OpenCore

Reference Manual (1.0.3.4)

[2024.12.22]

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

To obtain Data Hub logs, use the following command in macOS (Note that Data Hub logs do not log kernel and kext patches):

```
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. To maintain system integrity, the 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 the non-volatile flag will cause the log to be written to NVRAM flash after every printed line.

To obtain UEFI variable logs, 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 1: 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.

Warning 2: It is possible to enable fast file logging, which requires a fully compliant firmware FAT32 driver. On drivers with incorrect FAT32 write support (e.g. APTIO IV, but maybe others) this setting can result in corruption up to and including an unusable ESP filesystem, therefore be prepared to recreate the ESP partition and all of its contents if testing this option. This option can increase logging speed significantly on some suitable firmware, but may make little speed difference on some others.

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
- ~~NTBT~~NETB — OpenNetworkBoot
- MMDD — MmapDump
- OCPAVP — PavpProvision
- OCRST — ResetSystem
- OCUI — OpenCanopy
- OC — OpenCore main, also OcMainLib
- OLB — OpenLegacyBoot
- VMOPT — VerifyMemOpt

Libraries:

- AAPL — OcLogAggregatorLib, Apple EfiBoot logging

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.
`BiosVideo*`	CSM video driver implementing graphics output protocol based on VESA and legacy BIOS interfaces. Used for UEFI firmware with fragile GOP support (e.g. low resolution). Requires `ReconnectGraphicsOnConnect`. Included in OpenDuet out of the box.
`CrScreenshotDxe*`	Screenshot making driver saving images to the root of OpenCore partition (ESP) or any available writeable filesystem upon pressing `F10`. Accepts optional driver argument `--enable-mouse-click` to additionally take screenshot on mouse click. (It is recommended to enable this option only if a keypress would prevent a specific screenshot, and disable it again after use.) This is a modified version of `CrScreenshotDxe` driver by Nikolaj Schlej.
`EnableGop{Direct}*`	Early beta release firmware-embeddable driver providing pre-OpenCore non-native GPU support on MacPro5,1. Installation instructions can be found in the `Utilities/EnableGop` directory of the OpenCore release zip file - proceed with caution.
`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.
`FirmwareSettings*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to add an entry to the boot picker menu which reboots into UEFI firmware settings, when this is supported by the firmware.
`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.
`OpenLegacyBoot*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to allow detection and booting of legacy operating systems from OpenCore on Macs, OpenDuet and systems with a CSM.

Fedora `BootLoaderSpecByDefault` (but not pure Boot Loader Specification) can expand GRUB variables in the `*.conf` files – and this is used in practice in certain distros such as CentOS. In order to handle this correctly, when this situation is detected OpenLinuxBoot extracts all variables from `{boot}/grub2/grubenv` and also any unconditionally set variables from `{boot}/grub2/grub.cfg`, and then expands these where required in `*.conf` file entries.

The only currently supported method of starting Linux kernels from OpenLinuxBoot relies on their being compiled with EFISTUB. This applies to almost all modern distros, particularly those which use systemd. Note that most modern distros use systemd as their system manager, even though most do not use systemd-boot as their bootloader.

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 `efibootmgr` rather than `bootctl` must be used for any low-level Linux command line interaction with the boot menu.

11.8 OpenNetworkBoot

OpenNetworkBoot is an OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL`. It enables PXE and HTTP(S) Boot options in the OpenCore menu if these are supported by the underlying firmware, or if the required network boot drivers have been loaded using OpenCore.

It has additional support for loading `.dmg` files and their associated `.chunklist` file over HTTP(S) Boot, allowing macOS recovery to be started over HTTP(S) Boot: if either extension is seen in the HTTP(S) Boot URI then the other file of the pair is automatically loaded as well, and both are passed to OpenCore to verify and boot from the DMG file.

PXE Boot is already supported on most firmware, so in most cases PXE Boot entries should appear as soon as the driver is loaded. Using the additional network boot drivers provided with OpenCore, when needed, HTTP(S) Boot should be available on most firmware even if not natively supported.

Detailed information about the available network boot drivers and how to configure PXE and HTTP(S) Boot is provided on this page.

The ~~following~~ below configuration options may be specified in the `Arguments` section for this driver~~:~~ .

Note: There is no problem if configuration options within `<Arguments>...</Arguments>` are given on multiple lines, and option values enclosed within quotes can also span multiple lines. This applies to all drivers which use OpenCore argument parsing, and can be particularly convenient when multiple long options such as `uri`, `static4` and particularly `enroll-cert` may be needed.

- ~~`-4` - Boolean flag, enabled if present.~~

  ~~If specified enable IPv4 for PXE and HTTP(S) Boot. Disable IPV6 unless the `-6` flag is also present. If neither flag is present, both are enabled by default.~~

- ~~`-6` - Boolean flag, enabled if present.~~

  ~~If specified enable IPv6 for PXE and HTTP(S) Boot. Disable IPV4 unless the `-4` flag is also present. If neither flag is present, both are enabled by default.~~

- ~~`--aux`~~`aux` - Boolean flag, enabled if present.

  If specified the driver will generate auxiliary boot entries.

- ~~`--delete-all-certs`~~`delete-all-certs[:{OWNER_GUID}]` - Default: not set.

  If specified, delete all certificates present for `OWNER_GUID`. `OWNER_GUID` is optional, and will default to all zeros if not specified.

- ~~`--delete-cert`~~`delete-cert[:{OWNER_GUID}]="{cert-text}"` - Default: not set.

  If specified, delete the given certificate(s) for HTTPS Boot. The certificate(s) can be specified as a multi-line PEM value between double quotes. ~~`OWNER_GUID` is optional, and will default to all zeros if not specified.~~ A single PEM file can contain one or more certicates. Multiple instances of this option can be used to delete multiple different PEM files, if required. `OWNER_GUID` is optional, and will default to all zeros if not specified.

- ~~--enroll-cert~~enroll-cert[:{OWNER_GUID}]="{cert-text}" - Default: not set.

  If specified, enroll the given certificate(s) for HTTPS Boot. The certificate(s) can be specified as a multi-line PEM value between double quotes. ~~OWNER_GUID is optional, and will default to all zeros if not specified.~~ A single PEM file can contain one or more certicates. Multiple instances of this option can be used to enroll multiple different PEM files, if required. OWNER_GUID is optional, and will default to all zeros if not specified.

- ~~--http~~http - Boolean flag, enabled if present.

  If specified enable HTTP(S) Boot. Disable PXE Boot unless the ~~--pxe~~pxe flag is also present. If neither flag is present, both are enabled by default.

- ~~--https~~https - Boolean flag, enabled if present.

  If enabled, allow only `https://` URIs for HTTP(S) Boot. Additionally has the same behaviour as the ~~--http~~http flag.

- ~~--pxe~~ipv4 - Boolean flag, enabled if present.

  If specified enable ~~PXE Boot, and~~ IPv4 for PXE and HTTP(S) Boot. Disable IPV6 unless the ipv6 flag is also present. If neither flag is present, both are enabled by default.

- ipv6 - Boolean flag, enabled if present.

  If specified enable IPv6 for PXE and HTTP(S) Boot. Disable IPV4 unless the ipv4 flag is also present. If neither flag is present, both are enabled by default.

- pxe - Boolean flag, enabled if present.

  If specified enable PXE Boot, and disable HTTP(S) Boot unless the ~~--http~~http or ~~--https~~https flags are present. If none of these flags are present, both PXE and HTTP(S) Boot are enabled by default.

- ~~--uri~~static4:{MAC_ADDR}[\VLAN_ID][="{IP},{MASK},{GATEWAY}[,{DNS}]"] - String value~~, no default~~.

  Specify static IPv4 address for the network interface with the MAC address given by MAC_ADDR. MAC_ADDR must be specified as 12 consecutive hex digits, with no spaces, colons or hyphens separating digit pairs. In some advanced use-cases such as iSCSI, the MAC address length may be some other even number length of hex digits. The required MAC address can be found in the names of the boot options produced by this driver. Note that hyphens separating digit pairs must be removed, as compared to the format displayed in boot option names. It is also possible to specify a VLAN ID to use on the interface, by adding a backslash followed by a 4 digit hex representation of the VLAN ID following the MAC address. The VLAN ID will also be shown in the boot entry name, but note that it must be converted from decimal in the boot entry name to a 4 digit hex number in this option.

  Required elements in value are IP address in IP, network mask in MASK and gateway in GATEWAY. Optional is an additional space separated list of one or more DNS servers in DNS. DNS will be needed if the boot file URI includes a domain name rather than an IP address.

  MAC_ADDR is not optional.

  If value is omitted, then any static IP for this MAC address (and VLAN ID when present) will be deleted.

  - Example 1: static4:112233445566="192.168.1.20,255.255.255.0,192.168.1.1,8.8.8.8 4.4.4.4".
  - Example 2: static4:112233445566\0001="10.0.0.2,255.255.255.0,10.0.0.1".

  Note 1: This option is written to NVRAM and will remain present even if the option is removed from the driver Arguments, unless NVRAM is cleared or an alternative value is written or the value deleted, using this option.

  Note 2: This setting will normally cause a static IP to be assigned during pre-boot, even in vendor-provided network stacks. However, due to a quirk of the design of PXE and HTTP boot, any such static assignment

will then be ignored and DHCP used instead, during network boot. The OpenCore network stack (specifically `HttpBootDxe.efi`) is unusual in that it will allow HTTP boot from a static IP address, as long as an HTTP boot URI has also been specified, using the `uri` option for this driver (or e.g. in the OVMF admin screens if using OVMF, or similar options where present in other firmeare). If HTTP boot from static IP is required, then any pre-existing vendor-specific version of `HttpBootDxe.efi` will need to be unloaded (see `UEFI` `Unload` option) and the OpenCore version used instead.

Note 3: If `Ip4Dxe` is loaded before OpenCore then any setting here will only take effect after one reboot. If `Ip4Dxe` is loaded after `OpenNetworkBoot` the setting will take effect immediately.

Note 4: In the majority of cases this option is not required, and the default DHCP behaviour should be preferred, since IP address conflicts are automatically avoided, and any IP address assigned by DHCP during network boot will normally automatically match the IP address assigned in-OS, as the same MAC address is used in both cases.

- `uri` - String value, no default.

  If present, specify the URI to use for HTTP(S) Boot. If not present then DHCP boot options must be enabled on the network in order for HTTP(S) Boot to know what to boot.

11.8.1 OpenNetworkBoot Certificate Management

Certificates are enrolled to NVRAM storage, therefore once a certificate has been enrolled, it will remain enrolled even if the ~~`--enroll-cert`~~`enroll-cert` config option is removed. ~~`--delete-cert`~~`delete-cert` or ~~`--delete-all-certs`~~`delete-all-cert` should be used to remove enrolled certificates.

Checking for certificate presence by the ~~`--enroll-cert`~~`enroll-cert` and ~~`--delete-cert`~~`delete-cert` options uses the simple algorithm of matching by exact file contents, not by file meaning. The intended usage is to leave an ~~`--enroll-cert`~~`enroll-cert` option present in the config file until it is time to delete it, e.g. after another more up-to-date ~~`--enroll-cert`~~`enroll-cert` option has been added and tested. At this point the user can change ~~`--enroll-cert`~~`enroll-cert` to ~~`--delete-cert`~~`delete-cert` for the old certificate.

Certificate options are processed one at a time, in order, and each will potentially make changes to the certificate NVRAM storage. However each option will not change the NVRAM store if it is already correct for the option at that point in time (e.g. will not enroll a certificate if it is already enrolled). Avoid combinations such as ~~`--delete-all-certs`~~`delete-all-certs` followed by ~~`--enroll-cert`~~`enroll-cert`, as this will modify the NVRAM certificate storage twice on every boot. However a combination such as ~~`--delete-cert`~~`delete-cert``="{certA-text}"` followed by ~~`--enroll-cert`~~`enroll-cert``="{certB-text}"` (with `certA-text` and `certB-text` different) is safe, because certA will only be deleted if it is present and certB will only be added if it is not present, therefore no NVRAM changes will be made on the second and subsequent boots with these options.

In some cases (such as OVMF with https:// boot support) the `OpenNetworkBoot` certificate configuration options manage the same certificates as those seen in the firmware UI. In other cases of vendor customised HTTPS Boot firmware, the certificates managed by this driver will be separate from those managed by firmware.

When using the debug version of this driver, the OpenCore debug log includes ~~`NTBT`~~`NETB:` entries that show which certificates are enrolled and removed by these options, and which certificates are present after all certificate configuration options have been processed.

11.9 Other Boot Entry Protocol drivers

In addition to the OpenLinuxBoot and OpenNetworkBoot plugins, the following `OC_BOOT_ENTRY_PROTOCOL` plugins are made available to add optional, configurable boot entries to the OpenCore boot picker.

11.9.1 ResetNvramEntry

Adds a menu entry which resets NVRAM and immediately restarts. Additionally adds support for hotkey `CMD+OPT+P+R` to perform the same action. Note that on some combinations of firmware and drivers, the `TakeoffDelay` option must be configured in order for this and other builtin hotkeys to be reliably detected.

4. `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.

5. `Drivers`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected drivers from `OC/Drivers` directory.

   To be filled with `plist dict` values, describing each driver. Refer to the Drivers Properties section below.

6. `Input`
   Type: `plist dict`
   Description: Apply individual settings designed for input (keyboard and mouse) in the Input Properties section below.

7. `Output`
   Type: `plist dict`
   Description: Apply individual settings designed for output (text and graphics) in the Output Properties section below.

8. `ProtocolOverrides`
   Type: `plist dict`
   Description: Force builtin versions of certain protocols described in the ProtocolOverrides Properties section below.

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

9. `Quirks`
   Type: `plist dict`
   Description: Apply individual firmware quirks described in the Quirks Properties section below.

10. `ReservedMemory`
    Type: `plist array`
    Failsafe: Empty
    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. `Unload`
    Type: `plist array`
    Failsafe: Empty
    Description: Unload specified firmware drivers.

    To be filled with `plist string` entries containing the names of firmware drivers to unload before loading the `Drivers` section. This setting is typically only required if a user-provided driver is a variant of an existing system firmware driver, and if the new driver would detect itself as partially loaded, or otherwise fail to operate correctly, if the old driver is not unloaded first.

    Warning: Unloading system firmware drivers is usually not required and not recommended. Poorly written drivers may crash when unloaded, or cause subsequent crashes (e.g by allowing themselves to be unloaded even though they have active dependencies). However standard UEFI network stack drivers should unload cleanly.

    Note 1: Drivers specified in this option will be unloaded in the reverse of the order in which they were loaded, regardless of the order in which they are specified here.

Note 2: See `SysReport/Drivers/DriverImageNames.txt` for the list of drivers which this option can attempt to unload. The relevant name is the driver component name. Drivers are only listed if they implement `DriverBindingProtocol` and `LoadedImageProtocol`, and have an available component name.

Note ~~2~~3: The NVRAM `Lang` and `PlatformLang` variables are ignored when determining the driver component names recognised by this option, and listed in the `SysReport` file. This is in order to make unloading images stable across changes in these variables. The UEFI Shell `dh` command takes account of these variables, so in some circumstances may display different driver component names from those listed for this option, unless these variables are cleared.

11.13 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.

3. `HideVerbose`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Hide verbose output from APFS driver.

   APFS verbose output can be useful for debugging.

4. `JumpstartHotPlug`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load APFS drivers for newly connected devices.

   Permits APFS USB hot plug which enables loading APFS drivers, both at OpenCore startup and during OpenCore picker display. Disable if not required.

5. `MinDate`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal allowed APFS driver date.

   The APFS driver date connects the APFS driver with the calendar release date. Apple ultimately drops support for older macOS releases and APFS drivers from such releases may contain vulnerabilities that can be used to compromise a computer if such drivers are used after support ends. This option permits restricting APFS drivers to current macOS versions.

   - `0` — require the default supported release date of APFS in OpenCore. The default release date will increase with time and thus this setting is recommended. Currently set to 2021/01/01.
   - `-1` — permit any release date to load (strongly discouraged).
   - Other — use custom minimal APFS release date, e.g. `20200401` for 2020/04/01. APFS release dates can be found in OpenCore boot log and `OcApfsLib`.

6. `MinVersion`
   Type: `plist integer`