OpenCore

Reference Manual (0.5.~~8~~.9)

[2020.05.05]

- `Resources`
  Directory used for storing media resources, such as audio files for screen reader support. See `UEFI Audio Properties` section for more details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  Main booter driver responsible for operating system loading.
- `vault.plist`
  Hashes for all files potentially loadable by `OC Config`.
- `config.plist`
  `OC Config`.
- `vault.sig`
  Signature for `vault.plist`.
- `nvram.plist`
  OpenCore variable import file.
- `opencore-YYYY-MM-DD-HHMMSS.txt`
  OpenCore log file.

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

3.2 Installation and Upgrade

To install OpenCore reflect the Configuration Structure described in the previous section on a EFI volume of a GPT partition. While corresponding sections of this document do provide some information in regards to external resources like ACPI tables, UEFI drivers, or kernel extensions (kexts), completeness of the matter is out of the scope of this document. Information about kernel extensions may be found in a separate Kext List document available in OpenCore repository. Vaulting information is provided in Security Properties section of this document.

`OC config`, just like any property lists can be edited with any stock textual editor (e.g. nano, vim), but specialised software may provide better experience. On macOS the preferred GUI application is Xcode. For a lightweight cross-platform and open-source alternative ProperTree editor can be utilised.

For BIOS booting a third-party UEFI environment provider will have to be used. `DuetPkg` is one of the known UEFI environment providers for legacy systems. To run OpenCore on such a legacy system you can install `DuetPkg` with a dedicated tool BootInstall (bundled with OpenCore).

For upgrade purposes refer to `Differences.pdf` document, providing the information about the changes affecting the configuration compared to the previous release, and `Changelog.md` document, containing the list of modifications across all published updates.

3.3 Contribution

OpenCore can be compiled as an ordinary EDK II. Since UDK development was abandoned by TianoCore, OpenCore requires the use of EDK II Stable. Currently supported EDK II release (potentially with patches enhancing the experience) is hosted in acidanthera/audk.

The only officially supported toolchain is `XCODE5`. Other toolchains might work, but are neither supported, nor recommended. Contribution of clean patches is welcome. Please do follow EDK II C Codestyle.

Required external package dependencies include EfiPkg and MacInfoPkg.

To compile with `XCODE5`, besides Xcode, one should also install NASM and MTOC. The latest Xcode version is recommended for use despite the toolchain name. Example command sequence may look as follows:

```
git clone https://github.com/acidanthera/audk UDK
cd UDK
git clone https://github.com/acidanthera/DuetPkg
git clone https://github.com/acidanthera/EfiPkg
git clone https://github.com/acidanthera/MacInfoPkg
git clone https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
```

```
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

For IDE usage Xcode projects are available in the root of the repositories. Another approach could be Sublime Text with EasyClangComplete plugin. Add `.clang_complete` file with similar content to your UDK root:

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/MdeModulePkg
-I/UefiPackages/MdeModulePkg/Include
-I/UefiPackages/MdeModulePkg/Include/X64
-I/UefiPackages/DuetPkg/Include
-I/UefiPackages/EfiPkg
-I/UefiPackages/EfiPkg/Include
-I/UefiPackages/EfiPkg/Include/X64
-I/UefiPackages/AppleSupportPkg/Include
-I/UefiPackages/OpenCorePkg/Include
-I/UefiPackages/OvmfPkg/Include
-I/UefiPackages/MacInfoPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
-DNO_MSABI_VA_FUNCS=1
```

Listing 2: ECC Configuration

Warning: Tool developers modifying `config.plist` or any other OpenCore files must ensure that their tool checks for `opencore-version` NVRAM variable (see Debug Properties section below) and warn the user if the version listed is unsupported or prerelease. OpenCore configuration may change across the releases and the tool shall ensure that it carefully follows this document. Failure to do so may result in this tool to be considered as malware and blocked with all possible means.

3.4 Coding conventions

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

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

- Whenever changes are required in multiple repositories, separate pull requests should be sent to each.
- Committing the changes should happen firstly to dependent repositories, secondly to primary repositories to avoid automatic build errors.

8 Misc

8.1 Introduction

This section contains miscellaneous configuration ~~entries for OpenCore behaviour that does not go to any other sections~~ affecting OpenCore operating system loading behaviour as well as other entries, which do not go to any other section.

OpenCore tries to follow "bless" model also known as "Apple Boot Policy". The primary specialty of "bless" model is to allow embedding boot options within the file system (and be accessible through a specialised driver) as well as supporting a broader range of predefined boot paths compared to the removable media list found in the UEFI specification.

Each partition will only be used for booting when it corresponds to "Scan policy": a set of restrictions to only use partitions with specific file systems and from specific device types. Scan policy behaviour is discussed in `ScanPolicy` property description.

Scan process starts with obtaining all the partitions filtered with "Scan policy". Each partition may produce multiple primary and alternate options. Primary options describe operating systems installed on this media. Alternate options describe recovery options for the operating systems on the media. It is possible for alternate options to exist without primary options and vice versa. Be warned that the options may not necessarily describe the operating systems on the same partition. Each primary and alternate option can be an auxiliary option or not, refer to `HideAuxiliary` for more details. Algorithm to determine boot options behaves as follows:

1. Obtain all available partition handles filtered by "Scan policy" (and driver availability).
2. Obtain all available boot options from `BootOrder` UEFI variable.
3. For each found boot option:
   - Retrieve device path of the boot option.
   - Perform fixups (e.g. NVMe subtype correction) and expansion (e.g. for Boot Camp) of the device path.
   - Obtain device handle by locating device path of the resulting device path (ignore it on failure).
   - Find device handle in the list of partition handles (ignore it if missing).
   - For disk device paths (not specifying a bootloader) execute "bless" (may return > 1 entry).
   - For file device paths check presence on the file system directly.
   - Exclude options with blacklisted filenames (refer to `BlacklistAppleUpdate` option).
   - On OpenCore boot partition exclude all OpenCore bootstrap files by header checks.
   - Mark device handle as *used* in the list of partition handles if any.
   - Register the resulting entries as primary options and determine their types.
     The option will become auxiliary for some types (e.g. Apple HFS recovery).
4. For each partition handle:
   - If partition handle is marked as *unused* execute "bless" primary option list retrieval.
     In case `BlessOverride` list is set, not only standard "bless" paths will be found but also custom ones.
   - Exclude options with blacklisted filenames (refer to
   - BlacklistAppleUpdate option).
   - On OpenCore boot partition exclude all OpenCore bootstrap files by header checks.
   - Register the resulting entries as primary options and determine their types if found.
     The option will become auxiliary for some types (e.g. Apple HFS recovery).
   - If partition already has primary options of "Apple Recovery" type proceed to next handle.
   - Lookup alternate entries by "bless" recovery option list retrieval and predefined paths.
   - Register the resulting entries as alternate auxiliary options and determine their types if found.
5. Custom entries and tools are added as primary options without any checks with respect to `Auxiliary`.
6. System entries (e.g. `Reset NVRAM`) are added as primary auxiliary options.

The display order of the boot options in the picker and the boot process are determined separately from the scanning algorithm. The display order as follows:

- Alternate options follow corresponding primary options, i.e. Apple recovery will be following the relevant macOS option whenever possible.
- Options will be listed in file system handle firmware order to maintain an established order across the reboots regardless of the chosen operating system for loading.
- Custom entries, tools, and system entries will be added after all other options.
- Auxiliary options will only show upon entering "Advanced Mode" in the picker (usually by pressing "Space").

The boot process is as follows:

- Try looking up first valid primary option through `BootNext` UEFI variable.
- On failure looking up first valid primary option through `BootOrder` UEFI variable.
- 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 (`DuetPkg` or custom BDS). Without `BootProtect` it also is possible that other operating systems overwrite OpenCore, make sure to enable it if you plan to use them.

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

8.2 Properties

1. `Boot`
   Type: `plist dict`
   Description: Apply boot configuration described in Boot Properties section below.

2. `BlessOverride`
   Type: `plist array`
   Description: Add custom scanning paths through bless model.

   Designed to be filled with `plist string` entries containing absolute UEFI paths to customised bootloaders, for example, `\EFI\Microsoft\Boot\bootmgfw.efi` for Microsoft bootloader. This allows unusual boot paths to be automaticly discovered by the boot picker. Designwise they are equivalent to predefined blessed path, such as `\System\Library\CoreServices\boot.efi`, but unlike predefined bless paths they have highest priority.

3. `Debug`
   Type: `plist dict`
   Description: Apply debug configuration described in Debug Properties section below.

4. `Entries`
   Type: `plist array`
   Description: Add boot entries to boot picker.

   Designed to be filled with `plist dict` values, describing each load entry. See Entry Properties section below.

5. `Security`
   Type: `plist dict`
   Description: Apply security configuration described in Security Properties section below.

6. `Tools`
   Type: `plist array`
   Description: Add tool entries to boot picker.

   Designed to be filled with `plist dict` values, describing each load entry. See Entry Properties section below.

   Note: Select tools, for example, UEFI Shell, are very dangerous and **MUST NOT** appear in production configurations, especially in vaulted ones and protected with secure boot, as they may be used to easily bypass secure boot chain.

8.3 Boot Properties

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

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

   - `0x00` — `EFI_BLACK`

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

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

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

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

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. `Clean NVRAM`).

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

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

5. `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

Log is attempted to be written in the safest manner, and thus is very slow. Ensure that `DisableWatchDog` is set to `true` when you use a slow drive.

8.5 Security Properties

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

2. `AllowSetDefault`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Allow `CTRL+Enter` and `CTRL+Index` handling to set the default boot option in boot picker.

3. `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. To perform authenticated restart one can use a dedicated terminal command: `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.

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

5. `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 (`Boot9696`) in UEFI variable storage at bootloader startup. For this option to work `RequestBootVarRouting` is required to be enabled.

   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 bootstraping OpenCore.

   Note 1: Some firmewares may have broken NVRAM, no boot option support, or various other incompatibilities of any kind. While unlikely, the use of this option may even cause boot failure. Use at your own risk on boards known to be compatible.

   Note 2: Be warned that ~~NVRAM reset will also~~ while NVRAM reset executed from OpenCore should not erase the boot option created in `Bootstrap`~~mode~~, executing NVRAM reset prior to loading OpenCore will remove it.

6. `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.

- `HardDrive` — Generic OS (mandatory).
- `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.

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.

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.

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

11.5 OpenRuntime

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

- NVRAM namespaces, allowing to isolate operating systems from accessing select variables (e.g. `RequestBootVarRouting` or `ProtectSecureBoot`).
- NVRAM proxying, allowing to manipulate multiple variables on variable updates (e.g. `RequestBootVarFallback`).
- Read-only and write-only NVRAM variables, enhancing the security of OpenCore, Lilu, and Lilu plugins, like VirtualSMC, which implements `AuthRestart` support.
- NVRAM isolation, allowing to protect all variables from being written from an untrusted operating system (e.g. `DisableVariableWrite`).
- UEFI Runtime Services memory protection management to workaround read-only mapping (e.g. `EnableWriteUnprotector`).

11.6 Properties

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