

OpenCore

Reference Manual (0.0~~3~~.4)

[2019.07.09]

4 ACPI

4.1 Introduction

ACPI (Advanced Configuration and Power Interface) is an open standard to discover and configure computer hardware. ACPI specification defines the standard tables (e.g. DSDT, SSDT, FACS, DMAR) and various methods (e.g. _DSM, _PWRPRW) for implementation. Modern hardware needs little changes to maintain ACPI compatibility, yet some of those are provided as a part of OpenCore.

To compile and disassemble ACPI tables iASL compiler can be used developed by ACPICA. GUI front-end to iASL compiler can be downloaded from Acidanthera/MaciASL.

4.2 Properties

1. Add

Type: plist array

Failsafe: Empty

Description: Load selected tables from OC/ACPI directory.

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

2. Block

Type: plist array

Failsafe: Empty

Description: Remove selected tables from ACPI stack.

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

3. Patch

Type: plist array

Failsafe: Empty

Description: Perform binary patches in ACPI tables before table addition or removal.

Designed to be filled with `plist dictionary` values describing each patch entry. See Patch Properties section below.

4. Quirks

Type: plist dict

Description: Apply individual ACPI quirks described in Quirks Properties section below.

4.3 Add 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 ACPI table will not be added unless set to `true`.

3. Path

Type: plist string

Failsafe: Empty string

Description: File paths meant to be loaded as ACPI tables. Example values include `DSDT.aml`, `SubDir/SSDT-8.aml`, `SSDT-USBX.aml`, etc.

ACPI table load order follows the item order in the array. All ACPI tables load from OC/ACPI directory.

Note: All tables but tables with DSDT table identifier (determined by parsing data not by filename) insert new tables into ACPI stack. DSDT, unlike the rest, performs replacement of DSDT table.

6. Mask

Type: plist data

Failsafe: Empty data

Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Can be set to empty data to be ignored. Must equal to **Replace** in size otherwise.
7. OemTableId

Type: plist data, 8 bytes

Failsafe: All zero

Description: Match table OEM ID to be equal to this value unless all zero.
8. Replace

Type: plist data

Failsafe: Empty data

Description: Replacement data of one or more bytes.
9. ReplaceMask

Type: plist data

Failsafe: Empty data

Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Can be set to empty data to be ignored. Must equal to **Replace** in size otherwise.
10. Skip

Type: plist integer

Failsafe: 0

Description: Number of found occurrences to be skipped before replacement is done.
11. TableLength

Type: plist integer

Failsafe: 0

Description: Match table size to be equal to this value unless 0.
12. TableSignature

Type: textttplist data, 4 bytes

Failsafe: All zero

Description: Match table signature to be equal to this value unless all zero.

In the majority of the cases ACPI patches are not useful and harmful:

- Avoid renaming devices with ACPI patches. This may fail or perform improper renaming of unrelated devices (e.g. EC and EC0), be unnecessary, or even fail to rename devices in select tables. For ACPI consistency it is much safer to rename devices at I/O Registry level, as done by WhateverGreen.
- Avoid patching `_OSI` to support a higher level of feature sets unless absolutely required. Commonly this enables a number of hacks on APTIO firmwares, which result in the need to add more patches. Modern firmwares generally do not need it at all, and those that do are fine with much smaller patches.
- Try to avoid hacky changes like renaming `_PWRPRW` or `_DSM` whenever possible.

Several cases, where patching actually does make sense, include:

- Refreshing HPET (or another device) method header to avoid compatibility checks by `_OSI` on legacy hardware. `_STA` method with `if ((OSFL () == Zero)) { If (HPTE) ... Return (Zero)` content may be forced to always return 0xF by replacing `A0 10 93 4F 53 46 4C 00` with `A4 0A 0F A3 A3 A3 A3`.
- To provide custom method implementation with in an SSDT, for instance, to report functional key presses on a laptop, the original method can be replaced with a dummy name by patching `_Q11` with `XQ11`.

Tianocore AcpiAml.h source file may help understanding ACPI opcodes.

4.6 Quirks Properties

1. FadtEnableReset

Type: plist boolean

11 Troubleshooting

11.1 Windows support

Can I install Windows?

While no official Windows support is provided, 64-bit UEFI Windows installations (Windows 8 and above) prepared with Boot Camp are supposed to work. Third-party UEFI installations as well as systems partially supporting UEFI boot, like Windows 7, might work with some extra precautions. Things to keep in mind:

- MBR (Master Boot Record) installations are legacy and will not be supported.
- Installing Windows and macOS on the same drive is currently unsupported but will be addressed later.
- All the modifications applied (to ACPI, NVRAM, SMBIOS, etc.) are supposed to be operating system agnostic, i.e. apply equally regardless of the OS booted. This enables Boot Camp software experience on Windows.
- macOS requires the first partition to be EFI System Partition, and does not support the default Windows layout. While OpenCore does have a workaround for this, it is highly recommend not to rely on it and install properly.
- Windows may need to be reactivated. To avoid it consider leaving SystemUUID field empty, so that the original firmware UUID is used. Be warned, on old firmwares it may be invalid, i.e. not random. In case you still have issues, consider using HWID or KMS38 license. The nuances of Windows activation are out of the scope of this document and can be found online.

What additional software do I need?

To enable operating system switching and install relevant drivers in the majority of cases you will need Windows support software from Boot Camp. For simplicity of the download process or when configuring an already installed Windows version a third-party utility, Brigadier, can be used successfully. Note, that you may have to download and install 7-Zip prior to using Brigadier.

Remember to always use the latest version of Windows support software from Boot Camp, as versions prior to 6.1 do not support APFS, and thus will not function correctly. To download newest software pass most recent Mac model to Brigadier, for example `./brigadier.exe -m iMac19,1`. To install Boot Camp on an unsupported Mac model afterwards run PowerShell as Administrator and enter `msiexec /i BootCamp.msi`. In case you already have a previous version of Boot Camp installed you will have to remove it first by running `msiexec /x BootCamp.msi` command. `BootCamp.msi` file is located in `BootCamp/Drivers/Apple` directory and can be reached through Windows Explorer.

While Windows support software from Boot Camp solves most of compatibility problems, sometimes you may have to address some of them manually:

- To invert mouse wheel scroll direction `FlipFlopWheel` must be set to 1 as explained on SuperUser.
- `RealTimeIsUniversal` must be set to 1 to avoid time desync between Windows and macOS as explained on SuperUser (this one is usually not needed).
- To access Apple filesystems like HFS and APFS separate software may need to be installed. Some of the known tools are: Apple HFS+ driver (hack for Windows 10), HFSExplorer, MacDrive, Paragon APFS, Paragon HFS+, TransMac, etc. Remember to never ever attempt to modify Apple file systems from Windows as this often leads to irrecoverable data loss.

Why do I see Basic data partition in Boot Camp ~~Control~~ Startup Disk control panel?

Boot Camp control panel uses GPT partition table to obtain each boot option name. After installing Windows separately you will have to relabel the partition manually. This can be done with many tools including open-source `gdisk` utility. Reference example:

```
PS C:\gdisk> .\gdisk64.exe \\.\\physicaldrive0
GPT fdisk (gdisk) version 1.0.4
```

```
Command (? for help): p
Disk \\.\\physicaldrive0: 419430400 sectors, 200.0 GiB
Sector size (logical): 512 bytes
Disk identifier (GUID): DEC57EB1-B3B5-49B2-95F5-3B8C4D3E4E12
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, or Tuxera NTFS 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), there still exist vendor-specific workarounds for their products: Tuxera, Paragon, etc.

11.2 Tips and Tricks

1. How to debug boot failure?

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

- You have a DEBUG or NOOPT version of OpenCore.
- Logging is enabled (1) and shown onscreen (2): Misc → Debug → Target = 3.
- Logged messages from at least DEBUG_ERROR (0x80000000), DEBUG_WARN (0x00000002), and DEBUG_INFO (0x00000040) levels are visible onscreen: Misc → Debug → DisplayLevel = 0x80000042.
- Critical error messages, like DEBUG_ERROR, stop booting: Misc → Security → HaltLevel = 0x80000000.
- Watch Dog is disabled to prevent automatic reboot: Uefi → Quirks → DisableWatchDog = true.
- Boot Picker (entry selector) is enabled: Misc → Boot → ShowPicker = true.

If there is no obvious error, check the available hacks in Quirks sections one by one.

2. How to customise boot entries?

OpenCore follows standard Apple Bless model and extracts the entry name from `.contentDetails` and `.disk_label.contentDetails` files in the booter directory if present. These files contain an ASCII string with an entry title, which may then be customised by the user.

3. What is the simplest way to install macOS?

Copy online recovery image (*.dmg and *.chunklist files) to `com.apple.recovery.boot` directory on a FAT32 partition with OpenCore. Load OpenCore Boot Picker and choose the entry, it will have a (dmg) suffix. Custom name may be created by providing `.contentDetails` file.

To download recovery online you may use Recovery tool from OcSupportPkg.