OpenCore

Reference Manual (0.7.3.4)

[2021.09.19]

Copyright ©2018-2021 vit9696

figure. Available entries include:

- `BOOTx64.efi` or `BOOTIa32.efi`
  Initial bootstrap loaders, which load `OpenCore.efi`. `BOOTx64.efi` is loaded by the firmware by default consistent with the UEFI specification. However, it may also be renamed and put in a custom location to allow OpenCore coexist alongside operating systems, such as Windows, that use `BOOTx64.efi` files as their loaders. Refer to the `LauncherOption` property for details.
- `boot`
  Duet bootstrap loader, which initialises the UEFI environment on legacy BIOS firmware and loads `OpenCore.efi` similarly to other bootstrap loaders. A modern Duet bootstrap loader will default to `OpenCore.efi` on the same partition when present.
- `ACPI`
  Directory used for storing supplemental ACPI information for the `ACPI` section.
- `Drivers`
  Directory used for storing supplemental `UEFI` drivers for `UEFI` section.
- `Kexts`
  Directory used for storing supplemental kernel information for the `Kernel` section.
- `Resources`
  Directory used for storing media resources such as audio files for screen reader support. Refer to the `UEFI Audio Properties` section for details. This directory also contains image files for graphical user interface. Refer to the OpenCanopy section for details.
- `Tools`
  Directory used for storing supplemental tools.
- `OpenCore.efi`
  Main booter application responsible for operating system loading. The directory `OpenCore.efi` resides in is called the `root directory`, which is set to `EFI\OC` by default. When launching `OpenCore.efi` directly or through a custom launcher however, other directories containing `OpenCore.efi` files are also 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 the `0`-terminator) will be accessible within OpenCore.

3.2 Installation and Upgrade

To install OpenCore, replicate the Configuration Structure described in the previous section in the EFI volume of a GPT partition. While corresponding sections of this document 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 the OpenCore repository. Vaulting information is provided in the Security Properties section of this document.

The `OC config` file, as with any property list file, can be edited with any text editor, such as nano or vim. However, specialised software may provide a better experience. On macOS, the preferred GUI application is Xcode. ~~For a lightweight~~The ProperTree editor is a lightweight, cross-platform and open-source alternative~~, the ProperTree editor can be utilised~~.

It is strongly ~~advised not to use any software that is~~ recommended to avoid configuration creation tools that are aware of the internal ~~configration structure as it constantly gets out of date and will cause incorrect configuration to be~~

Note: The need for this quirk is determined by early boot failures.

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

    This quirk is useful on Mac firmware, which loads different operating systems with different hardware configurations. For example, it is supposed to enable Intel GPU in Windows and Linux in some dual-GPU MacBook models.

19. `SyncRuntimePermissions`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Update memory permissions for the runtime environment.

    Some types of firmware fail to properly handle runtime permissions:

    - They incorrectly mark `OpenRuntime` as not executable in the memory map.
    - They incorrectly mark `OpenRuntime` as not executable in the memory attributes table.
    - They lose entries from the memory attributes table after `OpenRuntime` is loaded.
    - They mark items in the memory attributes table as read-write-execute.

    This quirk attempts to update the memory map and memory attributes table to correct this.

    Note: The need for this quirk is indicated by early boot failures —(e.g. halts at black screen), particularly in early boot of the Linux kernel. Only firmware released after 2017 is typically affected.

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 OpenLinuxBoot

`OpenLinuxBoot` is an OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL`. It detects and boots Linux distros which are installed according to the Boot Loader Specification or to the closely related (but not identical, see next paragraph) systemd BootLoaderSpecByDefault. In effect this means Linux distributions where the available boot options are found in `{ESP}/loader/entries/*.conf` files (for instance `/boot/efi/loader/entries/*.conf`) or in `{boot}/loader/entries/*.conf` files (for instance `/boot/loader/entries/*.conf`). The former layout – pure Boot Loader Specification, using kernel files on the EFI System Partition or Extended Boot Loader Partition – is specific to systemd-boot, the latter layout with kernel files typically on the partition which will be mounted as `/boot` applies to most Fedora-related distros including Fedora itself, RHEL and variants.

BootLoaderSpecByDefault includes the possibility of expanding GRUB variables in its `*.conf` files – and this is used in practice in certain distros such as CentOS. In order to correctly handle this, `OpenLinuxBoot` extracts all variables from `{boot}/grub2/grubenv` and any unconditionally set variables from `{boot}/grub2/grub.cfg`. This has proved sufficient in practice to extract the required variables seen so far in distros which use this GRUB-specific feature.

For distributions which do not use either of the above schemes, `OpenLinuxBoot` will autodetect and boot `{boot}/vmlinuz*` kernel files directly, after linking these automatically – based on the kernel version in the filename – to their associated `{boot}/init*` ramdisk files, and after searching in `/etc/default/grub` for kernel boot options and `/etc/os-release` for the distro name. This layout applies to most Debian-related distros, including Debian itself, Ubuntu and variants.

The method of starting the kernel relies on it being compiled with EFISTUB, however this applies to almost all modern distros, particularly those which use systemd. Most modern distros use systemd as their system manager (even though at the same time most do *not* use systemd-boot as their bootloader).

The latest kernel version of a given install is always shown in the boot menu. Additional versions, recovery versions, etc. are added as auxiliary boot entries, so depending on OpenCore's `HideAuxiliary` setting may not be shown until the space key is pressed.

Note 1: `OpenLinuxBoot` requires filesystem drivers that may not be available in firmware such as EXT4 and BTRFS drivers. These drivers can be obtained from external sources. Drivers tested in basic scenarios can be downloaded from OcBinaryData. Be aware that these drivers are neither tested for reliability in all scenarious, nor underwent any tamper-resistance testing, therefore have may carry potential security or data-loss risks.

Most Linux distributions keep their boot files on ~~the~~ an EXT4 ~~file system~~ partition even when the distribution's ~~main~~ root filesystem is something else, such as BTRFS, therefore ~~a suitable UEFI~~ only an EXT4 ~~file system~~ driver such as `ext4_x64` is normally required. A BTRFS driver such as `btrfs_x64` will be required in ~~a~~ the currently somewhat less standard ~~setup~~ situation where the boot files are on a BTRFS partition, e.g. as is currently done by default in openSUSE.

Pure Boot Loader Spec (e.g. as implemented by systemd-boot) keeps all kernel and ramdisk images directly on the EFI System Partition (or an Extended Boot Loader Partition), therefore it requires no additional filesystem driver - but it is not widely used except in Arch Linux.

Note 2: `OpenLinuxBoot` does not attempt to read and interpret the layout of Linux installation media (which can be highly variable). Installation media should be booted directly either from the machine's own EFI boot menu or from the OpenCore boot menu. In some cases, e.g. Apple T2 hardware, then – depending on OpenCore's security settings – OpenCore may be able to start some Linux installers which the machine's own bootloader will refuse to boot.

Note 3: systemd-boot users (probably almost exclusively Arch Linux users) should be aware that `OpenLinuxBoot` does not support the systemd-boot–specific Boot Loader Interface; therefore use `efibootmgr` rather than `bootctl` for any

low-level Linux command line interaction with the boot menu.

Note 4: Be aware of the `SyncRuntimePermissions` quirk, which may need to be set to avoid early boot failure (i.e. halts with black screen) of the Linux kernel due to a firmware bug of some firmware released after 2017.

The default parameter values should work well, but if you need to parameterise this driver the following options may be specified in `UEFI/Drivers/Arguments`:

- `flags` - Default: all flags except `LINUX_BOOT_ADD_DEBUG_INFO` are set.

  Available flags are:

  - `0x00000001` (bit `0`) — `LINUX_BOOT_SCAN_ESP`, Allows scanning for entries on EFI System Partition.
  - `0x00000002` (bit `1`) — `LINUX_BOOT_SCAN_XBOOTLDR`, Allows scanning for entries on Extended Boot Loader Partition.
  - `0x00000004` (bit `2`) — `LINUX_BOOT_SCAN_LINUX_ROOT`, Allows scanning for entries on Linux Root filesystems.
  - `0x00000008` (bit `3`) — `LINUX_BOOT_SCAN_LINUX_DATA`, Allows scanning for entries on Linux Data filesystems.
  - `0x00000080` (bit `7`) — `LINUX_BOOT_SCAN_OTHER`, Allows scanning for entries on file systems not matched by any of the above.

    The following notes apply to all of the above options:

    Note 1: Apple filesystems APFS and HFS are never scanned.

    Note 2: Regardless of the above flags, a file system must first be allowed by `Misc/Security/ScanPolicy` before it can be seen by `OpenLinuxBoot` or any other `OC_BOOT_ENTRY_PROTOCOL` driver.

    Note 3: It is recommended to enable scanning `LINUX_ROOT` and `LINUX_DATA` in both `OpenLinuxBoot` flags and `Misc/Security/ScanPolicy` in order to be sure to detect all valid Linux installs.

  - `0x00000100` (bit `8`) — `LINUX_BOOT_ALLOW_AUTODETECT`, If set allows autodetecting and linking `vmlinuz*` and `init*` ramdisk files when `loader/entries` files are not found.
  - `0x00000200` (bit `9`) — `LINUX_BOOT_USE_LATEST`, When a Linux entry generated by `OpenLinuxBoot` is selected as the default boot entry in OpenCore, automatically switch to the latest kernel when a new version is installed.

    When this option is set, an internal menu entry id is shared between kernel versions from the same install of Linux. Linux boot options are always sorted highest kernel version first, so this means that the latest kernel version of the same install always shows as the default, with this option set.

    Note: This option is recommended on all systems.

  - `0x00000400` (bit `10`) — `LINUX_BOOT_ADD_RO`, This option applies to autodetected Linux only (i.e. to Debian-style distrubutions, not to BLSpec and Fedora-style distributions with `/loader/entries/*.conf` files). Some distrubtions run a filesystem check on loading which requires the root filesystem to initially be mounted read-only via the `ro` kernel option. Set this bit to add this option on autodetected distros; should be harmless but very slightly slow down boot time (due to requried remount as read-write) on distros which do not require it. To specify this option for specific distros only, use `partuuidopts:{partuuid}+=ro` instead of this flag.
  - `0x00004000` (bit `14`) — `LINUX_BOOT_LOG_VERBOSE`, Add additional debug log info about files encountered and autodetect options added while scanning for Linux boot entries.
  - `0x00008000` (bit `15`) — `LINUX_BOOT_ADD_DEBUG_INFO`, Adds a human readable file system type, followed by the first eight characters of the partition's unique partition uuid, to each generated entry name. Can help with debugging the origin of entries generated by the driver when there are multiple Linux installs on one system.

  Flag values can be specified in hexadecimal beginning with `0x` or in decimal, e.g. `flags=0x80` or `flags=128`.

- `partuuidopts:{partuuid}[+]="{options}"` - Default: not set.

  Allows specifying kernel options for a given partition only. If specified with `+=` then these are used in addition to autodetected options, if specified with `=` they are used instead. Used for autodetected Linux only. Values specified here are never used for entries created from `/loader/entries/*.conf` files.

  Note: The `partuuid` value to be specified here is typically the same as the `PARTUUID` seen in `root=PARTUUID=...` in the Linux kernel boot options (view using `cat /proc/cmdline`) for autodetected Debian-style distros, but is NOT the same for Fedora-style distros booted from `/loader/entries/*.conf` files.

Enabling this setting plays the boot chime using the builtin audio support. The volume level is determined by the `MinimumVolume` and `VolumeAmplifier` settings as well as the `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 the boot chime.

7. `ResetTrafficClass`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Set HDA Traffic Class Select Register to `TC0`.

   AppleHDA kext will function correctly only if `TCSEL` register is configured to use `TC0` traffic class. Refer to Intel I/O Controller Hub 9 (ICH9) Family Datasheet (or any other ICH datasheet) for more details about this register.

   Note: This option is independent from `AudioSupport`. If AppleALC is used it is preferred to use AppleALC `alctsel` property instead.

8. `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. A typical delay can be up to 0.5 seconds.

9. `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.

11.11 Drivers Properties

1. `Comment`
   Type: `plist string`
   Failsafe: Empty
   Description: Arbitrary ASCII string used to provide human readable reference for the entry. Whether this value is used is implementation defined.

2. `Path`
   Type: `plist string`
   Failsafe: Empty
   Description: Path of file to be loaded as a UEFI driver from `OC/Drivers` directory.

3. `Enabled`
   Type: `plist boolean`
   Failsafe: `false`
   Description: If `false` this driver entry will be ignored.

4. `Arguments`
   Type: `plist string`
   Failsafe: Empty
   Description: Some OC plugins accept optional additional arguments which may be specified as a string here.