENCORE_MLB_SN11`
   Description: Refer to SMBIOS `BoardSerialNumber`.

10. `ROM`
    Type: `plist data`, 6 bytes
    Failsafe: all zero
    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:

```
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.

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 `PartitionDxe` driver to run DMG recovery and installation. It is possible to set `DmgLoading` to `Disabled` to run the recovery without DMG loading avoiding the need for `PartitionDxe`.
- 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.