OpenCore

Reference Manual (0.7.~~2~~.3)

[2021.09.04]

open-source implementations with transparent binary generation is encouraged (e.g. OCAT), since other tools may contain malware. Remember that a configuration made for a different hardware setup shall never be used on another hardware setup.

For BIOS booting, a third-party UEFI environment provider is required and `OpenDuetPkg` is one such UEFI environment provider 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 the `Differences.pdf` document which provides information about changes to the configuration (as compared to the previous release) as well as to the `Changelog.md` document (which contains a list of modifications across all published updates).

3.3 Contribution

OpenCore can be compiled as a standard EDK II package and requires the EDK II Stable package. The currently supported EDK II release is hosted in acidanthera/audk. Required patches for this package can be found in the `Patches` directory.

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

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

```
git clone --depth=1 https://github.com/acidanthera/audk UDK
cd UDK
git submodule update --init --recommend-shallow
git clone --depth=1 https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
. ./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 using Language Server Protocols. For example, Sublime Text with LSP for Sublime Text plugin. Add `compile_flags.txt` file with similar content to the 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/OpenCorePkg/Include/AMI
-I/UefiPackages/OpenCorePkg/Include/Acidanthera
-I/UefiPackages/OpenCorePkg/Include/Apple
-I/UefiPackages/OpenCorePkg/Include/Apple/X64
-I/UefiPackages/OpenCorePkg/Include/Duet
-I/UefiPackages/OpenCorePkg/Include/Generic
-I/UefiPackages/OpenCorePkg/Include/Intel
-I/UefiPackages/OpenCorePkg/Include/Microsoft
-I/UefiPackages/OpenCorePkg/Include/Nvidia
-I/UefiPackages/OpenCorePkg/Include/VMware
-I/UefiPackages/OvmfPkg/Include
-I/UefiPackages/ShellPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-include
```

Warning: Certain firmware appear to have defective NVRAM garbage collection. As a result, they may not be able to always free space after variable deletion. Do not enable `non-volatile` NVRAM logging on such devices unless specifically required.

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

File logging will create a file named `opencore-YYYY-MM-DD-HHMMSS.txt` (in UTC) under the EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmware are not reliable and may corrupt data when writing files through UEFI. Log writing is attempted in the safest manner and thus, is very slow. Ensure that `DisableWatchDog` is set to `true` when a slow drive is used. Try to avoid frequent use of this option when dealing with flash drives as large I/O amounts may speed up memory wear and render the flash drive unusable quicker.

When interpreting the log, note that the lines are prefixed with a tag describing the relevant location (module) of the log line allowing better attribution of the line to the functionality.

The list of currently used tags is as follows.

Drivers and tools:

- `BMF` — OpenCanopy, bitmap font
- `BS` — Bootstrap
- `GSTT` — GoptStop
- `HDA` — AudioDxe
- `KKT` — KeyTester
- `LNX` — OpenLinuxBoot
- `MMDD` — MmapDump
- `OCPAVP` — PavpProvision
- `OCRST` — ResetSystem
- `OCUI` — OpenCanopy
- `OC` — OpenCore main, also OcMainLib
- `VMOPT` — VerifyMemOpt

Libraries:

- `AAPL` — OcDebugLogLib, Apple EfiBoot logging
- `OCABC` — OcAfterBootCompatLib
- `OCAE` — OcAppleEventLib
- `OCAK` — OcAppleKernelLib
- `OCAU` — OcAudioLib
- `OCA` —- OcAcpiLib
- `OCBP` — OcAppleBootPolicyLib
- `OCB` — OcBootManagementLib
- `OCLBT` — OcBlitLib
- `OCCL` — OcAppleChunkListLib
- `OCCPU` — OcCpuLib
- `OCC` — OcConsoleLib
- `OCDC` — OcDriverConnectionLib
- `OCDH` — OcDataHubLib
- `OCDI` — OcAppleDiskImageLib
- `OCDM` — OcDeviceMiscLib
- `OCFS` — OcFileLib
- `OCFV` — OcFirmwareVolumeLib
- `OCHS` — OcHashServicesLib
- `OCI4` — OcAppleImg4Lib
- `OCIC` — OcImageConversionLib
- `OCII` — OcInputLib
- `OCJS` — OcApfsLib
- `OCKM` — OcAppleKeyMapLib
- `OCL` — OcDebugLogLib

- `CSR_ALLOW_UNAUTHENTICATED_ROOT` (`0x800`) is not practical as it prevents incremental (non-full) OTA updates.

Note3: For any other value which you may need to use, it is possible to configure `CsrUtil.efi` as a `TextMode Tools` entry to configure a different value, e.g. use `toggle 0x6F` in `Arguments` to toggle the SIP disabled value set by default by `csrutil disable --no-internal` in Big Sur.

4. `ApECID`
   Type: `plist integer`, 64 bit
   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, generate a random 64-bit number with a cryptographically secure random number generator. As an alternative, the 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 must be reinstalled or personalised. Unless the operating system is personalised, macOS DMG recovery cannot be loaded. In cases where DMG recovery is missing, it can be downloaded by using the `macrecovery` utility and saved in `com.apple.recovery.boot` as explained in the 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 the `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 --folder "/Volumes/Macintosh HD/System/Library/CoreServices" \
     --bootefi --personalize
   ```

   On macOS 11 and newer the dedicated `x86legacy` model always uses `ApECID`. When this configuration setting is left as `0` first 8 bytes of `system-id` variable are used instead.

   On macOS versions 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, the macOS Installer from macOS 10.15 and older will often 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 the 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
   ```

5. `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 restarts by splitting and saving disk encryption keys between NVRAM and RTC, which despite being removed as soon as OpenCore starts, may be considered a security risk and thus is optional.

```
rm vault.pub
```

Note 1: While it may appear obvious, an external method is required to verify `OpenCore.efi` and `BOOTx64.efi` for secure boot path. For this, it is recommended to enable UEFI SecureBoot using a custom certificate and to sign `OpenCore.efi` and `BOOTx64.efi` with a custom key. More details on customising secure boot on modern firmware can be found in the Taming UEFI SecureBoot paper (in Russian).

Note 2: `vault.plist` and `vault.sig` are used regardless of this option when `vault.plist` is present or a public key is embedded into `OpenCore.efi`. Setting this option will only ensure configuration sanity, and abort the boot process otherwise.

14. `ScanPolicy`
Type: `plist integer`, 32 bit
Failsafe: `0x10F0103`
Description: Define operating system detection policy.

This value allows preventing scanning (and booting) untrusted sources based on a bitmask (sum) of a set of flags. As it is not possible to reliably detect every file system or device type, this feature cannot be fully relied upon in open environments, and additional measures are to be applied.

Third party drivers may introduce additional security (and performance) consideratons following the provided scan policy. The active Scan policy is exposed in the `scan-policy` variable of `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` GUID for UEFI Boot Services only.

- `0x00000001` (bit `0`) — `OC_SCAN_FILE_SYSTEM_LOCK`, restricts scanning to only known file systems defined as a part of this policy. File system drivers may not be aware of this policy. Hence, to avoid mounting of undesired file systems, drivers for such file systems should not be loaded. This bit does not affect DMG mounting, which may have any file system. Known file systems are prefixed with `OC_SCAN_ALLOW_FS_`.
- `0x00000002` (bit `1`) — `OC_SCAN_DEVICE_LOCK`, restricts scanning to only known device types defined as a part of this policy. It is not always possible to detect protocol tunneling, so be aware that on some systems, it may be possible for e.g. USB HDDs to be recognised as SATA instead. Cases like this must be reported. Known device types are prefixed with `OC_SCAN_ALLOW_DEVICE_`.
- `0x00000100` (bit `8`) — `OC_SCAN_ALLOW_FS_APFS`, allows scanning of APFS file system.
- `0x00000200` (bit `9`) — `OC_SCAN_ALLOW_FS_HFS`, allows scanning of HFS file system.
- `0x00000400` (bit `10`) — `OC_SCAN_ALLOW_FS_ESP`, allows scanning of EFI System Partition file system.
- `0x00000800` (bit `11`) — `OC_SCAN_ALLOW_FS_NTFS`, allows scanning of NTFS (Msft Basic Data) file system.
- `0x00001000` (bit `12`) — `OC_SCAN_ALLOW_FS_EXTLINUX_ROOT`, allows scanning of ~~EXT (Linux Root~~ Linux Root file systems.
- `0x00002000` (bit `13`) — `OC_SCAN_ALLOW_FS_LINUX_DATA`, allows scanning of Linux Data file systems.
- `0x00004000` (bit `14`) ~~file system~~ — `OC_SCAN_ALLOW_FS_XBOOTLDR`, allows scanning the Extended Boot Loader Partition as defined by the Boot Loader Specification.
- `0x00010000` (bit `16`) — `OC_SCAN_ALLOW_DEVICE_SATA`, allow scanning SATA devices.
- `0x00020000` (bit `17`) — `OC_SCAN_ALLOW_DEVICE_SASEX`, allow scanning SAS and Mac NVMe devices.
- `0x00040000` (bit `18`) — `OC_SCAN_ALLOW_DEVICE_SCSI`, allow scanning SCSI devices.
- `0x00080000` (bit `19`) — `OC_SCAN_ALLOW_DEVICE_NVME`, allow scanning NVMe devices.
- `0x00100000` (bit `20`) — `OC_SCAN_ALLOW_DEVICE_ATAPI`, allow scanning CD/DVD devices and old SATA.
- `0x00200000` (bit `21`) — `OC_SCAN_ALLOW_DEVICE_USB`, allow scanning USB devices.
- `0x00400000` (bit `22`) — `OC_SCAN_ALLOW_DEVICE_FIREWIRE`, allow scanning FireWire devices.
- `0x00800000` (bit `23`) — `OC_SCAN_ALLOW_DEVICE_SDCARD`, allow scanning card reader devices.
- `0x01000000` (bit `24`) — `OC_SCAN_ALLOW_DEVICE_PCI`, allow scanning devices directly connected to PCI bus (e.g. VIRTIO).

Note: Given the above description, a value of `0xF0103` is expected to do the following:

- Permit scanning SATA, SAS, SCSI, and NVMe devices with APFS file systems.
- Prevent scanning any devices with HFS or FAT32 file systems.
- Prevent scanning APFS file systems on USB, CD, and FireWire drives.

The combination reads as:

- `OC_SCAN_FILE_SYSTEM_LOCK`
- `OC_SCAN_DEVICE_LOCK`

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 loading additional UEFI modules as well as applying tweaks to 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:

`AudioDxe*`	HDA audio support driver in UEFI firmware for most Intel and some other analog audio controllers. Staging driver, refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
`btrfs_x64`	Open source BTRFS file system driver, required for booting with OpenLinuxBoot from a file system which is now quite commonly used with Linux.
`CrScreenshotDxe*`	Screenshot making driver saving images to the root of OpenCore partition (ESP) or any available writeable filesystem upon pressing `F10`. This is a modified version of `CrScreenshotDxe` driver by Nikolaj Schlej.
`ExFatDxe`	Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `ExFatDxeLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`ext4_x64`	Open source EXT4 file system driver, required for booting with OpenLinuxBoot from the file system most commonly used with Linux.
`HfsPlus`	Recommended. Proprietary HFS file system driver with bless support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `HfsPlusLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`HiiDatabase*`	HII services support driver from `MdeModulePkg`. This driver is included in most types of firmware starting with the Ivy Bridge generation. Some applications with GUI, such as UEFI Shell, may need this driver to work properly.
`EnhancedFatDxe`	FAT filesystem driver from `FatPkg`. This driver is embedded in all UEFI firmware and cannot be used from OpenCore. Several types of firmware have defective FAT support implementation that may lead to corrupted filesystems on write attempts. Embedding this driver within the firmware may be required in case writing to the EFI partition is needed during the boot process.
`NvmExpressDxe*`	NVMe support driver from `MdeModulePkg`. This driver is included in most firmware starting with the Broadwell generation. For Haswell and earlier, embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
`OpenCanopy*`	OpenCore plugin implementing graphical interface.
`OpenRuntime*`	OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol.
`OpenLinuxBoot*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to allow direct detection and booting of Linux distributiuons from OpenCore, without chainloading via GRUB.
`OpenUsbKbDxe*`	USB keyboard driver adding support for `AppleKeyMapAggregator` protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin `KeySupport`, which may work better or worse depending on the firmware.
`OpenPartitionDxe*`	Partition management driver with Apple Partitioning Scheme support. This driver can be used to support loading older DMG recoveries such as macOS 10.9 using Apple Partitioning Scheme. OpenDuet already includes this driver.
`Ps2KeyboardDxe*`	PS/2 keyboard driver from `MdeModulePkg`. `OpenDuetPkg` and some types of firmware may not include this driver, but it is necessary for PS/2 keyboard to work. Note, unlike `OpenUsbKbDxe` this driver has no `AppleKeyMapAggregator` support and thus requires `KeySupport` to be enabled.
`Ps2MouseDxe*`	PS/2 mouse driver from `MdeModulePkg`. Some very old laptop firmware may not include this driver but it is necessary for the touchpad to work in UEFI graphical interfaces such as `OpenCanopy`.

OpenHfsPlus*	HFS file system driver with bless support. This driver is an alternative to a closed source HfsPlus driver commonly found in Apple firmware. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit.
UsbMouseDxe*	USB mouse driver from MdeModulePkg. Some virtual machine firmware such as OVMF may not include this driver but it is necessary for the mouse to work in UEFI graphical interfaces such as OpenCanopy.
XhciDxe*	XHCI USB controller support driver from MdeModulePkg. This driver is included in most types of firmware starting with the Sandy Bridge generation. For earlier firmware or legacy systems, it may be used to support external USB 3.0 PCI cards.

Driver marked with * are bundled with OpenCore. To compile the drivers from UDK (EDK II) the same command used for OpenCore compilation can be taken, but choose a corresponding package:

```
git clone https://github.com/acidanthera/audk UDK
cd UDK
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p FatPkg/FatPkg.dsc
build -a X64 -b RELEASE -t XCODE5 -p MdeModulePkg/MdeModulePkg.dsc
```

11.3 Tools 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 (Refer to the Tools subsection for more details), 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. It is typically 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 the bless command, 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.
CsrUtil*	Simple implementation of SIP-related features of Apple csrutil.
GopStop*	Test GraphicsOutput protocol with a simple scenario.
KeyTester*	Test keyboard input in SimpleText mode.
MemTest86	Memory testing utility.
OpenControl*	Unlock and lock back NVRAM protection for other tools to be able to get full NVRAM access when launching from OpenCore.
OpenShell*	OpenCore-configured UEFI Shell for compatibility with a broad range of 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.
ControlMsrE2*	Check CFG Lock (MSR 0xE2 write protection) consistency across all cores and change such hidden options on selected platforms.
TpmInfo*	Check Intel PTT (Platform Trust Technology) capability on the platform, which allows using fTPM 2.0 if enabled. The tool does not check whether fTPM 2.0 is actually enabled.

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 EXT4 file system even when the distribution's main filesystem is something else such as BTRFS, therefore a suitable UEFI EXT4 file system driver such as `ext4_x64` is normally required. A BTRFS driver such as `btrfs_x64` will be required in a somewhat less standard setup where the boot files are on a BTRFS partition, e.g. as 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: 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.

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.
  - `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.

  Typically you should not need this option in the latter case, but in case you do, to find out the unique partition uuid to use, look for `LNX:` entries in the OpenCore debug log file. Alternatively, and for more advanced scenarios, you may wish to examine how your drives are mounted using the Linux `mount` command, and then find out the partuuid of relevant mounted drives by examining the output of `ls -l /dev/disk/by-partuuid`.

- autoopts[+]="{options}" - Default: None specified. The kernel options to use for autodetected Linux only. The value here is never used for entries created from /loader/entries/*.conf files. partuuidopts may be more suitable where there are multiple distros, but autoopts with no PARTUUID required is more convenient for just one distro. If specified with += then these are used in addition to autodetected options, if specified with = they are used instead. As example usage, it is possible to use += format to add a vt.handoff options, such as autopts+="vt.handoff=7" or autopts+="vt.handoff=3" (check cat /proc/cmdline when booted via your existing bootloader) on Ubuntu and related distros, in order to add the vt.handoff option to the auto-detected GRUB defaults, and avoid a flash of text showing before the distro splash screen.

  Users may wish to compare their Linux boot options (shown with cat /proc/cmdline) seen when booting via OpenLinuxBoot and via their distro's original bootloader, which is normally GRUB (but might also be e.g. systemd-boot or EXTLINUX). Expect the options generated by OpenLinuxBoot not to contain a BOOT_IMAGE=... value where GRUB options do, and to contain an initrd=... value where the GRUB options do not, since GRUB hands over ramdisks in a different way. All remaining parameters should match, however – perhaps excluding less important graphics handover options, such as in the Ubuntu example given in autoopts. OpenLinuxBoot will not start a distro unless it can find some configured options to use, therefore in the hopefully unlikely case where no auto-detectable options are available, the user will need to specify the correct options with partuuidopts or autoopts before the distro will boot. Examine the OpenCore debug log for LNX: entries containing further information about what was found.

11.7 Properties

1. APFS
   Type: plist dict
   Failsafe: None
   Description: Provide APFS support as configured in the APFS Properties section below.

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

   Audio support provides a way for upstream protocols to interact with the selected hardware and audio resources. All audio resources should reside in \EFI\OC\Resources\Audio directory. Currently the supported audio file formats are MP3 and WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

   Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: [audio type]_[audio localisation]_[audio path].[audio ext]. For unlocalised files filename does not include the language code and looks as follows: [audio type]_[audio path].[audio ext]. Audio extension can either be mp3 or wav.

   - Audio type can be OCEFIAudio for OpenCore audio files or AXEFIAudio for macOS bootloader audio files.
   - Audio localisation is a two letter language code (e.g. en) with an exception for Chinese, Spanish, and Portuguese. Refer to APPLE_VOICE_OVER_LANGUAGE_CODE definition for the list of all supported localisations.
   - Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to APPLE_VOICE_OVER_AUDIO_FILE definition. For OpenCore audio paths refer to OC_VOICE_OVER_AUDIO_FILE definition. The only exception is OpenCore boot chime file, which is OCEFIAudio_VoiceOver_Boot.mp3.

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

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

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

Note: Some types of firmware, particularly those made by Apple, only connect the boot drive to speed up the boot process. Enable this option to be able to see all the boot options when running multiple drives.

4. `Drivers`
   Type: `plist ~~array~~dict`
   Failsafe: None
   Description: Load selected drivers from `OC/Drivers` directory using the settings specified in the Drivers Properties section below.

   ~~To be filled with string filenames meant to be loaded as UEFI drivers.~~

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

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

7. `ProtocolOverrides`
   Type: `plist dict`
   Failsafe: None
   Description: Force builtin versions of certain protocols described in the ProtocolOverrides Properties section below.

   Note: all protocol instances are installed prior to driver loading.

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

9. `ReservedMemory`
   Type: `plist array`
   Description: To be filled with `plist dict` values, describing memory areas exclusive to specific firmware and hardware functioning, which should not be used by the operating system. Examples of such memory regions could be the second 256 MB corrupted by the Intel HD 3000 or an area with faulty RAM. Refer to the ReservedMemory Properties section below for details.

11.8 APFS Properties

1. `EnableJumpstart`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load embedded APFS drivers from APFS containers.

   An APFS EFI driver is bundled in all bootable APFS containers. This option performs the loading of signed APFS drivers (consistent with the `ScanPolicy`). Refer to the "EFI Jumpstart" section of the Apple File System Reference for details.

2. `GlobalConnect`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform full device connection during APFS loading.

   Every handle is connected recursively instead of the partition handle connection typically used for APFS driver loading. This may result in additional time being taken but can sometimes be the only way to access APFS partitions on certain firmware, such as those on older HP laptops.

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

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

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

11.12 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).