# OpenCore

Reference Manual (0.9.3.4)

[2023.07.04]

# 8 Misc

## 8.1 Introduction

This section contains miscellaneous configuration options affecting OpenCore operating system loading behaviour in addition to other options that do not readily fit into other sections.

OpenCore broadly follows the "bless" model, also known as the "Apple Boot Policy". The primary purpose of the "bless" model is to allow embedding boot options within the file system (and be accessible through a specialised driver) as well as supporting a broader range of predefined boot paths as compared to the removable media list set out in the UEFI specification.

Partitions can only booted by OpenCore when they meet the requirements of a predefined `Scan policy`. This policy sets out which specific file systems a partition must have, and which specific device types a partition must be located on, to be made available by OpenCore as a boot option. Refer to the `ScanPolicy` property for details.

The scan process starts with enumerating all available partitions, filtered based on the `Scan policy`. Each partition may generate multiple primary and alternate options. Primary options represent operating systems installed on the media, while alternate options represent recovery options for the operating systems on the media.

- Alternate options may exist without primary options and vice versa.
- Options may not necessarily represent operating systems on the same partition.
- Each primary and alternate option can be an auxiliary option or not.
  - Refer to the `HideAuxiliary` section for details.

### 8.1.1 Boot Algorithm

The algorithm to determine boot options behaves as follows:

1. Obtain all available partition handles filtered based on the `Scan policy` (and driver availability).
2. Obtain all available boot options from the `BootOrder` UEFI variable.
3. For each boot option found:
   - Retrieve the device path of the boot option.
   - Perform fixups (e.g. NVMe subtype correction) and expansion (e.g. for Boot Camp) of the device path.
   - On failure, if it is an OpenCore custom entry device path, pre-construct the corresponding custom entry and succeed.
   - Obtain the device handle by locating the device path of the resulting device path (ignore it on failure).
   - Locate the device handle in the list of partition handles (ignore it if missing).
   - For disk device paths (not specifying a bootloader), execute "bless" (may return > 1 entry).
   - For file device paths, check for presence on the file system directly.
   - Exclude entries if there is a `.contentVisibility` file ~~near the bootloader or inside the boot directory~~ at a relevant location (see below) with `Disabled` contents (ASCII) (and if the current `InstanceIentifier` matches an `Instance-List` if present, see below).
   - Mark device handle as *used* in the list of partition handles if any.
   - Register the resulting entries as primary options and determine their types.
     The option will become auxiliary for some types (e.g. Apple HFS recovery) or if its `.contentVisibility` file contains `Auxiliary` (and if the current `InstanceIentifier` matches an `Instance-List` if present, see below).
4. For each partition handle:
   - If the partition handle is marked as *unused*, execute "bless" primary option list retrieval.
     In case a `BlessOverride` list is set, both standard and custom "bless" paths will be found.
   - On the OpenCore boot partition, exclude OpenCore bootstrap files using header checks.
   - Register the resulting entries as primary options and determine their types if found.
     The option will become auxiliary for some types (e.g. Apple HFS recovery).
   - If a partition already has any primary options of the "Apple Recovery" type, proceed to the next handle.
   - Lookup alternate entries by "bless" recovery option list retrieval and predefined paths.
   - Register the resulting entries as alternate auxiliary options and determine their types if found.
5. Custom entries and tools, except such pre-constructed previously, are added as primary options without any checks with respect to `Auxiliary`.
6. System entries, such as `Reset NVRAM`, are added as primary auxiliary options.

A `.contentVisibility` file may be placed next to the bootloader (such as `boot.efi`), or in the boot folder (for DMG folder based boot items). Example locations, as seen from within macOS, are:

- `/System/Volumes/Preboot/{GUID}/System/Library/CoreServices/.contentVisibility`
- `/Volumes/{ESP}/EFI/BOOT/.contentVisibility`

In addition a `.contentVisibility` file may be placed in the instance-specific (for macOS) or absolute root folders related to a boot entry, for example:

- `/System/Volumes/Preboot/{GUID}/.contentVisibility`
- `/System/Volumes/Preboot/.contentVisibility`
- `/Volumes/{ESP}/.contentVisibility` (*not recommended*)

These root folder locations are supported specifically for macOS, because non-Apple files next to the Apple bootloader are removed by macOS updates. It is supported but not recommended to place a `.contentVisibility` file in a non-macOS root location (such as the last location shown above), because it will hide all entries on the drive.

The `.contentVisibility` file, when present, may optionally target only specific instances of OpenCore. Its contents are `[{Instance-List}:](Disabled|Auxiliary)`. If a colon (`:`) is present, the preceding `Instance-List` it is a comma separated list of `InstanceIdentifier` values (example: `OCA,OCB:Disabled`). When this list is present, the specified visibility is only applied if the `InstanceIdentifier` of the current instance of OpenCore is present in the list. When the list is not present, the specified visibility is applied for all instances of OpenCore.

*Note 1*: For any instance of OpenCore with no `InstanceIdentifier` value, the specified visibility from a `.contentVisibility` file with an `Instance-List` will never be applied.

*Note 2*: Visibilities with a visibility list will be treated as invalid, and so ignored, in earlier versions of OpenCore - which may be useful when comparing behaviour of older and newer versions.

*Note 3*: Avoid extraneous spaces in the `.contentVisibility` file: these will not be treated as whitespace, but as part of the adjacent token.

The display order of the boot options in the OpenCore picker and the boot process are determined separately from the scanning algorithm.

The display order is as follows:

- Alternate options follow corresponding primary options. That is, Apple recovery options will follow the relevant macOS option whenever possible.
- Options will be listed in file system handle firmware order to maintain an established order across reboots regardless of the operating system chosen for loading.
- Custom entries, tools, and system entries will be added after all other options.
- Auxiliary options will only be displayed upon entering "Extended Mode" in the OpenCore picker (typically by pressing the `Space` key).

The boot process is as follows:

- Look up the first valid primary option in the `BootNext` UEFI variable.
- On failure, look up the first valid primary option in the `BootOrder` UEFI variable.
- Mark the option as the default option to boot.
- Boot option through the picker or without it depending on the `ShowPicker` option.
- Show picker on failure otherwise.

*Note 1*: This process will only work reliably when the `RequestBootVarRouting` option is enabled or the firmware does not control UEFI boot options (`OpenDuetPkg` or custom BDS). When `LauncherOption` is not enabled, other operating systems may overwrite OpenCore settings and this property should therefore be enabled when planning to use other operating systems.

*Note 2*: UEFI variable boot options boot arguments will be removed, if present, as they may contain arguments that can compromise the operating system, which is undesirable when secure boot is enabled.

*Note 3*: Some operating systems, such as Windows, may create a boot option and mark it as the topmost option upon first boot or after NVRAM resets from within OpenCore. When this happens, the default boot entry choice will remain changed until the next manual reconfiguration.

- Should only be used on systems with reliable hibernation wake in macOS, otherwise users may not be able to visually see boot loops that may occur.
- Highly recommended to pair this option with `PollAppleHotKeys`, allows to enter picker in case of issues with hibernation wake.
- Visual indication for hibernation wake is currently out of scope.

4. `HideAuxiliary`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Set to `true` to hide auxiliary entries from the picker menu.

   An entry is considered auxiliary when at least one of the following applies:

   - Entry is macOS recovery.
   - Entry is macOS Time Machine.
   - Entry is explicitly marked as `Auxiliary`.
   - Entry is system (e.g. `Reset NVRAM`).

   To display all entries, the picker menu can be reloaded into "Extended Mode" by pressing the `Spacebar` key. Hiding auxiliary entries may increase boot performance on multi-disk systems.

5. `InstanceIdentifier`
   **Type**: `plist string`
   **Failsafe**: Empty
   **Description**: An optional identifier for the current instance of OpenCore.

   This should typically be a short alphanumeric string. The current use of this value is to optionally target `.contentVisibility` files to specific instances of OpenCore, as explained in the Boot Algorithm section.

6. `LauncherOption`
   **Type**: `plist string`
   **Failsafe**: `Disabled`
   **Description**: Register the launcher option in the firmware preferences for persistence.

   Valid values:

   - `Disabled` — do nothing.
   - `Full` — create or update the top priority boot option in UEFI variable storage at bootloader startup.
     - For this option to work, `RequestBootVarRouting` is required to be enabled.
   - `Short` — create a short boot option instead of a complete one.
     - This variant is useful for some older types of firmware, typically from Insyde, that are unable to manage full device paths.
   - `System` — create no boot option but assume specified custom option is blessed.
     - This variant is useful when relying on `ForceBooterSignature` quirk and OpenCore launcher path management happens through `bless` utilities without involving OpenCore.

   This option allows integration with third-party operating system installation and upgrades (which may overwrite the `\EFI\BOOT\BOOTx64.efi` file). The BOOTx64.efi file is no longer used for bootstrapping OpenCore if a custom option is created. The custom path used for bootstrapping can be specified by using the `LauncherPath` option.

   *Note 1*: Some types of firmware may have NVRAM implementation flaws, no boot option support, or other incompatibilities. While unlikely, the use of this option may result in boot failures and should only be used exclusively on boards known to be compatible. Refer to acidanthera/bugtracker#1222 for some known issues affecting Haswell and other boards.

   *Note 2*: While NVRAM resets executed from OpenCore would not typically erase the boot option created in `Bootstrap`, executing NVRAM resets prior to loading OpenCore will erase the boot option. Therefore, for significant implementation updates, such as was the case with OpenCore 0.6.4, an NVRAM reset should be executed with `Bootstrap` disabled, after which it can be re-enabled.

   *Note 3*: Some versions of Intel Visual BIOS (e.g. on Intel NUC) have an unfortunate bug whereby if any boot option is added referring to a path on a USB drive, from then on that is the only boot option which will be shown