



OpenCore

Reference Manual (0.9.~~1~~.2)

[2023.05.06]

4 ACPI

4.1 Introduction

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

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

ACPI changes apply globally (to every operating system) with the following effective order:

- **Delete** is processed.
- **Quirks** are processed.
- **Patch** is processed.
- **Add** is processed.

*Note: **RebaseRegions** and **SyncTableIds** quirks are special and are processed after all other ACPI changes since they can only be applied on the final ACPI configuration including all the patches and added tables.*

Applying the changes globally resolves the problems of incorrect operating system detection (consistent with the ACPI specification, not possible before the operating system boots), operating system chainloading, and difficult ACPI debugging. Hence, more attention may be required when writing changes to _OSI.

Applying the patches early makes it possible to write so called “proxy” patches, where the original method is patched in the original table and is implemented in the patched table.

There are several sources of ACPI tables and workarounds. Commonly used ACPI tables are provided with OpenCore, VirtualSMC, VoodooPS2, and WhateverGreen releases. Besides those, several third-party instructions may be found on the AppleLife Laboratory and DSDT subforums (e.g. Battery register splitting guide). A slightly more user-friendly explanation of some tables included with OpenCore can also be found in Dortania’s Getting started with ACPI guide. For more exotic cases, there are several alternatives such as daliansky’s ACPI sample collection (English Translation by 5T33Z0 et al). Please note however, that suggested solutions from third parties may be outdated or may contain errors.

4.2 Properties

1. Add

Type: plist array

Failsafe: Empty

Description: Load selected tables from the OC/ACPI directory.

To be filled with **plist dict** values, describing each add entry. Refer to the Add Properties section below for details.

2. Delete

Type: plist array

Failsafe: Empty

Description: Remove selected tables from the ACPI stack.

To be filled with **plist dict** values, describing each delete entry. Refer to the Delete Properties section below for details.

3. Patch

Type: plist array

Failsafe: Empty

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

To be filled with **plist dictionary** values describing each patch entry. Refer to the Patch Properties section below for details.

3. DevirtualiseMmio

Type: plist boolean

Failsafe: false

Description: Remove runtime attribute from certain MMIO regions.

This quirk reduces the stolen memory footprint in the memory map by removing the runtime bit for known memory regions. This quirk may result in an increase of KASLR slides available but without additional measures, 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.

[Note: When Apple Secure Boot is enabled single user mode is always disabled.](#)

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.

7 Kernel

7.1 Introduction

This section allows the application of different kinds of kernelspace modifications on Apple Kernel (XNU). The modifications currently provide driver (kext) injection, kernel and driver patching, and driver blocking.

Kernel and kext changes apply with the following effective order:

- Block is processed.
- Add and Force are processed.
- Emulate and Quirks are processed.
- Patch is processed.
- ~~Add and Force are processed.~~

7.2 Properties

1. Add

Type: plist array

Failsafe: Empty

Description: Load selected kernel extensions (kexts) from the `OC/Kexts` directory.

To be filled with `plist dict` values, describing each kext. Refer to the Add Properties section below for details.

Note 1: The load order is based on the order in which the kexts appear in the array. Hence, dependencies must appear before kexts that depend on them.

Note 2: To track the dependency order, inspect the `OSBundleLibraries` key in the `Info.plist` file of the kext being added. Any kext included under the key is a dependency that must appear before the kext being added.

Note 3: Kexts may have inner kexts (`Plugins`) included in the bundle. Such `Plugins` must be added separately and follow the same global ordering rules as other kexts.

2. Block

Type: plist array

Failsafe: Empty

Description: Remove selected kernel extensions (kexts) from the prelinked kernel.

To be filled with `plist dictionary` values, describing each blocked kext. Refer to the Block Properties section below for details.

3. Emulate

Type: plist dict

Description: Emulate certain hardware in kernelspace via parameters described in the Emulate Properties section below.

4. Force

Type: plist array

Failsafe: Empty

Description: Load kernel extensions (kexts) from the system volume if they are not cached.

To be filled with `plist dict` values, describing each kext. Refer to the Force Properties section below for details. This section resolves the problem of injecting kexts that depend on other kexts, which are not otherwise cached. The issue typically affects older operating systems, where various dependency kexts, such as `IOAudioFamily` or `IONetworkingFamily` may not be present in the kernel cache by default.

Note 1: The load order is based on the order in which the kexts appear in the array. Hence, dependencies must appear before kexts that depend on them.

Note 2: **Force** happens before **Add**.

Note 3: The signature of the “forced” kext is not checked in any way. This makes using this feature extremely dangerous and undesirable for secure boot.

Note 4: This feature may not work on encrypted partitions in newer operating systems.

Failsafe: 0 (Search entire text or kernel)
Description: Maximum number of bytes to search for.

9. Mask

Type: plist data
Failsafe: Empty (Ignored)
Description: Data bitwise mask used during find comparison. Allows fuzzy search by ignoring not masked (set to zero) bits. Must be equal to **Replace** in size if set.

10. MaxKernel

Type: plist string
Failsafe: Empty
Description: Patches data on specified macOS version or older.
Note: Refer to the **Add MaxKernel** description for matching logic.

11. MinKernel

Type: plist string
Failsafe: Empty
Description: Patches data on specified macOS version or newer.
Note: Refer to the **Add MaxKernel** description for matching logic.

12. Replace

Type: plist data
Failsafe: Empty
Description: Replacement data of one or more bytes.

13. ReplaceMask

Type: plist data
Failsafe: Empty (Ignored)
Description: Data bitwise mask used during replacement. Allows fuzzy replacement by updating masked (set to non-zero) bits. Must be equal to **Replace** in size if set.

14. Skip

Type: plist integer
Failsafe: 0 (Do not skip any occurrences)
Description: Number of found occurrences to skip before replacements are applied.

7.8 Quirks Properties

1. AppleCpuPmCfgLock

Type: plist boolean
Failsafe: false
Requirement: 10.4-12
Description: Disables `PKG_CST_CONFIG_CONTROL` (0xE2) MSR modification in `AppleIntelCPUPowerManagement.kext`, commonly causing early kernel panic, when it is locked from writing.

Note: `AppleIntelCPUPowerManagement.kext` is removed as of macOS 13. However, a legacy version can be injected and thus get patched using this quirk.

Some types of firmware lock the `PKG_CST_CONFIG_CONTROL` MSR register and the bundled `ControlMsrE2` tool can be used to check its state. Note that some types of firmware only have this register locked on some cores. As modern firmware provide a `CFG Lock` setting that allows configuring the `PKG_CST_CONFIG_CONTROL` MSR register lock, this option should be avoided whenever possible.

On APTIO firmware that do not provide a `CFG Lock` setting in the GUI, it is possible to access the option directly:

- Download UEFITool and IFR-Extractor.
- Open the firmware image in UEFITool and find `CFG Lock` unicode string. If it is not present, the firmware may not have this option and the process should therefore be discontinued.
- Extract the `Setup.bin` PE32 Image Section (the UEFITool found) through the `Extract Body` menu option.
- Run IFR-Extractor on the extracted file (e.g. `./ifrextract Setup.bin Setup.txt`).

Note 3: For this patch to be correctly applied, **Override** must be enabled with all keys properly set in **Custom**, under section **Misc->Serial**.

Note 4: This patch is for PMIO support and is therefore not applied if **UseMmio** under section **Misc->Serial->Custom** is false. For MMIO, there are boot arguments **pcie_mmio_uart=ADDRESS** and **mmio_uart=ADDRESS** that allow the kernel to use MMIO for serial port access.

Note 5: The serial baud rate must be correctly set in both **BaudRate** under section **Misc->Serial->Custom** and via **serialbaud=VALUE** boot argument, both of which should match against each other. The default baud rate is 115200.

6. **CustomSMBIOSGuid**

Type: plist boolean

Failsafe: false

Requirement: 10.4

Description: Performs GUID patching for **UpdateSMBIOSMode Custom** mode. Usually relevant for Dell laptops.

7. **DisableIoMapper**

Type: plist boolean

Failsafe: false

Requirement: 10.8 (not required for older)

Description: Disables **IoMapper** support in XNU (VT-d), which may conflict with the firmware implementation.

Note 1: This option is a preferred alternative to deleting **DMAR** ACPI table and disabling VT-d in firmware preferences, which does not obstruct VT-d support in other systems in case they need this.

Note 2: Misconfigured IOMMU in the firmware may result in broken devices such as ethernet or Wi-Fi adapters. For instance, an ethernet adapter may cycle in link-up link-down state infinitely and a Wi-Fi adapter may fail to discover networks. Gigabyte is one of the most common OEMs with these issues.

8. **DisableIoMapperMapping**

Type: plist boolean

Failsafe: false

Requirement: 13.3 (not required for older)

Description: Disables mapping PCI bridge device memory in IOMMU (VT-d).

Note 1: This option resolves compatibility issues with Wi-Fi, Ethernet and Thunderbolt devices when **AppleVTD** is enabled on systems where the native **DMAR** table contains one or more **Reserved Memory Regions** and **iGPU** is enabled and more than 16 GB memory is installed.

Note 2: This option is not needed on AMD systems.

9. **DisableLinkeditJettison**

Type: plist boolean

Failsafe: false

Requirement: 11

Description: Disables **__LINKEDIT** jettison code.

This option lets **Lilu.kext**, and possibly other kexts, function in macOS Big Sur at their best performance levels without requiring the **keepsyms=1** boot argument.

10. **DisableRtcChecksum**

Type: plist boolean

Failsafe: false

Requirement: 10.4

Description: Disables primary checksum (0x58-0x59) writing in **AppleRTC**.

Note 1: This option will not protect other areas from being overwritten, see **RTCMemoryFixup** kernel extension if this is desired.

Note 2: This option will not protect areas from being overwritten at firmware stage (e.g. macOS bootloader), see **AppleRtcRam** protocol description if this is desired.

11. **ExtendBTFeatureFlags**

Type: plist boolean

This option may be useful for keyboard layouts with `Option` key situated to the right of `Command` key.

6. `PointerSupport`

Type: plist boolean

Failsafe: false

Description: Enable internal pointer driver.

This option implements standard UEFI pointer protocol (`EFI_SIMPLE_POINTER_PROTOCOL`) through certain OEM protocols. The option may be useful on Z87 ASUS boards, where `EFI_SIMPLE_POINTER_PROTOCOL` is defective.

7. `PointerSupportMode`

Type: plist string

Failsafe: Empty

Description: Set OEM protocol used for internal pointer driver.

Currently the only supported variant is ASUS, using specialised protocol available on certain Z87 and Z97 ASUS boards. More details can be found in [LongSoft/UefiTool#116](#). The value of this property cannot be empty if `PointerSupport` is enabled.

8. `TimerResolution`

Type: plist integer

Failsafe: 0

Description: Set architecture timer resolution.

This option allows updating the firmware architecture timer period with the specified value in 100 nanosecond units. Setting a lower value typically improves performance and responsiveness of the interface and input handling.

The recommended value is 50000 (5 milliseconds) or slightly higher. Select ASUS Z87 boards use 60000 for the interface. Apple boards use 100000. In case of issues, this option can be left as 0 to not change the timer resolution.

11.16 Output Properties

1. `InitialMode`

Type: plist string

Failsafe: Auto

Description: Selects the internal `ConsoleControl` mode in which `TextRenderer` will operate.

Available values are `Auto`, `Text` and `Graphics`. `Text` and `Graphics` specify the named mode. `Auto` uses the current mode of the system `ConsoleControl` protocol when one exists, defaulting to `Text` mode otherwise.

UEFI firmware typically supports `ConsoleControl` with two rendering modes: `Graphics` and `Text`. Some types of firmware do not provide a native `ConsoleControl` and rendering modes. OpenCore and macOS expect text to only be shown in `Text` mode but graphics to be drawn in any mode, and this is how the OpenCore `Builtin` renderer behaves. Since this is not required by the UEFI specification, behaviour of the system `ConsoleControl` protocol, when it exists, may vary.

2. `TextRenderer`

Type: plist string

Failsafe: `BuiltinGraphics`

Description: Chooses renderer for text going through standard console output.

Currently two renderers are supported: `Builtin` and `System`. The `System` renderer uses firmware services for text rendering—, however with additional options provided to sanitize the output. The `Builtin` bypasses renderer bypasses firmware services and performs text rendering on its own. Different renderers support Each renderer supports a different set of options. It is recommended to use the `Builtin` renderer, as it supports HiDPI mode and uses full screen resolution.

UEFI firmware typically supports Each renderer provides its own `ConsoleControl` with two rendering modes: `Graphics` and `Text`. Some types of firmware do not support protocol (in the case of `ConsoleControl` and rendering modes. OpenCore and macOS expect text to only be shown in `Graphics` mode and graphics to be drawn in any mode. Since this is not required by UEFI specification, exact behaviour variesSystem only, this passes some operations through to the system `ConsoleControl` protocol, if one exists).

Valid values [of this option](#) are combinations of ~~text-renderer and rendering mode~~ [: the renderer to use and the ConsoleControl mode to set on the underlying system ConsoleControl protocol before starting. To control the initial mode of the provided ConsoleControl protocol once started, use the InitialMode option.](#)

- `BuiltinGraphics` — Switch to Graphics mode ~~and then~~ [use Builtin renderer with custom ConsoleControl.](#)
- `BuiltinText` — Switch to Text mode ~~and then~~ [use Builtin renderer with custom ConsoleControl.](#)
- `SystemGraphics` — Switch to Graphics mode ~~and then~~ [use System renderer with custom ConsoleControl.](#)
- `SystemText` — Switch to Text mode ~~and then~~ [use System renderer with custom ConsoleControl.](#)
- `SystemGeneric` — Use System renderer with [custom a ConsoleControl protocol which passes its mode set and get operations through to system ConsoleControl assuming it behaves correctly when it exists.](#)

The use of `BuiltinGraphics` is straightforward. For most platforms, it is necessary to enable `ProvideConsoleGop` and set `Resolution` to `Max`. The `BuiltinText` variant is an alternative [to](#) `BuiltinGraphics` for some very old and defective laptop firmware, which can only draw in `Text` mode.

The use of `System` protocols is more complicated. Typically, the preferred setting is `SystemGraphics` or `SystemText`. Enabling `ProvideConsoleGop`, setting `Resolution` to `Max`, enabling `ReplaceTabWithSpace` is useful on almost all platforms. `SanitiseClearScreen`, `IgnoreTextInGraphics`, and `ClearScreenOnModeSwitch` are more specific, and their use depends on the firmware.

Note: Some Macs, such as the `MacPro5,1`, may have incompatible console output when using modern GPUs, and thus only `BuiltinGraphics` may work for them in such cases. NVIDIA GPUs may require additional firmware upgrades.

3. `ConsoleMode`

Type: `plist string`

Failsafe: Empty (Maintain current console mode)

Description: Sets console output mode as specified with the `WxH` (e.g. `80x24`) formatted string.

Set to `Max` to attempt using the largest available console mode. ~~This option is currently ignored as the Builtin text-renderer only supports one console mode.~~

Note: This field is best left empty on most types of firmware.

4. `Resolution`

Type: `plist string`

Failsafe: Empty (Maintain current screen resolution)

Description: Sets console output screen resolution.

- Set to `WxH@Bpp` (e.g. `1920x1080@32`) or `WxH` (e.g. `1920x1080`) formatted string to request custom resolution from GOP if available.
- Set to `Max` to attempt using the largest available screen resolution.

On HiDPI screens `APPLE_VENDOR_VARIABLE_GUID UIScale` NVRAM variable may need to be set to `02` to enable HiDPI scaling in `Builtin` text renderer, FileVault 2 UEFI password interface, and boot screen logo. Refer to the Recommended Variables section for details.

Note: This will fail when console handle has no GOP protocol. When the firmware does not provide it, it can be added with `ProvideConsoleGop` set to `true`.

5. `ForceResolution`

Type: `plist boolean`

Failsafe: `false`

Description: Forces `Resolution` to be set in cases where the desired resolution is not available by default, such as on legacy Intel GMA and first generation Intel HD Graphics (Ironlake/Arrandale). Setting `Resolution` to `Max` will try to pull the largest available resolution from the connected display's EDID.

Note: This option depends on the `OC_FORCE_RESOLUTION_PROTOCOL` protocol being present. This protocol is currently only supported by `OpenDuetPkg`. The `OpenDuetPkg` implementation currently only supports Intel iGPUs.

6. `ClearScreenOnModeSwitch`

Type: `plist boolean`

Failsafe: `false`

Description: Some types of firmware only clear part of the screen when switching from graphics to text mode, leaving a fragment of previously drawn images visible. This option fills the entire graphics screen with black colour before switching to text mode.

Note: This option only applies to **System** renderer.

7. DirectGopRendering

Type: plist boolean

Failsafe: false

Description: Use builtin graphics output protocol renderer for console.

On certain firmware, such as on the **MacPro5,1**, this may provide better performance or fix rendering issues. However, this option is not recommended unless there is an obvious benefit as it may result in issues such as slower scrolling.

This renderer fully supports **AppleEg2Info** protocol and will provide screen rotation for all EFI applications. In order to provide seamless rotation compatibility with **EfiBoot**, builtin **AppleFramebufferInfo** should also be used, i.e. it may need to be overridden on Mac EFI.

8. GopBurstMode

Type: plist boolean

Failsafe: false

Description: Enable write-combining (WC) caching for GOP memory, if system firmware has not already enabled it.

Some older firmware (e.g. EFI-era Macs) fails to set write-combining caching (aka burst mode) for GOP memory, even though the CPU supports it. Setting this can give a considerable speed-up for GOP operations, especially on systems which require **DirectGopRendering**.

Note 1: This [quirk](#) takes effect whether or not **DirectGopRendering** is set, and ~~may give some in some cases~~ [may give a noticeable](#) speed-up to GOP operations even when **DirectGopRendering** is false.

Note 2: [On most systems from circa 2013 onwards, write-combining caching is already applied by the firmware to GOP memory, in which case GopBurstMode is unnecessary. On such systems enabling the quirk should normally be harmless, producing an OCC: debug log entry indicating that burst mode is already started.](#)

Note 3: [Some caution should be taken when enabling this quirk, as it has been observed to cause hangs on a few systems. Since additional guards have been added to try to prevent this, please log a bugtracker issue if such a system is found.](#)

9. GopPassThrough

Type: plist string

Failsafe: Disabled

Description: Provide GOP protocol instances on top of UGA protocol instances.

This option provides the GOP protocol via a UGA-based proxy for firmware that do not implement the protocol. The supported values for the option are as follows:

- **Enabled** — provide GOP for all UGA protocols.
- **Apple** — provide GOP for **AppleFramebufferInfo**-enabled protocols.
- **Disabled** — do not provide GOP.

Note: This option requires **ProvideConsoleGop** to be enabled.

10. IgnoreTextInGraphics

Type: plist boolean

Failsafe: false

Description: Some types of firmware output text onscreen in both graphics and text mode. This is typically unexpected as random text may appear over graphical images and cause UI corruption. Setting this option to true will discard all text output ~~when-if~~ console control is ~~in a different mode from~~ [not in](#) **Text** [mode](#).

Note: This option only applies to the **System** renderer.

11. ReplaceTabWithSpace

Type: plist boolean