



OpenCore

Reference Manual (0.5.~~0~~.1)

[2019.09.18]

8 Misc

8.1 Introduction

This section contains miscellaneous configuration entries for OpenCore behaviour that does not go to any other sections

8.2 Properties

1. Boot

Type: plist dict

Description: Apply boot configuration described in Boot Properties section below.

2. BlessOverride

Type: plist array

Description: Add custom scanning paths through bless model.

Designed to be filled with `plist string` entries containing absolute UEFI paths to customised bootloaders, for example, `\EFI\Microsoft\bootmgfw.efi` for Microsoft bootloader. This allows unusual boot paths to be automatically discovered by the boot picker. Designwise they are equivalent to predefined blessed path, such as `\System\Library\CoreServices\boot.efi`, but unlike predefined bless paths they have highest priority.

3. Debug

Type: plist dict

Description: Apply debug configuration described in Debug Properties section below.

4. Entries

Type: plist array

Description: Add boot entries to boot picker.

Designed to be filled with `plist dict` values, describing each load entry. See Entry Properties section below.

5. Security

Type: plist dict


Description: Apply security configuration described in Security Properties section below.

6. Tools

Type: plist array

Description: Add tool entries to boot picker.

Designed to be filled with `plist dict` values, describing each load entry. See Entry Properties section below.

Note: Select tools, for example, UEFI Shell  are very dangerous and **MUST NOT** appear in production configurations, especially in vaulted ones and protected with secure boot, as they may be used to easily bypass secure boot chain.

8.3 Boot Properties

1. ConsoleMode

Type: plist string

Failsafe: Empty string

Description: Sets console output mode as specified with the WxH (e.g. 80x24) formatted string. Set to empty string not to change console mode. Set to `Max` to try to use largest available console mode.

Note: This field is best to be left empty on most firmwares.

2. ConsoleBehaviourOs

Type: plist string

Failsafe: Empty string

Description: Set console control behaviour upon operating system load.

Console control is a legacy protocol used for switching between text and graphics screen output. Some firmwares do not provide it, yet select operating systems require its presence, which is what `ConsoleControl` UEFI protocol is for.

- 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.
- 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.

Note: Given the above description, 0xF0103 value is expected to allow scanning of SATA, SAS, SCSI, and NVMe devices with APFS file system, and prevent scanning of any devices with HFS or FAT32 file systems in addition to not scanning APFS file systems on USB, CD, USB, and FireWire drives. The combination reads as:

- OC_SCAN_FILE_SYSTEM_LOCK
- OC_SCAN_DEVICE_LOCK
- OC_SCAN_ALLOW_FS_APFS
- OC_SCAN_ALLOW_DEVICE_SATA
- OC_SCAN_ALLOW_DEVICE_SASEX
- OC_SCAN_ALLOW_DEVICE_SCSI
- OC_SCAN_ALLOW_DEVICE_NVME

8.6 Entry Properties

1. Comment

Type: plist string

Failsafe: Empty string

Description: Arbitrary ASCII string used to provide human readable reference for the entry. It is implementation defined whether this value is used.

2. Enabled

Type: plist boolean

Failsafe: false

Description: This entry will not be listed unless set to **true**.

3. Name

Type: plist string

Failsafe: Empty string

Description: Human readable entry name displayed in boot picker.

4. Path

Type: plist string

Failsafe: Empty string

Description: Entry location depending on entry type.

- **Entries** specify external boot options, and therefore take device paths in **Path** key. These values are not checked, thus be extremely careful. Example: `PciRoot(0x0)/Pci(0x1,0x1)/.../EFI\COOL.EFI`
- **Tools** specify internal boot options, which are part of bootloader vault, and therefore take file paths relative to OC/Tools directory. Example: `CleanNvramShell.efi`.

Partition table holds up to 128 entries
Main partition table begins at sector 2 and ends at sector 33
First usable sector is 34, last usable sector is 419430366
Partitions will be aligned on 2048-sector boundaries
Total free space is 4029 sectors (2.0 MiB)

| Number | Start (sector) | End (sector) | Size | Code | Name |
|--------|----------------|--------------|-----------|------|------------------------|
| 1 | 2048 | 1023999 | 499.0 MiB | 2700 | Basic data partition |
| 2 | 1024000 | 1226751 | 99.0 MiB | EF00 | EFI system partition |
| 3 | 1226752 | 1259519 | 16.0 MiB | 0C01 | Microsoft reserved ... |
| 4 | 1259520 | 419428351 | 199.4 GiB | 0700 | Basic data partition |

Command (? for help): c
Partition number (1-4): 4
Enter name: BOOTCAMP

Command (? for help): w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING PARTITIONS!!

Do you want to proceed? (Y/N): Y
OK; writing new GUID partition table (GPT) to \\.\physicaldrive0.
Disk synchronization succeeded! The computer should now use the new partition table.
The operation has completed successfully.

Listing 3: Relabeling Windows volume

How to choose Windows BOOTCAMP with custom NTFS drivers?

Third-party drivers providing NTFS support, such as NTFS-3G, Paragon NTFS, Tuxera NTFS or Seagate Paragon Driver break certain macOS functionality, including Startup Disk preference pane normally used for operating system selection. While the recommended option remains not to use such drivers as they commonly corrupt the filesystem, and prefer the driver bundled with macOS with optional write support (command or GUI), there still exist vendor-specific workarounds for their products: Tuxera, Paragon, etc.

12.2 Debugging

Similar to other projects working with hardware OpenCore supports auditing and debugging. The use of NOOPT or DEBUG build modes instead of RELEASE can produce a lot more debug output. With NOOPT source level debugging with GDB or IDA Pro is also available. For GDB check OcSupport Debug page. For IDA Pro you will need IDA Pro 7.3 or newer, refer to Debugging the XNU Kernel with IDA Pro for more details.

To obtain the log during boot you can make the use of serial port debugging. Serial port debugging is enabled in Target, e.g. 0xB for onscreen with serial. OpenCore uses 115200 baud rate, 8 data bits, no parity, and 1 stop bit. For macOS your best choice are CP2102-based UART devices. Connect motherboard TX to USB UART ~~GND~~RX, and motherboard GND to USB UART ~~RX~~GND. Use screen utility to get the output, or download GUI software, such as CoolTerm.

Note: On several motherboards (and possibly USB UART dongles) PIN naming may be incorrect. It is very common to have GND swapped with RX, thus you have to connect motherboard “TX” to USB UART GND, and motherboard “GND” to USB UART RX.

Remember to enable COM port in firmware settings, and never use USB cables longer than 1 meter to avoid output corruption. To additionally enable XNU kernel serial output you will need debug=0x8 boot argument.

12.3 Tips and Tricks

1. How to debug boot failure?

Normally it is enough to obtain the actual error message. For this ensure that: