OpenCore

Reference Manual (0.7.4.5)

[2021.11.01]

it is not necessarily compatible with the target board. This quirk typically frees between 64 and 256 megabytes of memory, present in the debug log, and on some platforms, is the only way to boot macOS, which otherwise fails with allocation errors at the bootloader stage.

This option is useful on all types of firmware, except for some very old ones such as Sandy Bridge. On certain firmware, a list of addresses that need virtual addresses for proper NVRAM and hibernation functionality may be required. Use the `MmioWhitelist` section for this.

4. `DisableSingleUser`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Disable single user mode.

   This is a security option that restricts the activation of single user mode by ignoring the `CMD+S` hotkey and the `-s` boot argument. The behaviour with this quirk enabled is supposed to match T2-based model behaviour. Refer to this archived article to understand how to use single user mode with this quirk enabled.

5. `DisableVariableWrite`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Protect from macOS NVRAM write access.

   This is a security option that restricts NVRAM access in macOS. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `OpenRuntime.efi`.

   Note: This quirk can also be used as an ad hoc workaround for defective UEFI runtime services implementations that are unable to write variables to NVRAM and results in operating system failures.

6. `DiscardHibernateMap`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reuse original hibernate memory map.

   This option forces the XNU kernel to ignore a newly supplied memory map and assume that it did not change after waking from hibernation. This behaviour is required by Windows to work. Windows mandates preserving runtime memory size and location after S4 wake.

   Note: This may be used to workaround defective memory map implementations on older, rare legacy hardware. Examples of such hardware are Ivy Bridge laptops with Insyde firmware such as the Acer V3-571G. Do not use this option without a full understanding of the implications.

7. `EnableSafeModeSlide`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Patch bootloader to have KASLR enabled in safe mode.

   This option is relevant to users with issues booting to safe mode (e.g. by holding `shift` or with using the `-x` boot argument). By default, safe mode forces `0` slide as if the system was launched with the `slide=0` boot argument.

   - This quirk attempts to patch the `boot.efi` file to remove this limitation and to allow using other values (from `1` to `255` inclusive).
   - This quirk requires enabling `ProvideCustomSlide`.

   Note: The need for this option is dependent on the availability of safe mode. It can be enabled when booting to safe mode fails.

8. `EnableWriteUnprotector`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Permit write access to UEFI runtime services code.

   This option bypasses RWX permissions in code pages of UEFI runtime services by removing write protection (`WP`) bit from `CR0` register during their execution. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `OpenRuntime.efi`.

14. `ProvideCustomSlide`
Type: `plist boolean`
Failsafe: `false`
Description: Provide custom KASLR slide on low memory.

This option performs memory map analysis of the firmware and checks whether all slides (from `1` to `255`) can be used. As `boot.efi` generates this value randomly with `rdrand` or pseudo randomly `rdtsc`, there is a chance of boot failure when it chooses a conflicting slide. In cases where potential conflicts exist, this option forces macOS to select a pseudo random value from the available values. This also ensures that the `slide=` argument is never passed to the operating system (for security reasons).

Note: The need for this quirk is determined by the `OCABC: Only N/256 slide values are usable!` message in the debug log.

15. `ProvideMaxSlide`
Type: `plist integer`
Failsafe: `0`
Description: Provide maximum KASLR slide when higher ones are unavailable.

This option overrides the maximum slide of 255 by a user specified value between 1 and 254 (inclusive) when `ProvideCustomSlide` is enabled. It is assumed that modern firmware allocates pool memory from top to bottom, effectively resulting in free memory when slide scanning is used later as temporary memory during kernel loading. When such memory is not available, this option stops the evaluation of higher slides.

Note: The need for this quirk is determined by random boot failures when `ProvideCustomSlide` is enabled and the randomized slide falls into the unavailable range. When `AppleDebug` is enabled, the debug log typically contains messages such as `AAPL: [EB|'LD:LKC] } Err(0x9)`. To find the optimal value, append `slide=X`, where `X` is the slide value, to the `boot-args` and select the largest one that does not result in boot failures.

16. `RebuildAppleMemoryMap`
Type: `plist boolean`
Failsafe: `false`
Description: Generate macOS compatible Memory Map.

The Apple kernel has several limitations on parsing the UEFI memory map:

- The Memory map size must not exceed 4096 bytes as the Apple kernel maps it as a single 4K page. As some types of firmware can have very large memory maps, potentially over 100 entries, the Apple kernel will crash on boot.
- The Memory attributes table is ignored. `EfiRuntimeServicesCode` memory statically gets `RX` permissions while all other memory types get `RW` permissions. As some firmware drivers may write to global variables at runtime, the Apple kernel will crash at calling UEFI runtime services unless the driver `.data` section has a `EfiRuntimeServicesData` type.

To workaround these limitations, this quirk applies memory attribute table permissions to the memory map passed to the Apple kernel and optionally attempts to unify contiguous slots of similar types if the resulting memory map exceeds 4 KB.

Note 1: Since several types of firmware come with incorrect memory protection tables, this quirk often comes paired with `SyncRuntimePermissions`.

Note 2: The need for this quirk is determined by early boot failures. This quirk replaces `EnableWriteUnprotector` on firmware supporting Memory Attribute Tables (MAT). This quirk is typically unnecessary when using `OpenDuetPkg` but may be required to boot macOS 10.6, and earlier, for reasons that are as yet unclear.

17. `ResizeAppleGpuBars`
Type: `plist integer`
Failsafe: `-1`
Description: Reduce GPU PCI BAR sizes for compatibility with macOS.

This quirk reduces GPU PCI BAR sizes for Apple macOS up to the specified value or lower if it is unsupported. 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. Apple macOS supports 1 GB maximum, which is `10`. Use `-1` to disable this quirk.

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 `ResizeAppleGpuBars` to 1 GB will change `BAR0` to 1 GB and leave `BAR1` unchanged.
Example 2: Setting `ResizeAppleGpuBars` to 1 MB will change `BAR0` to 256 MB and `BAR0` to 2 MB.
Example 3: Setting `ResizeAppleGpuBars` to 16 GB will make no changes.

Note 1: See `ResizeGpuBars` quirk for general GPU PCI BAR size configuration and more details about the technology.

Note 2: Certain GPU drivers do not support non-standard BAR sizes, causing sleep wake issues, for this reason for macOS it is recommended to use minimal supported BAR sizes, i.e. specify `0` (1 MB).

18. `SetupVirtualMap`
Type: `plist boolean`
Failsafe: `false`
Description: Setup virtual memory at `SetVirtualAddresses`.

Some types of firmware access memory by virtual addresses after a `SetVirtualAddresses` call, resulting in early boot crashes. This quirk workarounds the problem by performing early boot identity mapping of assigned virtual addresses to physical memory.

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

19. `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.

20. `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~~ note: includes halt at black screen ~~),~~ ~~particularly in~~ as well as more obvious crash). Particularly likely to affect early boot of ~~the Linux kernel~~Windows or Linux (but not always both) on affected systems. Only firmware released after 2017 is typically affected.

Note: This option should be avoided whenever possible. Modern firmware typically have compatible AHCI controllers.

11. `ForceSecureBootScheme`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 11
    Description: Force `x86` scheme for IMG4 verification.

    Note: This option is required on virtual machines when using `SecureBootModel` different from `x86legacy`.

12. `IncreasePciBarSize`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.10
    Description: ~~Increases 32-bit PCI bar size in IOPCIFamily from 1 to 4 GBs~~Allows IOPCIFamily to boot with 2 GB PCI BARs.

    Normally macOS restricts PCI BARs to 1 GB. Enabling this option (still) does not let macOS actually use PCI devices with larger BARs.

    Note: This option should be avoided whenever possible. A need for this option indicates misconfigured or defective firmware.

13. `LapicKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.6 (64-bit)
    Description: Disables kernel panic on LAPIC interrupts.

14. `LegacyCommpage`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.4 - 10.6
    Description: Replaces the default 64-bit commpage bcopy implementation with one that does not require SSSE3, useful for legacy platforms. This prevents a `commpage no match for last` panic due to no available 64-bit bcopy functions that do not require SSSE3.

15. `PanicNoKextDump`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.13 (not required for older)
    Description: Prevent kernel from printing kext dump in the panic log preventing from observing panic details. Affects 10.13 and above.

16. `PowerTimeoutKernelPanic`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.15 (not required for older)
    Description: Disables kernel panic on setPowerState timeout.

    An additional security measure was added to macOS Catalina (10.15) causing kernel panic on power change timeout for Apple drivers. Sometimes it may cause issues on misconfigured hardware, notably digital audio, which sometimes fails to wake up. For debug kernels `setpowerstate_panic=0` boot argument should be used, which is otherwise equivalent to this quirk.

17. `ProvideCurrentCpuInfo`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.8
    Description: Provides current CPU info to the kernel.

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`~~ 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~~ aims to automatically detect and boot most Linux distros without additional configuration.

Usage is as follows:

- Add `OpenLinuxBoot.efi` and also typically (see below) `ext4_x64.efi` to the `config.plist` `Drivers` section.
- Make sure `RequestBootVarRouting` and `LauncherOption` are enabled in ~~`{ESP}/loader/entries/*`~~`config`~~`.conf`~~`plist`~~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~~ ; it is also recommended to enable ~~`/boot`~~`HideAuxiliary` ~~applies to most Fedora-related distros including Fedora itself, RHEL and variants. BootLoaderSpecByDefault includes the possibility of expanding GRUB variables in its `*.conf` files~~ in order to hide older Linux kernels except when required (they are added as auxiliary entries and so may then be shown by pressing the `Spacebar` key in the OpenCore boot menu).
- Install Linux as normal if this has not been done earlier ~~– 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.~~ OpenLinuxBoot is not involved in this stage.
- Reboot into OpenCore: the installed Linux distribution should just appear and boot directly from OpenCore when selected, which it does without chainloading via GRUB.

~~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.~~ If OpenCore has already been manually set up to boot Linux, e.g. via `BlessOverride` or via `Entries` then then these settings may be removed so that the Linux distribution is not displayed twice in the boot menu.

~~The method of starting the kernel relies on it being compiled with EFISTUB, however this applies to almost all modern distros, particularly those which usesystemd. Most modern distros use systemd as their system manager (even though at the same time most do *not*~~ It is recommended to install Linux with its default bootloader, even though this will not be actively used when booting via OpenLinuxBoot. This is because OpenLinuxBoot has to detect the correct kernel options to 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~~, and does so by looking in files left by the default bootloader. If no bootloader was installed (or these options cannot be found) booting is still possible, but the correct boot options must be manually specified before OpenLinuxBoot will attempt to start the distro.

~~*Note 1:* `OpenLinuxBoot`~~ OpenLinuxBoot typically requires filesystem drivers that ~~may not be~~ are not 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~~ not tested for

reliability in all scenarious, nor ~~underwent any~~ did they undergo tamper-resistance testing, therefore ~~have~~ they may carry potential security or data-loss risks.

Most Linux ~~distributions keep their boot files on an EXT4 partition even when the distribution's root filesystem is something else, such as BTRFS, therefore only an EXT4 driver such as~~ distros require the `ext4_x64` ~~is normally required. A BTRFS driversuch as~~ driver, a few may require the `btrfs_x64` ~~will be required in the currently somewhat less standard 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*~~driver, and a few may require no additional file system driver: ~~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~~it depends on the filesystem of the boot partition of the installed distro, and on what filesystems are already supported by the system's firmware. LVM is not currently supported - this is because it is not believed that there is currently a stand-alone UEFI LVM filesystem driver.

~~*Note 4:*~~ Be aware of the `SyncRuntimePermissions` quirk, which may need to be set to avoid early boot failure (~~i.e. halts with~~ typically halting with a black screen) of the Linux kernel, due to a firmware bug of some firmware released after 2017. When present and not mitigated by this quirk, this affects booting via OpenCore with or without OpenLinuxBoot.

After installing OpenLinuxBoot, it is recommended to compare the Linux boot options (shown with `cat /proc/cmdline`) seen when booting via OpenLinuxBoot and via the distro's original bootloader. If the default bootloader is GRUB, expect the options generated by OpenLinuxBoot not to contain a `BOOT_IMAGE=...` value where the GRUB options do, and to contain an `initrd=...` value while the GRUB options do not. All remaining options should match (option order does not matter) – perhaps excluding less important graphics handover options (such as in the Ubuntu example given in `autoopts` below). If they do not, it is recommended to manually add the missing options, e.g. with `partuuidopts:{partuuid}+={opts}` to target a specific distro (or just with `autoopts+={opts}`, which applies to all installed distros, if only one distro is in use).

If using OpenLinuxBoot with Secure Boot, users may wish to use the `shim-to-cert.tool` included in OpenCore utilities, which can be used to extract the required public key to validate a distro's kernels directly, rather than via shim. For non-GRUB distros, the required public key must be found by user research.

11.6.1 Configuration

The default parameter values should work well with no changes under most circumstances, but if ~~you need to parameterise this driver~~ required the following options for the driver may be specified in `UEFI/Drivers/Arguments`:

- `flags` - Default: all flags except `LINUX_BOOT_ADD_DEBUG_INFO` and `LINUX_BOOT_LOG_VERBOSE` 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`~~ 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`~~ OpenLinuxBoot flags and `Misc/Security/ScanPolicy` in order to be sure to detect all valid Linux installs, since Linux boot filesystems are very often marked as `LINUX_DATA`.

- `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`~~ 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.
- `0x00002000` (bit `13`) — `LINUX_BOOT_ALLOW_CONF_AUTO_ROOT`, In some instances of `BootLoaderSpecByDefault` in combination with `ostree`, the `/loader/entries/*.conf` files do not specify a required `root=...` kernel option – it is added by GRUB. If this bit is set and this situation is detected, then automatically add this option. (Required for example by Endless OS.)
- `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~~ not the same for Fedora-style distros booted from `/loader/entries/*.conf` files.

  Typically ~~you should not need this option~~ this option should not be needed in the latter case, but in case ~~you do~~it is, 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~~ it is possible to examine how ~~your drives~~ the distro's partitions are mounted using the Linux `mount` command, and then find out the partuuid of relevant mounted ~~drives~~ partitions 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~~ the distro's default 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~~

11.6.2 Additional information

OpenLinuxBoot can detect the `loader/entries/*.conf` files created according to the Boot Loader Specification or the closely related systemd BootLoaderSpecByDefault. The former is specific to systemd-boot and is used by Arch Linux, the latter applies to most Fedora-related distros including Fedora itself, RHEL and variants.

Where the above files are not present, OpenLinuxBoot can autodetect and boot `{boot}/vmlinuz*` kernel files directly. It links these automatically – ~~perhaps excluding less important graphics handover options, such as in the Ubuntu example given in~~ based on the kernel version in the filename – to their associated `{boot}/init*` ramdisk files. This applies to most Debian-related distros, including Debian itself, Ubuntu and variants.

When autodetecting, OpenLinuxBoot looks 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,~~ `/etc/default/grub` for kernel boot options and `/etc/os-release` for the distro name.

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 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 ~~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.~~ boot menu.

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.

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.

9. `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`.

10. `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.

11. `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.