# OpenCore

Reference Manual (0.5.4.5)

[2020.02.01]

When console control is available, OpenCore can be made console control aware, and set different modes for the operating system booter (`ConsoleBehaviourOs`), which normally runs in graphics mode, and its own user interface (`ConsoleBehaviourUi`), which normally runs in text mode. Possible behaviours, set as values of these options, include:

- Empty string — Do not modify console control mode.
- `Text` — Switch to text mode.
- `Graphics` — Switch to graphics mode.
- `ForceText` — Switch to text mode and preserve it (requires `ConsoleControl`).
- `ForceGraphics` — Switch to graphics mode and preserve it (require `ConsoleControl`).

Hints:

- Unless empty works, firstly try to set `ConsoleBehaviourOs` to `Graphics` and `ConsoleBehaviourUi` to `Text`.
- On APTIO IV (Haswell and earlier) it is usually enough to have `ConsoleBehaviourOs` set to `Graphics` and `ConsoleBehaviourUi` set to `ForceText` to avoid visual glitches.
- On APTIO V (Broadwell and newer) `ConsoleBehaviourOs` set to `ForceGraphics` and `ConsoleBehaviourUi` set to `ForceText` usually works best.
- On Apple firmwares `ConsoleBehaviourOs` set to `Graphics` and `ConsoleBehaviourUi` set to `Text` is supposed to work best.

*Note*: `IgnoreTextInGraphics` and `SanitiseClearScreen` may need to be enabled for select firmware implementations. Particularly APTIO firmwares.

3. `ConsoleBehaviourUi`
   **Type**: `plist string`
   **Failsafe**: Empty string
   **Description**: Set console control behaviour upon OpenCore user interface load. Refer to `ConsoleBehaviourOs` description for details.

4. `HibernateMode`
   **Type**: `plist string`
   **Failsafe**: `None`
   **Description**: Hibernation detection mode. The following modes are supported:

   - `None` — Avoid hibernation for your own good.
   - `Auto` — Use RTC and NVRAM detection.
   - `RTC` — Use RTC detection.
   - `NVRAM` — Use NVRAM detection.

5. `HideSelf`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Hides own boot entry from boot picker. This may potentially hide other entries, for instance, when another UEFI OS is installed on the same volume and driver boot is used.

6. `PollAppleHotKeys`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable ~~modifier hotkey~~ `modifier hotkey` handling in boot picker.

   In addition to ~~action hotkeys~~`action hotkeys`, which are partially described in `UsePicker` section and are normally handled by Apple BDS, there exist modifier keys, which are handled by operating system bootloader, namely `boot.efi`. These keys allow to change operating system behaviour by providing different boot modes.

   On some firmwares it may be problematic to use modifier keys due to driver incompatibilities. To workaround this problem this option allows registering select hotkeys in a more permissive manner from within boot picker. Such extensions include the support of tapping on keys in addition to holding and pressing `Shift` along with other keys instead of just `Shift` alone, which is not detectible on many PS/2 keyboards. This list of known ~~hotkeys~~ `modifier hotkeys` includes:

   - `CMD+C+MINUS` — disable board compatibility checking.
   - `CMD+K` — boot release kernel, similar to `kcsuffix=release`.

- `CMD+S` — single user mode.
- `CMD+S+MINUS` — disable KASLR slide, requires disabled SIP.
- `CMD+V` — verbose mode.
- `Shift` — safe mode.

7. `Resolution`
   **Type**: `plist string`
   **Failsafe**: Empty string
   **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 empty string not to change screen resolution.
   - Set to `Max` to try to use 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 FileVault 2 UEFI password interface and boot screen logo. Refer to Recommended Variables section for more 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` UEFI quirk set to `true`.

8. `ShowPicker`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Show simple boot picker to allow boot entry selection.

9. `TakeoffDelay`
   **Type**: `plist integer`, 32 bit
   **Failsafe**: `0`
   **Description**: Delay in microseconds performed before handling picker startup and `action hotkeys`.

   Introducing a delay may give extra time to hold the right `action hotkey` sequence to e.g. boot to recovery mode. On some platforms setting this option to at least `5000-10000` microseconds may be necessary to access `action hotkeys` at all due to the nature of the keyboard driver.

10. `Timeout`
    **Type**: `plist integer`, 32 bit
    **Failsafe**: `0`
    **Description**: Timeout in seconds in boot picker before automatic booting of the default boot entry. Use 0 to disable timer.

11. `UsePicker`
    **Type**: `plist boolean`
    **Failsafe**: `false`
    **Description**: Use OpenCore built-in boot picker for boot management.

    `UsePicker` set to `false` entirely disables all boot management in OpenCore except policy enforcement. In this case a custom user interface may utilise OcSupportPkg `OcBootManagementLib` to implement a user friendly boot picker oneself. Reference example of external graphics interface is provided in ExternalUi test driver.

    OpenCore built-in boot picker contains a set of actions chosen during the boot process. The list of supported actions is similar to Apple BDS and ~~currently consists of the following options~~in general can be accessed by holding `action hotkeys` during boot process. Currently the following actions are considered:

    - `Default` — this is the default option, and it lets OpenCore built-in boot picker to loads the default boot option as specified in Startup Disk preference pane.
    - `ShowPicker` — this option forces picker to show. Normally it can be achieved by holding `OPT` key during boot. Setting `ShowPicker` to `true` will make `ShowPicker` the default option.
    - `ResetNvram` — this option performs select UEFI variable erase and is normally achieved by holding `CMD+OPT+P+R` key combination during boot. Another way to erase UEFI variables is to choose `Reset NVRAM` in the picker. This option requires `AllowNvramReset` to be set to `true`.

- `BootApple` — this options performs booting to the first found Apple operating system unless the default chosen operating system is already made by Apple. Hold `X` key to choose this option.
- `BootAppleRecovery` — this option performs booting to Apple operating system recovery. Either the one related to the default chosen operating system, or first found in case default chosen operating system is not made by Apple or has no recovery. Hold `CMD+R` key combination to choose this option.

*Note*: ~~activated~~ Activated `KeySupport`, `AppleUsbKbDxe`, or similar driver is required for key handling to work. On many firmwares it is not possible to get all the keys function.

In addition to `OPT` OpenCore supports `Escape` key `ShowPicker`. This key exists for firmwares with PS/2 keyboards that fail to report held `OPT` key and require continual presses of `Escape` key to enter the boot menu.

## 8.4 Debug Properties

1. `DisableWatchDog`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Select firmwares may not succeed in quickly booting the operating system, especially in debug mode, which results in watch dog timer aborting the process. This option turns off watch dog timer.

2. `DisplayDelay`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Delay in microseconds performed after every printed line visible onscreen (i.e. console).

3. `DisplayLevel`
   **Type**: `plist integer`, 64 bit
   **Failsafe**: `0`
   **Description**: EDK II debug level bitmask (sum) showed onscreen. Unless `Target` enables console (onscreen) printing, onscreen debug output will not be visible. The following levels are supported (discover more in DebugLib.h):

   - `0x00000002` (bit `1`) — `DEBUG_WARN` in `DEBUG`, `NOOPT`, `RELEASE`.
   - `0x00000040` (bit `6`) — `DEBUG_INFO` in `DEBUG`, `NOOPT`.
   - `0x00400000` (bit `22`) — `DEBUG_VERBOSE` in custom builds.
   - `0x80000000` (bit `31`) — `DEBUG_ERROR` in `DEBUG`, `NOOPT`, `RELEASE`.

4. `Target`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: A bitmask (sum) of enabled logging targets. By default all the logging output is hidden, so this option is required to be set when debugging is necessary.

   The following logging targets are supported:

   - `0x01` (bit `0`) — Enable logging, otherwise all log is discarded.
   - `0x02` (bit `1`) — Enable basic console (onscreen) logging.
   - `0x04` (bit `2`) — Enable logging to Data Hub.
   - `0x08` (bit `3`) — Enable serial port logging.
   - `0x10` (bit `4`) — Enable UEFI variable logging.
   - `0x20` (bit `5`) — Enable non-volatile UEFI variable logging.
   - `0x40` (bit `6`) — Enable logging to file.

   Console logging prints less than all the other variants. Depending on the build type (`RELEASE`, `DEBUG`, or `NOOPT`) different amount of logging may be read (from least to most).

   Data Hub log will not log kernel and kext patches. To obtain Data Hub log use the following command in macOS:

```
ioreg -lw0 -p IODeviceTree | grep boot-log | sort | sed 's/.*<\(.*\)>.*/\1/' | xxd -r -p
```

   UEFI variable log does not include some messages and has no performance data. For safety reasons log size is limited to 32 kilobytes. Some firmwares may truncate it much earlier or drop completely if they have no memory.

Using non-volatile flag will write the log to NVRAM flash after every printed line. To obtain UEFI variable log use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-log |
  awk '{gsub(/%0d%0a%00/,"");gsub(/%0d%0a/,"\n")}1'
```

*Warning*: Some firmwares are reported to have broken NVRAM garbage collection. This means that they may not be able to always free space after variable deletion. Do not use non-volatile NVRAM logging without extra need on such devices.

While OpenCore boot log already contains basic version information with build type and date, this data may also be found in NVRAM in `opencore-version` variable even with boot log disabled.

File logging will create a file named `opencore-YYYY-MM-DD-HHMMSS.txt` at EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmwares are not reliable, and may corrupt data when writing files through UEFI. Log is attempted to be written in the safest manner, and thus is very slow. Ensure that `DisableWatchDog` is set to `true` when you use a slow drive.

## 8.5 Security Properties

1. `AllowNvramReset`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Allow `CMD+OPT+P+R` handling and enable showing `NVRAM Reset` entry in boot picker.

2. `AllowSetDefault`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Allow `CTRL+Enter` and `CTRL+Index` handling to set the default boot option in boot picker.

3. `AuthRestart`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable `VirtualSMC`-compatible authenticated restart.

   Authenticated restart is a way to reboot FileVault 2 enabled macOS without entering the password. To perform authenticated restart one can use a dedicated terminal command: `sudo fdesetup authrestart`. It is also used when installing operating system updates.

   VirtualSMC performs authenticated restart by saving disk encryption key split in NVRAM and RTC, which despite being removed as soon as OpenCore starts, may be considered a security risk and thus is optional.

4. `ExposeSensitiveData`
   **Type**: `plist integer`
   **Failsafe**: `0x6`
   **Description**: Sensitive data exposure bitmask (sum) to operating system.

   - `0x01` — Expose printable booter path as an UEFI variable.
   - `0x02` — Expose OpenCore version as an UEFI variable.
   - `0x04` — Expose OpenCore version in boot picker menu title.
   - `0x08` — Expose OEM information as a set of UEFI variables.

   Exposed booter path points to OpenCore.efi or its booter depending on the load order. To obtain booter path use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path
```

   To use booter path for mounting booter volume use the following command in macOS:

```
u=$(nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-path | sed 's/.*GPT,\([^,]*\),.*/\1/'); \
  if [ "$u" != "" ]; then sudo diskutil mount $u ; fi
```

To obtain OpenCore version use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:opencore-version
```

To obtain OEM information use the following commands in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:oem-product # SMBIOS Type1 ProductName
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:oem-vendor  # SMBIOS Type2 Manufacturer
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:oem-board   # SMBIOS Type2 ProductName
```

5. `HaltLevel`
   **Type**: `plist integer`, 64 bit
   **Failsafe**: `0x80000000` (`DEBUG_ERROR`)
   **Description**: EDK II debug level bitmask (sum) causing CPU to halt (stop execution) after obtaining a message of `HaltLevel`. Possible values match `DisplayLevel` values.

6. `RequireSignature`
   **Type**: `plist boolean`
   **Failsafe**: `true`
   **Description**: Require `vault.sig` signature file for `vault.plist` in `OC` directory.

   This file should contain a raw 256 byte RSA-2048 signature from SHA-256 hash of `vault.plist`. The signature is verified against the public key embedded into `OpenCore.efi`.

   To embed the public key you should do either of the following:

   - Provide public key during the `OpenCore.efi` compilation in `OpenCoreVault.c` file.
   - Binary patch `OpenCore.efi` replacing zeroes with the public key between `=BEGIN OC VAULT=` and `==END OC VAULT==` ASCII markers.

   RSA public key 520 byte format description can be found in Chromium OS documentation. To convert public key from X.509 certificate or from PEM file use RsaTool.

   *Note*: `vault.sig` is used regardless of this option when public key is embedded into `OpenCore.efi`. Setting it to `true` will only ensure configuration sanity, and abort the boot process when public key is not set but was supposed to be used for verification.

7. `RequireVault`
   **Type**: `plist boolean`
   **Failsafe**: `true`
   **Description**: Require `vault.plist` file present in `OC` directory.

   This file should contain SHA-256 hashes for all files used by OpenCore. Presence of this file is highly recommended to ensure that unintentional file modifications (including filesystem corruption) do not happen unnoticed. To create this file automatically use `create_vault.sh` script.

   Regardless of the underlying filesystem, path name and case must match between `config.plist` and `vault.plist`.

   *Note*: `vault.plist` is tried to be read regardless of the value of this option, but setting it to `true` will ensure configuration sanity, and abort the boot process.

   The complete set of commands to:

   - Create `vault.plist`.
   - Create a new RSA key (always do this to avoid loading old configuration).
   - Embed RSA key into `OpenCore.efi`.
   - Create `vault.sig`.

   Can look as follows:

```
cd /Volumes/EFI/EFI/OC
/path/to/create_vault.sh .
/path/to/RsaTool -sign vault.plist vault.sig vault.pub
off=$(($(strings -a -t d OpenCore.efi | grep "=BEGIN OC VAULT=" | cut -f1 -d' ')+16))
dd of=OpenCore.efi if=vault.pub bs=1 seek=$off count=528 conv=notrunc
```

- `Overwrite` — Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
- `Custom` — Write first SMBIOS table (`gEfiSmbiosTableGuid`) to `gOcCustomSmbiosTableGuid` to workaround firmwares overwriting SMBIOS contents at ExitBootServices. Otherwise equivalent to `Create`. Requires patching AppleSmbios.kext and AppleACPIPlatform.kext to read from another GUID: `"EB9D2D31"` - `"EB9D2D35"` (in ASCII), done automatically by `CustomSMBIOSGuid` quirk.

6. `Generic`
   **Type**: `plist dictonary`
   **Optional**: When `Automatic` is `false`
   **Description**: Update all fields. This section is read only when `Automatic` is active.

7. `DataHub`
   **Type**: `plist dictonary`
   **Optional**: When `Automatic` is `true`
   **Description**: Update Data Hub fields. This section is read only when `Automatic` is not active.

8. `PlatformNVRAM`
   **Type**: `plist dictonary`
   **Optional**: When `Automatic` is `true`
   **Description**: Update platform NVRAM fields. This section is read only when `Automatic` is not active.

9. `SMBIOS`
   **Type**: `plist dictonary`
   **Optional**: When `Automatic` is `true`
   **Description**: Update SMBIOS fields. This section is read only when `Automatic` is not active.

## 10.2 Generic Properties

1. `SpoofVendor`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Sets SMBIOS vendor fields to `Acidanthera`.

   It is dangerous to use Apple in SMBIOS vendor fields for reasons given in `SystemManufacturer` description. However, certain firmwares may not provide valid values otherwise, which could break some software.

2. ~~`SupportsCsm`~~`AdviseWindows`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Forces ~~CSM~~ Windows support in `FirmwareFeatures`.

   Added bits to `FirmwareFeatures`:

   - `FW_FEATURE_SUPPORTS_CSM_LEGACY_MODE (0x1)` - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being not the first partition on the disk.
   - `FW_FEATURE_SUPPORTS_UEFI_WINDOWS_BOOT (0x20000000)` - Without this bit it is not possible to reboot to Windows installed on a drive with EFI partition being the first partition on the disk.

   ~~*Note*: This was enabled unconditionally in older OpenCore versions.~~

3. `SystemProductName`
   **Type**: `plist string`
   **Failsafe**: `MacPro6,1`
   **Description**: Refer to SMBIOS `SystemProductName`.

4. `SystemSerialNumber`
   **Type**: `plist string`
   **Failsafe**: `OPENCORE_SN1`
   **Description**: Refer to SMBIOS `SystemSerialNumber`.

5. `SystemUUID`
   **Type**: `plist string`, GUID

# 11 UEFI

## 11.1 Introduction

UEFI (Unified Extensible Firmware Interface) is a specification that defines a software interface between an operating system and platform firmware. This section allows to load additional UEFI modules and/or apply tweaks for the onboard firmware. To inspect firmware contents, apply modifications and perform upgrades UEFITool and supplementary utilities can be used.

## 11.2 Properties

1. `ConnectDrivers`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Perform UEFI controller connection after driver loading.

   This option is useful for loading filesystem drivers, which usually follow UEFI driver model, and may not start by themselves. While effective, this option may not be necessary for drivers performing automatic connection, and may slightly slowdown the boot.

   *Note*: Some firmwares, made by Apple in particular, only connect the boot drive to speedup the boot process. Enable this option to be able to see all the boot options when having multiple drives.

2. `Drivers`
   **Type**: `plist array`
   **Failsafe**: None
   **Description**: Load selected drivers from `OC/Drivers` directory.

   Designed to be filled with string filenames meant to be loaded as UEFI drivers. Depending on the firmware a different set of drivers may be required. Loading an incompatible driver may lead your system to unbootable state or even cause permanent firmware damage. Some of the known drivers include:

   - `ApfsDriverLoader` — APFS file system bootstrap driver adding the support of embedded APFS drivers in bootable APFS containers in UEFI firmwares.
   - `FwRuntimeServices` — `OC_FIRMWARE_RUNTIME` protocol implementation that increases the security of OpenCore and Lilu by supporting read-only and write-only NVRAM variables. Some quirks, like `RequestBootVarRouting`, require this driver for proper function. Due to the nature of being a runtime driver, i.e. functioning in parallel with the target operating system, it cannot be implemented within OpenCore itself, but is bundled with OpenCore releases.
   - `EnhancedFatDxe` — FAT filesystem driver from `FatPkg`. This driver is embedded in all UEFI firmwares, and cannot be used from OpenCore. It is known that multiple firmwares have a bug in their FAT support implementation, which leads to corrupted filesystems on write attempt. Embedding this driver within the firmware may be required in case writing to EFI partition is needed during the boot process.
   - `NvmExpressDxe` — NVMe support driver from `MdeModulePkg`. This driver is included in most firmwares starting with Broadwell generation. For Haswell and earlier embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
   - `AppleUsbKbDxe` — USB keyboard driver adding the support of `AppleKeyMapAggregator` protocols on top of a custom USB keyboard driver implementation. This is an alternative to builtin `KeySupport`, which may work better or worse depending on the firmware.
   - `VBoxHfs` — HFS file system driver with bless support. This driver is an alternative to a closed source `HFSPlus` driver commonly found in Apple firmwares. While it is feature complete, it is approximately 3 times slower and is yet to undergo a security audit.
   - `XhciDxe` — XHCI USB controller support driver from `MdeModulePkg`. This driver is included in most firmwares starting with Sandy Bridge generation. For earlier firmwares or legacy systems it may be used to support external USB 3.0 PCI cards.

   To compile the drivers from UDK (EDK II) use the same command you do normally use for OpenCore compilation, but choose a corresponding package:

```
git clone https://github.com/acidanthera/audk UDK
cd UDK
```

```
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p FatPkg/FatPkg.dsc
build -a X64 -b RELEASE -t XCODE5 -p MdeModulePkg/MdeModulePkg.dsc
```

3. `Input`
   **Type**: `plist dict`
   **Failsafe**: None
   **Description**: Apply individual settings designed for input (keyboard and mouse) in Input Properties section below.

4. `Protocols`
   **Type**: `plist dict`
   **Failsafe**: None
   **Description**: Force builtin versions of select protocols described in Protocols Properties section below.

   *Note*: all protocol instances are installed prior to driver loading.

5. `Quirks`
   **Type**: `plist dict`
   **Failsafe**: None
   **Description**: Apply individual firmware quirks described in Quirks Properties section below.

## 11.3 Input Properties

1. `KeyForgetThreshold`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Remove key unless it was submitted during this timeout in milliseconds.

   `AppleKeyMapAggregator` protocol is supposed to contain a fixed length buffer of currently pressed keys. However, the majority of the drivers only report key presses as interrupts and pressing and holding the key on the keyboard results in subsequent submissions of this key with some defined time interval. As a result we use a timeout to remove once pressed keys from the buffer once the timeout expires and no new submission of this key happened.

   This option allows to set this timeout based on your platform. The recommended value that works on the majority of the platforms is `5` milliseconds. For reference, holding one key on VMware will repeat it roughly every `2` milliseconds and the same value for APTIO V is `3-4` milliseconds. Thus it is possible to set a slightly lower value on faster platforms and slightly higher value on slower platforms for more responsive input.

2. `KeyMergeThreshold`
   **Type**: `plist integer`
   **Failsafe**: `0`
   **Description**: Assume simultaneous combination for keys submitted within this timeout in milliseconds.

   Similarly to `KeyForgetThreshold`, this option works around the sequential nature of key submission. To be able to recognise simultaneously pressed keys in the situation when all keys arrive sequentially, we are required to set a timeout within which we assume the keys were pressed together.

   Holding multiple keys results in reports every `2` and `1` milliseconds for VMware and APTIO V respectively. Pressing keys one after the other results in delays of at least `6` and `10` milliseconds for the same platforms. The recommended value for this option is `2` milliseconds, but it may be decreased for faster platforms and increased for slower.

3. `KeySupport`
   **Type**: `plist boolean`
   **Failsafe**: `false`
   **Description**: Enable internal keyboard input translation to `AppleKeyMapAggregator` protocol.

   This option activates the internal keyboard interceptor driver, based on `AppleGenericInput` aka (~~`AptioIntputFix`~~`AptioInput` to fill `AppleKeyMapAggregator` database for input functioning. In case a separate driver is used, such as `AppleUsbKbDxe`, this option should never be enabled.

4. KeySupportMode
   **Type**: plist string
   **Failsafe**: empty string
   **Description**: Set internal keyboard input translation to AppleKeyMapAggregator protocol mode.

   - Auto — Performs automatic choice as available with the following preference: AMI, V2, V1.
   - V1 — Uses UEFI standard legacy input protocol EFI_SIMPLE_TEXT_INPUT_PROTOCOL.
   - V2 — Uses UEFI standard modern input protocol EFI_SIMPLE_TEXT_INPUT_EX_PROTOCOL.
   - AMI — Uses APTIO input protocol AMI_EFIKEYCODE_PROTOCOL.

5. KeySwap
   **Type**: plist boolean
   **Failsafe**: false
   **Description**: Swap Command and Option keys during submission.

   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 select OEM protocols. The option may be useful on Z87 ASUS boards, where EFI_SIMPLE_POINTER_PROTOCOL is broken.

7. PointerSupportMode
   **Type**: plist string
   **Failsafe**: empty string
   **Description**: Set OEM protocol used for internal pointer driver.

   Currently the only supported variant is ASUS, using specialised protocol available on select Z87 and Z97 ASUS boards. More details can be found in LongSoft/UefiTool#116.

8. TimerResolution
   **Type**: plist integer
   **Failsafe**: 0
   **Description**: Set architecture timer resolution.

   This option allows to update firmware architecture timer period with the specified value in 100 nanosecond units. Setting a lower value generally 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. You may leave it as 0 in case there are issues.

## 11.4 Protocols Properties

1. AppleBootPolicy
   **Type**: plist boolean
   **Failsafe**: false
   **Description**: Reinstalls Apple Boot Policy protocol with a builtin version. This may be used to ensure APFS compatibility on VMs or legacy Macs.

   *Note*: Some Macs like MacPro5,1 do have APFS compatibility, but their Apple Boot Policy protocol contains recovery detection issues, thus using this option is advised on them as well.

2. AppleEvent
   **Type**: plist boolean
   **Failsafe**: false
   **Description**: Reinstalls Apple Event protocol with a builtin version. This may be used to ensure File Vault 2 compatibility on VMs or legacy Macs.

3. AppleImageConversion
   **Type**: plist boolean