OpenCore

Reference Manual (0.5.~~7~~.8)

[2020.04.13]

  - `.VolumeIcon.icns` file at volume root for other filesystems.
  - `<TOOL_NAME>.icns` file for `Tools`.

  Volume icons can be set in Finder. Note, that enabling this may result in external and internal icons to be indistinguishable.
- `0x0002` — `OC_ATTR_USE_DISK_LABEL_FILE`, provides custom rendered titles for boot entries:
  - `.disk_label` (`.disk_label_2x`) file near bootloader for all filesystems.
  - `<TOOL_NAME.lbl` (`<TOOL_NAME.l2x`) file near tool for `Tools`.

  Prerendered labels can be generated via `disklabel` utility or `bless` command. When disabled or missing text labels (`.contentDetails` or `.disk_label.contentDetails`) are to be rendered instead.
- `0x0004` — `OC_ATTR_USE_GENERIC_LABEL_IMAGE`, provides predefined label images for boot entries without custom entries. May give less detail for the actual boot entry.
- `0x0008` — `OC_ATTR_USE_ALTERNATE_ICONS`, changes used icon set to an alternate one if it is supported. For example, this could make a use of old-style icons with a custom background colour.

6. `PickerAudioAssist`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enable screen reader by default in boot picker.

   For macOS bootloader screen reader preference is set in `preferences.efires` archive in `isVOEnabled.int32` file and is controlled by the operating system. For OpenCore screen reader support this option is an independent equivalent. Toggling screen reader support in both OpenCore boot picker and macOS bootloader FileVault 2 login window can also be done with `Command` + `F5` key combination.

   Note: screen reader requires working audio support, see `UEFI Audio Properties` section for more details.

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

   In addition to `action hotkeys`, which are partially described in `PickerMode` 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 `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.

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.

9 NVRAM

9.1 Introduction

Has `plist dict` type and allows to set volatile UEFI variables commonly referred as NVRAM variables. Refer to `man nvram` for more details. macOS extensively uses NVRAM variables for OS — Bootloader — Firmware intercommunication, and thus supplying several NVRAM is required for proper macOS functioning.

Each NVRAM variable consists of its name, value, attributes (refer to UEFI specification), and its GUID, representing which 'section' NVRAM variable belongs to. macOS uses several GUIDs, including but not limited to:

- `4D1EDE05-38C7-4A6A-9CC6-4BCCA8B38C14` (`APPLE_VENDOR_VARIABLE_GUID`)
- `7C436110-AB2A-4BBB-A880-FE41995C9F82` (`APPLE_BOOT_VARIABLE_GUID`)
- `8BE4DF61-93CA-11D2-AA0D-00E098032B8C` (`EFI_GLOBAL_VARIABLE_GUID`)
- `4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102` (`OC_VENDOR_VARIABLE_GUID`)

Note: Some of the variables may be added by PlatformNVRAM or Generic subsections of PlatformInfo section. Please ensure that variables of this section never collide with them, as behaviour is undefined otherwise.

For proper macOS functioning it is often required to use `OC_FIRMWARE_RUNTIME` protocol implementation currently offered as a part of `OpenRuntime` driver. While it brings any benefits, there are certain limitations which arise depending on the use.

1. Not all tools may be aware of protected namespaces.
   When `RequestBootVarRouting` is used `Boot`-prefixed variable access is restricted and protected in a separate namespace. To access the original variables tools have to be aware of `OC_FIRMWARE_RUNTIME` logic.
2. Assigned NVRAM variables are not always allowed to exceed 512 bytes.
   This is true for `Boot`-prefixed variables when `RequestBootVarFallback` is used, and for overwriting volatile variables with non-volatile on UEFI 2.8 non-conformant firmwares.

9.2 Properties

1. `Add`
   Type: `plist dict`
   Description: Sets NVRAM variables from a map (`plist dict`) of GUIDs to a map (`plist dict`) of variable names and their values in `plist metadata` format. GUIDs must be provided in canonic string format in upper or lower case (e.g. `8BE4DF61-93CA-11D2-AA0D-00E098032B8C`).

   Created variables get `EFI_VARIABLE_BOOTSERVICE_ACCESS` and `EFI_VARIABLE_RUNTIME_ACCESS` attributes set. Variables will only be set if not present ~~and not blocked. To overwrite a variable add it to~~ or blocked. I.e. to overwrite an existing variable value add the variable name to the `Block` section. This approach enables to provide default values till the operating system takes the lead.

   Note: If `plist key` does not conform to GUID format, behaviour is undefined.

2. `Block`
   Type: `plist dict`
   Description: Removes NVRAM variables from a map (`plist dict`) of GUIDs to an array (`plist array`) of variable names in `plist string` format.

3. `LegacyEnable`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Enables loading of NVRAM variable file named `nvram.plist` from EFI volume root.

   This file must have root `plist dictionary` type and contain two fields:

   - `Version` — `plist integer`, file version, must be set to 1.
   - `Add` — `plist dictionary`, equivalent to `Add` from `config.plist`.

   Variable loading happens prior to `Block` (and `Add`) phases. Unless `LegacyOverwrite` is enabled, it will not overwrite any existing variable. Variables allowed to be set must be specified in `LegacySchema`. Third-party scripts may be used to create `nvram.plist` file. An example of such script can be found in `Utilities`. The use of

- `Overwrite` — Overwrite existing gEfiSmbiosTableGuid and gEfiSmbiosTable3Guid data if it fits new size. Abort with unspecified state otherwise.
- `Custom` — Write ~~first SMBIOS table (~~SMBIOS tables (~~`gEfiSmbiosTableGuid`~~`gEfiSmbios(3)TableGuid`) to ~~`gOcCustomSmbiosTableGuid`~~`gOcCustomSmbios(3)TableGuid` 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.

Note: A side effect of using `Custom` approach is making SMBIOS updates exclusive to macOS, avoiding a collission with existing Windows activation and custom OEM software but potentially breaking Apple-specific tools.

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. `AdviseWindows`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Forces 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.

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`.

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 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 are listed below:

~~APFS file system bootstrap driver adding the support of embedded APFS drivers in bootable APFS containers in UEFI firmwares~~	
AudioDxe	HDA audio support driver in UEFI firmwares for most Intel and some other analog audio controllers. Refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
ExFatDxe	Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmwares. For Sandy Bridge and earlier CPUs `ExFatDxeLegacy` driver should be used due to the lack of `RDRAND` instruction support.
HfsPlus	Proprietary HFS file system driver with bless support commonly found in Apple firmwares. For Sandy Bridge and earlier CPUs `HfsPlusLegacy` driver should be used due to the lack of `RDRAND` instruction support.
HiiDatabase*	HII services support driver from `MdeModulePkg`. This driver is included in most firmwares starting with Ivy Bridge generation. Some applications with the GUI like UEFI Shell may need this driver to work properly.
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.
OpenCanopy*	OpenCore plugin implementing graphical interface.
OpenRuntime*	OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol.
OpenUsbKbDxe*	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.

Driver marked with * are bundled with OpenCore. To compile the drivers from UDK (EDK II) use the same command you 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
```

- `ResetNVRAM` — Reset NVRAM system action or tool.
- `Shell` — Entry with UEFI Shell name (e.g. `OpenShell`).
- `Tool` — Any other tool.

Predefined labels are put to `\EFI\OC\Resources\Label` directory. Each label has `.lbl` or `.l2x` suffix to represent the scaling level. Full list of labels is provided below. All labels are mandatory.

- `EFIBoot` — Generic OS.
- `Apple` — Apple OS.
- `AppleRecv` — Apple Recovery OS.
- `AppleTM` — Apple Time Machine.
- `Windows` — Windows.
- `Other` — Custom entry (see `Entries`).
- `ResetNVRAM` — Reset NVRAM system action or tool.
- `Shell` — Entry with UEFI Shell name (e.g. `OpenShell`).
- `Tool` — Any other tool.

Label and icon generation can be performed with bundled utilities: `disklabel` and `icnspack`. Please refer to sample data for the details about the dimensions.

WARNING: OpenCanopy is currently considered experimental and is not recommended for everyday use. Refer to acidanthera/bugtracker#759 for more details regarding the current limitations.

11.5 OpenRuntime

`OpenRuntime` is an OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol. This protocol implements multiple features required for OpenCore that are otherwise not possible to implement in OpenCore itself as they are needed to work in runtime, i.e. during operating system functioning. Feature highlights:

- NVRAM namespaces, allowing to isolate operating systems from accessing select variables (e.g. `RequestBootVarRouting` or `ProtectSecureBoot`).
- NVRAM proxying, allowing to manipulate multiple variables on variable updates (e.g. `RequestBootVarFallback`).
- Read-only and write-only NVRAM variables, enhancing the security of OpenCore, Lilu, and Lilu plugins, like VirtualSMC, which implements `AuthRestart` support.
- NVRAM isolation, allowing to protect all variables from being written from an untrusted operating system (e.g. `DisableVariableWrite`).
- UEFI Runtime Services memory protection management to workaround read-only mapping (e.g. `EnableWriteUnprotector`).

11.6 Properties

1. APFS
   Type: plist dict
   Failsafe: None
   Description: Provide APFS support as configured in APFS Properties section below.
2. `Audio`
   Type: `plist dict`
   Failsafe: None
   Description: Configure audio backend support described in Audio Properties section below.

   Audio support provides a way for upstream protocols to interact with the selected hardware and audio resources. All audio resources should reside in `\EFI\OC\Resources\Audio` directory. Currently the only supported audio file format is WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

   Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: `[audio type]_[audio localisation]_[audio path].wav`. For unlocalised files filename does not include the language code and looks as follows: `[audio type]_[audio path].wav`.

   - Audio type can be `OCEFIAudio` for OpenCore audio files or `AXEFIAudio` for macOS bootloader audio files.

- Audio localisation is a two letter language code (e.g. `en`) with an exception for Chinese, Spanish, and Portuguese. Refer to `APPLE_VOICE_OVER_LANGUAGE_CODE` definition for the list of all supported localisations.
- Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to `APPLE_VOICE_OVER_AUDIO_FILE` definition. For OpenCore audio paths refer to `OC_VOICE_OVER_AUDIO_FILE` definition. The only exception is OpenCore boot chime file, which is `OCEFIAudio_VoiceOver_Boot.wav`.

Audio localisation is determined separately for macOS bootloader and OpenCore. For macOS bootloader it is set in `preferences.efires` archive in `systemLanguage.utf8` file and is controlled by the operating system. For OpenCore the value of `prev-lang:kbd` variable is used. When native audio localisation of a particular file is missing, English language (`en`) localisation is used. Sample audio files can be found in OcBinaryData repository.

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

   This option is useful for loading drivers following UEFI driver model as they may not start by themselves. Examples of such drivers are filesystem or audio drivers. 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.

4. `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.

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

6. `Output`
   Type: `plist dict`
   Failsafe: None
   Description: Apply individual settings designed for output (text and graphics) in Output Properties section below.

7. `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.

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

11.7 APFS Properties

1. `EnableJumpstart`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load embedded APFS drivers from APFS containers.

   APFS EFI driver is bundled in all bootable APFS containers. This option performs loading of signed APFS drivers with respect to `ScanPolicy`. See more details in "EFI Jumpstart" section of Apple File System Reference.

2. `HideVerbose`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Hide verbose output from APFS driver.

   APFS verbose output can be useful for debugging.

3. `JumpstartHotPlug`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Load APFS drivers for newly connected devices.

   Performs APFS driver loading not only at OpenCore startup but also during boot picker. This permits APFS USB hot plug. Disable if not required.

4. `MinDate`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal allowed APFS driver date.

   APFS driver date connects APFS driver with the calendar release date. Older versions of APFS drivers may contain unpatched vulnerabilities, which can be used to inflict harm on your computer. This option permits restricting APFS drivers to only recent releases.

   - `0` — require the default supported release date of APFS in OpenCore. The default release date will increase with time and thus this setting is recommended. Currently set to 2020/01/01.
   - `-1` — permit any release date to load (strongly discouraged).
   - Other — use custom minimal APFS release date, e.g. `20200401` for 2020/04/01. APFS release dates can be found in OpenCore boot log and `OcApfsLib`.

5. `MinVersion`
   Type: `plist integer`
   Failsafe: `0`
   Description: Minimal allowed APFS driver version.

   APFS driver version connects APFS driver with the macOS release. APFS drivers from older macOS releases will become unsupported and thus may contain unpatched vulnerabilities, which can be used to inflict harm on your computer. This option permits restricting APFS drivers to only modern macOS versions.

   - `0` — require the default supported version of APFS in OpenCore. The default version will increase with time and thus this setting is recommended. Currently set to the latest point release from High Sierra.
   - `-1` — permit any version to load (strongly discouraged).
   - Other — use custom minimal APFS version, e.g. `1412101001000000` from macOS Catalina 10.15.4. APFS versions can be found in OpenCore boot log and `OcApfsLib`.

11.8 Audio Properties

1. `AudioCodec`
   Type: `plist integer`
   Failsafe: `0`
   Description: Codec address on the specified audio controller for audio support.

   Normally this contains first audio codec address on the builtin analog audio controller (`HDEF`). Audio codec addresses, e.g. `2`, can be found in the debug log (marked in bold):

   ```
   OCAU: 1/3 PciRoot(0x0)/Pci(0x1,0x0)/Pci(0x0,0x1)/VenMsg(<redacted>,00000000) (4 outputs)
   OCAU: 2/3 PciRoot(0x0)/Pci(0x3,0x0)/VenMsg(<redacted>,00000000) (1 outputs)
   OCAU: 3/3 PciRoot(0x0)/Pci(0x1B,0x0)/VenMsg(<redacted>,02000000) (7 outputs)
   ```

   As an alternative this value can be obtained from `IOHDACodecDevice` class in I/O Registry containing it in `IOHDACodecAddress` field.

2. `AudioDevice`
   Type: `plist string`

12 Troubleshooting

12.1 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.
- To install Windows, macOS, and OpenCore on the same drive you can specify Windows bootloader path (`\EFI\Microsoft\Boot\bootmgfw.efi`) in `BlessOverride` section.
- 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 setting SystemUUID to the original firmware UUID. 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~~ or making the use `Custom` `UpdateSMBIOSMode`. Other 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 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
```