OpenCore

Reference Manual (0.7.~~0~~.1)

[2021.06.27]

3 Setup

3.1 Directory Structure

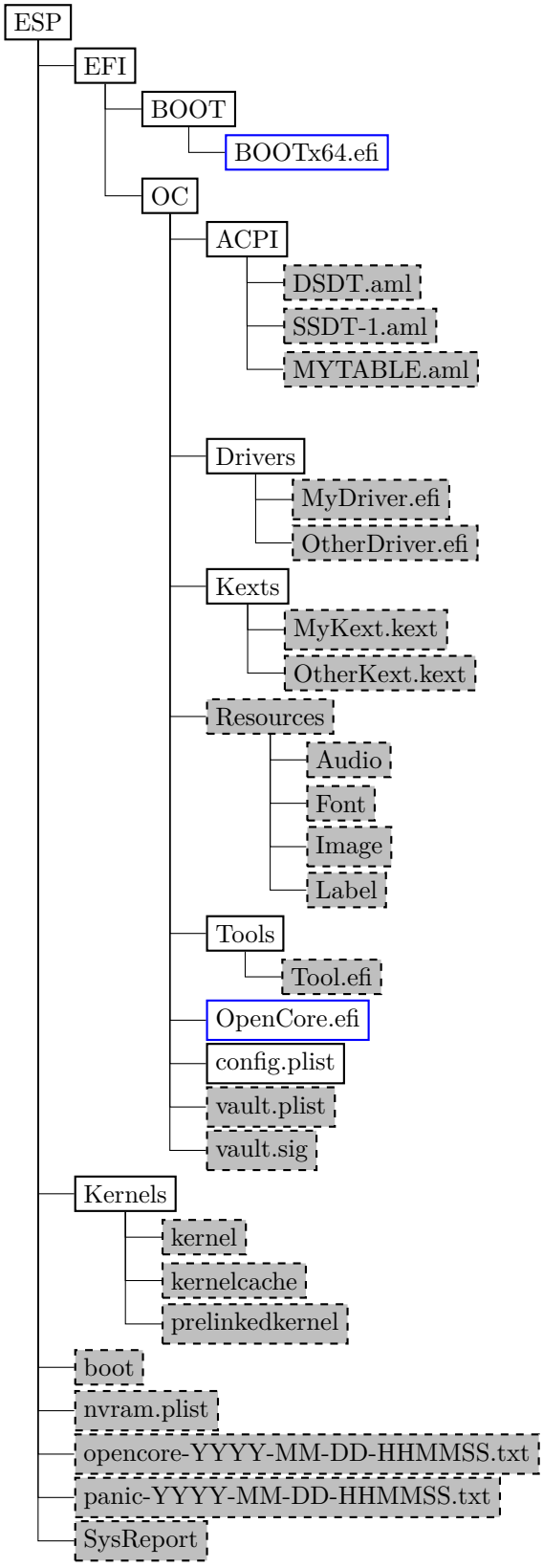

Figure 1. Directory Structure

When directory boot is used, the directory structure used should follow the descriptions in the Directory Structure

open-source implementations with transparent binary generation is encouraged (e.g. OCAT), since other tools may contain malware. Remember that a configuration made for a different hardware setup shall never be used on another hardware setup.

For BIOS booting, a third-party UEFI environment provider is required and `OpenDuetPkg` is one such UEFI environment provider for legacy systems. To run OpenCore on such a legacy system, `OpenDuetPkg` can be installed with a dedicated tool — BootInstall (bundled with OpenCore). Third-party utilities can be used to perform this on systems other than macOS.

For upgrade purposes, refer to the `Differences.pdf` document which provides information about changes to the configuration (as compared to the previous release) as well as to the `Changelog.md` document (which contains a list of modifications across all published updates).

3.3 Contribution

OpenCore can be compiled as a standard EDK II package and requires the EDK II Stable package. The currently supported EDK II release is hosted in acidanthera/audk. Required patches for this package can be found in the `Patches` directory.

The only officially supported toolchain is `XCODE5`. Other toolchains might work but are neither supported nor recommended. Contributions of clean patches are welcome. Please do follow EDK II C Codestyle.

To compile with `XCODE5`, besides Xcode, users should also install NASM and MTOC. The latest Xcode version is recommended for use despite the toolchain name. An example command sequence is as follows:

```
git clone --depth=1 https://github.com/acidanthera/audk UDK
cd UDK
git submodule update --init --recommend-shallow
git clone --depth=1 https://github.com/acidanthera/OpenCorePkg
source edksetup.sh
make -C BaseTools
build -a X64 -b RELEASE -t XCODE5 -p OpenCorePkg/OpenCorePkg.dsc
```

Listing 1: Compilation Commands

For IDE usage Xcode projects are available in the root of the repositories. Another approach could be using Language Server Protocols. For example, Sublime Text with ~~EasyClangComplete~~LSP for Sublime Text plugin. Add ~~.clang~~compile_~~complete~~flags.txt file with similar content to the UDK root:

```
-I/UefiPackages/MdePkg
-I/UefiPackages/MdePkg/Include
-I/UefiPackages/MdePkg/Include/X64
-I/UefiPackages/MdeModulePkg
-I/UefiPackages/MdeModulePkg/Include
-I/UefiPackages/MdeModulePkg/Include/X64
-I/UefiPackages/OpenCorePkg/Include/AMI
-I/UefiPackages/OpenCorePkg/Include/Acidanthera
-I/UefiPackages/OpenCorePkg/Include/Apple
-I/UefiPackages/OpenCorePkg/Include/Apple/X64
-I/UefiPackages/OpenCorePkg/Include/Duet
-I/UefiPackages/OpenCorePkg/Include/Generic
-I/UefiPackages/OpenCorePkg/Include/Intel
-I/UefiPackages/OpenCorePkg/Include/Microsoft
-I/UefiPackages/OpenCorePkg/Include/Nvidia
-I/UefiPackages/OpenCorePkg/Include/VMware
-I/UefiPackages/OvmfPkg/Include
-I/UefiPackages/ShellPkg/Include
-I/UefiPackages/UefiCpuPkg/Include
-IInclude
-include
/UefiPackages/MdePkg/Include/Uefi.h
```

```
-fshort-wchar
-Wall
-Wextra
-Wno-unused-parameter
-Wno-missing-braces
-Wno-missing-field-initializers
-Wno-tautological-compare
-Wno-sign-compare
-Wno-varargs
-Wno-unused-const-variable
-DOC_TARGET_NOOPT=1
-DNO_MSABI_VA_FUNCS=1
```

Listing 2: ECC Configuration

Note: `/UefiPackages` in the sample file denotes an absolute path.

Warning: Tool developers modifying `config.plist` or any other OpenCore files must ensure that their tools check the `opencore-version` NVRAM variable (see the Debug Properties section below) and warn users if the version listed is unsupported or prerelease. The OpenCore configuration may change across releases and such tools shall ensure that they carefully follow this document. Failure to do so may result in such tools being considered to be malware and blocked by any means.

3.4 Coding conventions

As with any other project, we have conventions that we follow during development. All third-party contributors are advised to adhere to the conventions listed below before submitting patches. To minimise abortive work and the potential rejection of submissions, third-party contributors should initially raise issues to the Acidanthera Bugtracker for feedback before submitting patches.

Organisation. The codebase is contained in the `OpenCorePkg` repository, which is the primary EDK II package.

- Whenever changes are required in multiple repositories, separate pull requests should be sent to each.
- Committing the changes should happen firstly to dependent repositories, secondly to primary repositories to avoid automatic build errors.
- Each unique commit should compile with `XCODE5` and preferably with other toolchains. In the majority of the cases it can be checked by accessing the CI interface. Ensuring that static analysis finds no warnings is preferred.
- External pull requests and tagged commits must be validated. That said, commits in master may build but may not necessarily work.
- Internal branches should be named as follows: `author-name-date`, e.g. `vit9696-ballooning-20191026`.
- Commit messages should be prefixed with the primary module (e.g. library or code module) the changes were made in. For example, `OcGuardLib: Add OC_ALIGNED macro`. For non-library changes `Docs` or `Build` prefixes are used.

Design. The codebase is written in a subset of freestanding C11 (C17) supported by most modern toolchains used by EDK II. Applying common software development practices or requesting clarification is recommended if any particular case is not discussed below.

- Never rely on undefined behaviour and try to avoid implementation defined behaviour unless explicitly covered below (feel free to create an issue when a relevant case is not present).
- Use `OcGuardLib` to ensure safe integral arithmetics avoiding overflows. Unsigned wraparound should be relied on with care and reduced to the necessary amount.
- Check pointers for correct alignment with `OcGuardLib` and do not rely on the architecture being able to dereference unaligned pointers.
- Use flexible array members instead of zero-length or one-length arrays where necessary.
- Use static assertions (`STATIC_ASSERT`) for type and value assumptions, and runtime assertions (`ASSERT`) for precondition and invariant sanity checking. Do not use runtime assertions to check for errors as they should never alter control flow and potentially be excluded.
- Assume `UINT32`/`INT32` to be `int`-sized and use `%u`, `%d`, and `%x` to print them.
- Assume `UINTN`/`INTN` to be of unspecified size, and cast them to `UINT64`/`INT64` for printing with `%Lu`, `%Ld` and so on as normal.

2. `NormalizeHeaders`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Cleanup ACPI header fields to workaround macOS ACPI implementation flaws that result in boot crashes. Reference: Debugging AppleACPIPlatform on 10.13 by Alex James (also known as theracermaster). The issue was fixed in macOS Mojave (10.14).

3. `RebaseRegions`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Attempt to heuristically relocate ACPI memory regions. Not recommended.

   ACPI tables are often generated dynamically by the underlying firmware implementation. Among the position-independent code, ACPI tables may contain the physical addresses of MMIO areas used for device configuration, typically grouped by region (e.g. `OperationRegion`). Changing firmware settings or hardware configuration, upgrading or patching the firmware inevitably leads to changes in dynamically generated ACPI code, which sometimes results in the shift of the addresses in the aforementioned `OperationRegion` constructions.

   For this reason, the application of modifications to ACPI tables is extremely risky. The best approach is to make as few changes as possible to ACPI tables and to avoid replacing any tables, particularly DSDT tables. When this cannot be avoided, ensure that any custom DSDT tables are based on the most recent DSDT tables or attempt to remove reads and writes for the affected areas.

   When nothing else helps, this option could be tried to avoid stalls at `PCI Configuration Begin` phase of macOS booting by attempting to fix the ACPI addresses. It is not a magic bullet however, and only works with the most typical cases. Do not use unless absolutely required as it can have the opposite effect on certain platforms and result in boot failures.

4. `ResetHwSig`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reset `FACS` table `HardwareSignature` value to `0`.

   This works around firmware that fail to maintain hardware signature across the reboots and cause issues with waking from hibernation.

5. `ResetLogoStatus`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Reset `BGRT` table `Displayed` status field to `false`.

   This works around firmware that provide a `BGRT` table but fail to handle screen updates afterwards.

6. `SyncTableIds`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Sync table identifiers with the `SLIC` table.

   This works around patched tables becoming incompatible with the `SLIC` table causing licensing issues in older Windows operating systems.

Requirement: 10.8 (not required for older)
Description: Forces maximum performance in XCPM mode.

This patch writes `0xFF00` to `MSR_IA32_PERF_CONTROL` (`0x199`), effectively setting maximum multiplier for all the time.

Note: While this may increase the performance, this patch is strongly discouraged on all systems but those explicitly dedicated to scientific or media calculations. Only certain Xeon models typically benefit from the patch.

5. `CustomSMBIOSGuid`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.4
Description: Performs GUID patching for `UpdateSMBIOSMode Custom` mode. Usually relevant for Dell laptops.

6. `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.

7. `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.

8. `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.

9. `ExtendBTFeatureFlags`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.8-11
Description: Set `FeatureFlags` to `0x0F` for full functionality of Bluetooth, including Continuity.

Note: This option is a substitution for BT4LEContinuityFixup.kext, which does not function properly due to late patching progress.

10. `ExternalDiskIcons`
Type: `plist boolean`
Failsafe: `false`
Requirement: 10.4
Description: Apply icon type patches to AppleAHCIPort.kext to force internal disk icons for all AHCI disks.

18. `SetApfsTrimTimeout`
    Type: `plist integer`
    Failsafe: `-1`
    Requirement: 10.14 (not required for older)
    Description: Set trim timeout in microseconds for APFS filesystems on SSDs.

    The APFS filesystem is designed in a way that the space controlled via the spaceman structure is either used or free. This may be different in other filesystems where the areas can be marked as used, free, and *unmapped*. All free space is trimmed (unmapped/deallocated) at macOS startup. The trimming procedure for NVMe drives happens in LBA ranges due to the nature of the `DSM` command with up to 256 ranges per command. The more fragmented the memory on the drive is, the more commands are necessary to trim all the free space.

    Depending on the SSD controller and the level of drive fragmenation, the trim procedure may take a considerable amount of time, causing noticeable boot slowdown. The APFS driver explicitly ignores previously unmapped areas and repeatedly trims them on boot. To mitigate against such boot slowdowns, the macOS driver introduced a timeout (`9.999999` seconds) that stops the trim operation when not finished in time.

    On several controllers, such as Samsung, where the deallocation process is relatively slow, this timeout can be reached very quickly. Essentially, it means that the level of fragmentation is high, thus macOS will attempt to trim the same lower blocks that have previously been deallocated, but never have enough time to deallocate higher blocks. The outcome is that trimming on such SSDs will be non-functional soon after installation, resulting in additional wear on the flash.

    One way to workaround the problem is to increase the timeout to an extremely high value, which at the cost of slow boot times (extra minutes) will ensure that all the blocks are trimmed. Set this option to a high value, such as `4294967295`, to ensure that all blocks are trimmed. Alternatively, use over-provisioning, if supported, or create a dedicated unmapped partition where the reserve blocks can be found by the controller. Conversely, the trim operation can be disabled by setting a very low timeout value. e.g. `999`. Refer to this article for details.

19. `ThirdPartyDrives`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.6 (not required for older)
    Description: Apply vendor patches to IOAHCIBlockStorage.kext to enable native features for third-party drives, such as TRIM on SSDs or hibernation support on 10.15 and newer.

    Note: This option may be avoided on user preference. NVMe SSDs are compatible without the change. For AHCI SSDs on modern macOS version there is a dedicated built-in utility called `trimforce`. Starting from 10.15 this utility creates `EnableTRIM` variable in `APPLE_BOOT_VARIABLE_GUID` namespace with `01 00 00 00` value.

20. `XhciPortLimit`
    Type: `plist boolean`
    Failsafe: `false`
    Requirement: 10.11 (not required for older)
    Description: Patch various kexts (AppleUSBXHCI.kext, AppleUSBXHCIPCI.kext, IOUSBHostFamily.kext) to remove USB port count limit of 15 ports.

    Note: This option should be avoided whenever possible. USB port limit is imposed by the amount of used bits in locationID format and there is no possible way to workaround this without heavy OS modification. The only valid solution is to limit the amount of used ports to 15 (discarding some). More details can be found on AppleLife.ru.

7.9 Scheme Properties

These properties are particularly relevant for older macOS operating systems. Refer to the Legacy Apple OS section for details on how to install and troubleshoot such macOS installations.

1. `CustomKernel`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Use customised kernel cache from the `Kernels` directory located at the root of the ESP partition.

Unsupported platforms including `Atom` and `AMD` require modified versions of XNU kernel in order to boot. This option provides the possibility to using a customised kernel cache which contains such modifications from ESP partition.

2. `FuzzyMatch`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Use `kernelcache` with different checksums when available.

   On macOS 10.6 and earlier, `kernelcache` filename has a checksum, which essentially is `adler32` from SMBIOS product name and EfiBoot device path. On certain firmware, the EfiBoot device path differs between UEFI and macOS due to ACPI or hardware specifics, rendering `kernelcache` checksum as always different.

   This setting allows matching the latest `kernelcache` with a suitable architecture when the `kernelcache` without suffix is unavailable, improving macOS 10.6 boot performance on several platforms.

3. `KernelArch`
   Type: `plist string`
   Failsafe: `Auto` (Choose the preferred architecture automatically)
   Description: Prefer specified kernel architecture (`i386`, `i386-user32`, `x86_64`) when available.

   On macOS 10.7 and earlier, the XNU kernel can boot with architectures different from the usual `x86_64`. This setting will use the specified architecture to boot macOS when it is supported by the macOS and the configuration:

   - `i386` — Use `i386` (32-bit) kernel when available.
   - `i386-user32` — Use `i386` (32-bit) kernel when available and force the use of 32-bit userspace on 64-bit capable processors if supported by the operating system.
     - On macOS, 64-bit capable processors are assumed to support `SSSE3`. This is not the case for older 64-bit capable Pentium processors, which cause some applications to crash on macOS 10.6. This behaviour corresponds to the `-legacy` kernel boot argument.
     - This option is unavailable on macOS 10.4 and 10.5 when running on 64-bit firmware due to an uninitialised 64-bit segment in the XNU kernel, which causes AppleEFIRuntime to incorrectly execute 64-bit code as 16-bit code.
   - `x86_64` — Use `x86_64` (64-bit) kernel when available.

   The algorithm used to determine the preferred kernel architecture is set out below.

   (a) `arch` argument in image arguments (e.g. when launched via UEFI Shell) or in `boot-args` variable overrides any compatibility checks and forces the specified architecture, completing this algorithm.
   (b) OpenCore build architecture restricts capabilities to `i386` and `i386-user32` mode for the 32-bit firmware variant.
   (c) Determined EfiBoot version restricts architecture choice:
   - 10.4-10.5 — `i386` or `i386-user32` (only on 32-bit firmware)
   - 10.6 — `i386`, `i386-user32`, or `x86_64`
   - 10.7 — `i386` or `x86_64`
   - 10.8 or newer — `x86_64`

   (d) If `KernelArch` is set to `Auto` and `SSSE3` is not supported by the CPU, capabilities are restricted to `i386-user32` if supported by EfiBoot.
   (e) Board identifier (from SMBIOS) based on EfiBoot version disables `x86_64` support on an unsupported model if any `i386` variant is supported. `Auto` is not consulted here as the list is not overridable in EfiBoot.
   (f) `KernelArch` restricts the support to the explicitly specified architecture (when not set to `Auto`) if the architecture remains present in the capabilities.
   (g) The best supported architecture is chosen in this order: `x86_64`, `i386`, `i386-user32`.

   Unlike macOS 10.7 (where certain board identifiers are treated as the `i386` only machines), and macOS 10.5 or earlier (where `x86_64` is not supported by the macOS kernel), macOS 10.6 is very special. The architecture choice on macOS 10.6 depends on many factors including not only the board identifier, but also the macOS product type (client vs server), macOS point release, and amount of RAM. The detection of all these is complicated and impractical, as several point releases had implementation flaws resulting in a failure to properly execute the server detection in the first place. For this reason, OpenCore on macOS 10.6 falls back on the `x86_64` architecture whenever it is supported by the board, as it is on macOS 10.7.

11.3 Tools and Applications

Standalone tools may help to debug firmware and hardware. Some of the known tools are listed below. While some tools can be launched from within OpenCore (Refer to the Tools subsection for more details), most should be run separately either directly or from `Shell`.

To boot into OpenShell or any other tool directly save `OpenShell.efi` under the name of `EFI\BOOT\BOOTX64.EFI` on a FAT32 partition. It is typically unimportant whether the partition scheme is `GPT` or `MBR`.

While the previous approach works both on Macs and other computers, an alternative Mac-only approach to bless the tool on an HFS+ or APFS volume:

```
sudo bless --verbose --file /Volumes/VOLNAME/DIR/OpenShell.efi \
  --folder /Volumes/VOLNAME/DIR/ --setBoot
```

Listing 3: Blessing tool

Note 1: `/System/Library/CoreServices/BridgeVersion.bin` should be copied to `/Volumes/VOLNAME/DIR`.
Note 2: To be able to use the `bless` command, disabling System Integrity Protection is necessary.
Note 3: To be able to boot Secure Boot might be disabled if present.

Some of the known tools are listed below (builtin tools are marked with *):

`BootKicker*`	Enter Apple BootPicker menu (exclusive for Macs with compatible GPUs).
`ChipTune*`	Test BeepGen protocol and generate audio signals of different style and length.
`CleanNvram*`	Reset NVRAM alternative bundled as a standalone tool.
`CsrUtil*`	Simple implementation of SIP-related features of Apple `csrutil`.
`GopStop*`	Test GraphicsOutput protocol with a simple scenario.
`KeyTester*`	Test keyboard input in `SimpleText` mode.
`MemTest86`	Memory testing utility.
`OpenControl*`	Unlock and lock back NVRAM protection for other tools to be able to get full NVRAM access when launching from OpenCore.
`OpenShell*`	OpenCore-configured `UEFI Shell` for compatibility with a broad range of firmware.
`PavpProvision`	Perform EPID provisioning (requires certificate data configuration).
`ResetSystem*`	Utility to perform system reset. Takes reset type as an argument: `coldreset`, `firmware`, `shutdown`, `warmreset`. Defaults to `coldreset`.
`RtcRw*`	Utility to read and write RTC (CMOS) memory.
`ControlMsrE2*`	Check `CFG Lock` (MSR `0xE2` write protection) consistency across all cores and change such hidden options on selected platforms.
`TpmInfo*`	Check Intel PTT (Platform Trust Technology) capability on the platform, which allows using fTPM 2.0 if enabled. The tool does not check whether fTPM 2.0 is actually enabled.

11.4 OpenCanopy

OpenCanopy is a graphical OpenCore user interface that runs in `External PickerMode` and relies on OpenCorePkg `OcBootManagementLib` similar to the builtin text interface.

OpenCanopy requires graphical resources located in `Resources` directory to run. Sample resources (fonts and images) can be found in OcBinaryData repository. Customised icons can be found over the internet (e.g. here or there).

OpenCanopy provides full support for `PickerAttributes` and offers a configurable builtin icon set. The chosen icon set may depend on the `DefaultBackgroundColor` variable value. Refer to `PickerVariant` for more details.

Predefined icons are saved in the `PickerVariant`-derived subdirectory of the `\EFI\OC\Resources\Image` directory. A full list of supported icons (in `.icns` format) is provided below. When optional icons are missing, the closest available icon will be used. External entries will use `Ext`-prefixed icon if available (e.g. `OldExtHardDrive.icns`).

Note: In the following all dimensions are normative for the 1x scaling level and shall be scaled accordingly for other levels.

- `Cursor` — Mouse cursor (mandatory, up to 144x144).
- `Selected` — Selected item (mandatory, 144x144).
- `Selector` — Selecting item (mandatory, up to 144x40).