OpenCore

Reference Manual (0.8.8.9)

[2023.01.31]

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. The ProperTree editor is a lightweight, cross-platform and open-source alternative.

It is strongly recommended to avoid configuration creation tools that are aware of the internal configuration structure as this may result in invalid configurations (since the structure gets constantly updated). If such tools are to be used despite this warning, ensure that only stable versions of OpenCore explicitly supported by such tools are used. In such cases, the use of open-source implementations with transparent binary generation (such as OCAT) is encouraged, given that other tools may contain malware. In addition, configurations created for a specific hardware setup should never be used on different hardware setups.

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.

When updating the LaTeX documentation (e.g. `Configuration.tex`) please do *not* rebuild the PDF files till merging to master happens. This avoids unnecessary merge conflicts:

- External contributors using the pull-request approach should request the maintainers to handle the PDF rebuild in the pull-request message.
- Internal contributors should rebuild the documentation at merge time in the same or in a separate commit. One can ask another maintainer to rebuild the documentation when lacking the necessary tools in the pull-request message.

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
rm -rf OpenCorePkg
git clone --depth=1 https://github.com/acidanthera/OpenCorePkg
. ./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
```

loaded and connected first. Configuring the boot chime and adding this longer additional delay can also be useful in systems where fast boot time and/or slow monitor signal synchronisation may cause the boot logo not to be shown at all on some boots or reboots.

12. `Timeout`
    Type: `plist integer`, 32 bit
    Failsafe: `0`
    Description: Timeout in seconds in the OpenCore picker before automatic booting of the default boot entry. Set to `0` to disable.

13. `PickerMode`
    Type: `plist string`
    Failsafe: `Builtin`
    Description: Choose picker used for boot management.

    `PickerMode` describes the underlying boot management with an optional user interface responsible for handling boot options.

    The following values are supported:

    - `Builtin` — boot management is handled by OpenCore, a simple text-only user interface is used.
    - `External` — an external boot management protocol is used if available. Otherwise, the `Builtin` mode is used.
    - `Apple` — Apple boot management is used if available. Otherwise, the `Builtin` mode is used.

    Upon success, the `External` mode may entirely disable all boot management in OpenCore except for policy enforcement. In the `Apple` mode, it may additionally bypass policy enforcement. Refer to the OpenCanopy plugin for an example of a custom user interface.

    The OpenCore built-in picker contains a set of actions chosen during the boot process. The list of supported actions is similar to Apple BDS and typically can be accessed by holding `action hotkeys` during the boot process.

    The following actions are currently considered:

    - `Default` — this is the default option, and it lets the built-in OpenCore picker load the default boot option as specified in the Startup Disk preference pane.
    - `ShowPicker` — this option forces the OpenCore picker to be displayed. This can typically be achieved by holding the `OPT` key during boot. Setting `ShowPicker` to `true` will make `ShowPicker` the default option.
    - `BootApple` — this options performs booting to the first Apple operating system found unless the chosen default operating system is one from Apple. Hold the `X` key down to choose this option.
    - `BootAppleRecovery` — this option performs booting into the Apple operating system recovery partition. This is either that related to the default chosen operating system, or first one found when the chosen default operating system is not from Apple or does not have a recovery partition. Hold the `CMD+R` hotkey combination down to choose this option.

    Note 1: On non-Apple firmware `KeySupport`, `OpenUsbKbDxe`, or similar drivers are required for key handling. However, not all of the key handling functions can be implemented on several types of firmware.

    Note 2: In addition to `OPT`, OpenCore supports using both the `Escape` and `Zero` keys to enter the OpenCore picker when `ShowPicker` is disabled. `Escape` exists to support co-existence with the Apple picker (including OpenCore `Apple` picker mode) and to support firmware that fails to report held `OPT` key, as on some PS/2 keyboards. In addition, `Zero` is provided to support systems on which `Escape` is already assigned to some other pre-boot firmware feature. In systems which do not require `KeySupport`, pressing and holding one of these keys from after power on until the picker appears should always be successful. The same should apply when using `KeySupport` mode if it is correctly configured for the system, i.e. with a long enough `KeyForgetThreshold`. If pressing and holding the key is not successful to reliably enter the picker, multiple repeated keypresses may be tried instead.

    Note 3: On Macs with problematic GOP, it may be difficult to re-bless OpenCore if its bless status is lost. The `BootKicker` utility can be used to work around this problem, if set up as a Tool in OpenCore ~~(e.g. on a CDROM)~~ with `FullNvramAccess` enabled. It will launch the Apple picker, which allows selection of an item to boot next (with `Enter`), or next and ~~subsequently, i.e. as the blessed entry~~ from then on until the next change

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`. 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.
`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.
`OpenNtfsDxe*`	New Technologies File System (NTFS) read-only driver. NTFS is the primary file system for Microsoft Windows versions that are based on Windows NT.
`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.

Description: Attempt to detach USB controller ownership from the firmware driver. While most types of firmware manage to do this properly, or at least have an option for this, some do not. As a result, the operating system may freeze upon boot. Not recommended unless specifically required.

10. `ReloadOptionRoms`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Query PCI devices and reload their Option ROMs if available.

    For example, this option allows reloading NVIDIA GOP Option ROM on older Macs after the firmware version is upgraded via `ForgeUefiSupport`.

11. `RequestBootVarRouting`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Request redirect of all `Boot` prefixed variables from `EFI_GLOBAL_VARIABLE_GUID` to `OC_VENDOR_VARIABLE_GUID`.

    This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `OpenRuntime.efi`. The quirk lets default boot entry preservation at times when the firmware deletes incompatible boot entries. In summary, this quirk is required to reliably use the Startup Disk preference pane in firmware that is not compatible with macOS boot entries by design.

    By redirecting `Boot` prefixed variables to a separate GUID namespace with the help of `RequestBootVarRouting` quirk we achieve multiple goals:

    - Operating systems are jailed and only controlled by OpenCore boot environment to enhance security.
    - Operating systems do not mess with OpenCore boot priority, and guarantee fluent updates and hibernation wakes for cases that require reboots with OpenCore in the middle.
    - Potentially incompatible boot entries, such as macOS entries, are not deleted or corrupted in any way.

12. `ResizeUsePciRbIo`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Use PciRootBridgeIo for `ResizeGpuBars` and `ResizeAppleGpuBars`

    The quirk makes `ResizeGpuBars` and `ResizeAppleGpuBars` use `PciRootBridgeIo` instead of PciIo. This is needed on systems with a buggy `PciIo` implementation where trying to configure Resizable BAR results in `Capability I/O Error`. Typically this is required on older systems which have been modified with ReBarUEFI.

13. `ResizeGpuBars`
    Type: `plist integer`
    Failsafe: `-1`
    Description: Configure GPU PCI BAR sizes.

    This quirk sets GPU PCI BAR sizes as specified or chooses the largest available below the `ResizeGpuBars` value. The specified value follows PCI Resizable BAR spec. Use `0` for 1 MB, `1` for 2 MB, `2` for 4 MB, and so on up to `19` for 512 GB.

    Resizable BAR technology allows to ease PCI device programming by mapping a configurable memory region, BAR, into CPU address space (e.g. VRAM to RAM) as opposed to a fixed memory region. This technology is necessary, because one cannot map the largest memory region by default, for the reasons of backwards compatibility with older hardware not supporting 64-bit BARs. Consequentially devices of the last decade use BARs up to 256 MB by default (4 remaining bits are used by other data) but generally allow resizing them to both smaller and larger powers of two (e.g. from 1 MB up to VRAM size).

    Operating systems targeting x86 platforms generally do not control PCI address space, letting UEFI firmware decide on the BAR addresses and sizes. This illicit practice resulted in Resizable BAR technology being unused up until 2020 despite being standardised in 2008 and becoming widely available in the hardware soon after.

    Modern UEFI firmware allow the use of Resizable BAR technology but generally restrict the configurable options to failsafe default (`OFF`) and maximum available (`ON`). This quirk allows to fine-tune this value for testing and development purposes.

Consider a GPU with 2 BARs:

- `BAR0` supports sizes from 256 MB to 8 GB. Its value is 4 GB.
- `BAR1` supports sizes from 2 MB to 256 MB. Its value is 256 MB.

Example 1: Setting `ResizeGpuBars` to 1 GB will change `BAR0` to 1 GB and leave `BAR1` unchanged.
Example 2: Setting `ResizeGpuBars` to 1 MB will change `BAR0` to 256 MB and `BAR0` to 2 MB.
Example 3: Setting `ResizeGpuBars` to 16 GB will change `BAR0` to 8 GB and leave `BAR1` unchanged.

Note 1: This quirk shall not be used to workaround macOS limitation to address BARs over 1 GB. `ResizeAppleGpuBars` should be used instead.

Note 2: While this quirk can increase GPU PCI BAR sizes, this will not work on most firmware as is, because the quirk does not relocate BARs in memory, and they will likely overlap. ~~Contributions to improve this feature are welcome~~In most cases it is best to either update the firmware to the latest version or customise it with a specialised driver like ReBarUEFI.

14. `TscSyncTimeout`
Type: `plist integer`
Failsafe: `0`
Description: Attempts to perform TSC synchronisation with a specified timeout.

The primary purpose of this quirk is to enable early bootstrap TSC synchronisation on some server and laptop models when running a debug XNU kernel. For the debug kernel the TSC needs to be kept in sync across the cores before any kext could kick in rendering all other solutions problematic. The timeout is specified in microseconds and depends on the amount of cores present on the platform, the recommended starting value is `500000`.

This is an experimental quirk, which should only be used for the aforementioned problem. In all other cases, the quirk may render the operating system unstable and is not recommended. The recommended solution in the other cases is to install a kernel extension such as VoodooTSCSync, TSCAdjustReset, or CpuTscSync (a more specialised variant of VoodooTSCSync for newer laptops).

Note: This quirk cannot replace the kernel extension because it cannot operate in ACPI S3 (sleep wake) mode and because the UEFI firmware only provides very limited multicore support which prevents precise updates of the MSR registers.

15. `UnblockFsConnect`
Type: `plist boolean`
Failsafe: `false`
Description: Some types of firmware block partition handles by opening them in `By Driver` mode, resulting in an inability to install File System protocols.

Note: This quirk is useful in cases where unsuccessful drive detection results in an absence of boot entries.

11.19 ReservedMemory Properties

1. `Address`
Type: `plist integer`
Failsafe: `0`
Description: Start address of the reserved memory region, which should be allocated as reserved effectively marking the memory of this type inaccessible to the operating system.

The addresses written here must be part of the memory map, have a `EfiConventionalMemory` type, and be page-aligned (4 KBs).

Note: Some types of firmware may not allocate memory areas used by S3 (sleep) and S4 (hibernation) code unless CSM is enabled causing wake failures. After comparing the memory maps with CSM disabled and enabled, these areas can be found in the lower memory and can be fixed up by doing the reservation. Refer to the `Sample.plist` file for details.

2. `Comment`
Type: `plist string`
Failsafe: Empty