Vendor Native Development Kit (VNDK)

The Vendor Native Development Kit (VNDK) is a set of libraries exclusively for vendors to implement their HALs. The VNDK ships in `system.img` and is dynamically linked to vendor code at runtime.

Why VNDK?

Android O enables framework-only updates in which the system partition can be upgraded to the latest version while vendor partitions are left unchanged. This implies that binaries built at different times must be able to work with each other; VNDK covers API/ABI changes across Android releases.

Framework-only updates include the following challenges:

- **Dependency between framework modules and vendor modules**. Before Android O, modules from both sides could link with modules from the other side. However, dependencies from vendor modules imposed undesired restrictions to framework modules development.
- **Extensions to AOSP libraries**. Android O requires all Android devices to pass CTS when the system partition is replaced with a standard AOSP system image. However, as vendors extend AOSP libraries to boost performance or to add extra functionalities for their HIDL implementations, flashing the system partition with a standard AOSP system image might break a vendor's HIDL implementation. (For guidelines on preventing such breakages, see VNDK extensions (https://source.android.com/devices/architecture/vndk/extensions.html).)

To address these challenges, Android O introduces several techniques such as VNDK (described in this section), HIDL (https://source.android.com/devices/architecture/hidl/index.html), hwbinder, device tree overlay (https://source.android.com/devices/architecture/dto/index.html), and sepolicy overlay.

VNDK resources

This section includes the following VNDK resources:

- *VNDK concepts* (#vndk-concepts) (below) describes framework shared libraries, same-process HALs (SP-HALs), and VNDK terminology.
- *VNDK extensions* (https://source.android.com/devices/architecture/vndk/extensions.html) classifies vendor-specific changes into categories. For example, libraries with extended functionalities on which vendor modules rely must be copied into the vendor partition, but ABI-incompatible changes are prohibited.
- The *VNDK Definition Tool* (https://source.android.com/devices/architecture/vndk/deftool.html) helps migrate your source tree to Android O.
- The *Linker Namespace* (https://source.android.com/devices/architecture/vndk/linker-namespace.html) provides fine-grained control over shared library linkages.
- *Directories, Rules, and sepolicy* (https://source.android.com/devices/architecture/vndk/dir-rules-sepolicy.html) defines the directory structure for devices running Android O, VNDK rules, and associated sepolicy.
- The *VNDK Design in Android O* (https://source.android.com/devices/architecture/images/vndk_design_android_o.pdf) presentation illustrates fundamental VDNK concepts used in Android O.

VNDK concepts

In an ideal Android O world, framework processes do not load vendor shared libraries, all vendor processes load only vendor shared libraries (and a portion of framework shared libraries), and communications between framework processes and vendor processes are governed by HIDL and hardware binder.

Such a world includes the possibility that stable, public APIs from framework shared libraries might not be sufficient for vendor module developers (although APIs can change between Android releases), requiring that some portion of framework shared libraries be accessible to vendor processes. In addition, as performance requirements can lead to compromises, some response-time-critical HALs must be treated differently.

The following sections detail how VNDK handles framework shared libraries for vendors and Same-Process HALs (SP-HALs).

Framework shared libraries for vendor

This section describes the criteria for classifying shared libraries that are accessible to vendor processes. There are two approaches to support vendor modules across multiple Android releases:

1. **Stabilize the ABIs/APIs of the framework shared libraries**. New framework modules and old vendor modules can use the same shared library to reduce memory footprint and storage size. A unique shared library also avoids several double-loading issues. However, the development cost to maintain stable ABIs/APIs is high and it is unrealistic to stabilize all ABIs/APIs exported by every framework shared library.
2. **Copy old framework shared libraries**. Comes with the strong restriction against side channels, defined as all mechanisms to communicate among framework modules and vendor modules, including (but not limited to) binder, socket, pipe, shared memory, shared file, and system properties. There must be no communication unless the communication protocol is frozen and stable (e.g. HIDL through hwbinder). Double-loading shared libraries might cause problems as well; for example, if an object created by the new library is passed into the functions from the old library, an error may occur as these libraries may interpret the object differently.

Different approaches are used depending on the characteristics of the shared libraries. As a result, framework shared libraries are classified into three sub-categories:

- *LL-NDK* and *SP-NDK* are *Framework Shared Libraries* that are known to be stable. Their developers are committed to maintain their API/ABI stabilities.
  - LL-NDK includes the following libraries: `libandroid_net.so`, `libc.so`, `libstdc++.so`, `libdl.so`, `liblog.so`, `libm.so`, `libz.so`, and `libvndksupport.so`.
  - SP-NDK includes the following libraries: `libEGL.so`, `libGLESv1_CM.so`, `libGLESv2.so`, `libGLESv3.so`, `libvulkan.so`, `libnativewindow.so`, and `libsync.so`.
- *Eligible VNDK Libraries (VNDK)* are *Framework Shared Libraries* that are safe to be copied twice. *Framework Modules* and *Vendor Modules* can link with their own copies. A framework shared library can become an eligible VNDK library only if it satisfies the following criteria:
  - It does not send/receive IPCs to/from the framework.
  - It is not related to ART virtual machine.
  - It does not read/write files/partitions with unstable file formats.
  - It does not have special software license which requires legal reviews.
  - Its code owner does not have objections to vendor usages.
- *Framework-Only Libraries (FWK-ONLY)* are *Framework Shared Libraries* that do not belong to the categories mentioned above. These libraries:
  - Are considered framework internal implementation details.
  - Must not be accessed by vendor modules.
  - Have unstable ABIs/APIs and no API/ABI compatibility guarantees.
  - Are not copied.

Same-Process HAL (SP-HAL)

Same-Process HAL (*SP-HAL*) is a set of predetermined HALs implemented as *Vendor Shared Libraries* and loaded into *Framework Processes*. SP-HALs are isolated by a linker namespace (controls the libraries and symbols that are visible to the shared libraries). SP-HALs must depend only on *LL-NDK*, *SP-NDK*, and *VNDK-SP*.

VNDK-SP is a predefined subset of eligible VNDK libraries. VNDK-SP libraries are carefully reviewed to ensure double-loading VNDK-SP libraries into framework processes does not cause problems. Both SP-HALs and VNDK-SPs are defined by Google.

The following libraries are approved SP-HALs:

- `libGLESv2_${driver}.so`
- `libGLESv1_CM_${driver}.so`
- `libEGL_${driver}.so`
- `vulkan.${driver}.so`
- `android.hardware.renderscript@1.0-impl.so`

- `android.hardware.graphics.mapper@2.0-impl.so`

The following libraries are VNDK-SP libraries that are accessible by SP-HALs:

- `android.hardware.graphics.allocator@2.0.so`
- `android.hardware.graphics.common@1.0.so`
- `android.hardware.graphics.mapper@2.0.so`
- `android.hardware.renderscript@1.0.so` (Renderscript)
- `libRS_internal.so` (Renderscript)
- `libbase.so`
- `libc++.so`
- `libcutils.so`
- `libhardware.so`
- `libhidlbase.so`
- `libhidltransport.so`
- `libhwbinder.so`
- `libutils.so`

The following *VNDK-SP dependencies (VNDK-SP-Indirect)* are invisible to *SP-HALs*:

- `libRSCpuRef.so` (Renderscript)
- `libRSDriver.so` (Renderscript)
- `libbacktrace.so`
- `libblas.so` (Renderscript)
- `libbcinfo.so` (Renderscript)
- `liblzma.so`
- `libunwind.so`

The following *private VNDK-SP dependency (VNDK-SP-Indirect-Private)* is invisible to all vendor modules:

- `libcompiler_rt.so` (Renderscript)

The following are *framework-only libraries with RS exceptions (FWK-ONLY-RS)*:

- `libft2.so` (Renderscript)
- `libmediandk.so` (Renderscript)

VNDK terminology

- *Modules* refer to either *Shared Libraries* or *Executables*.
- *Processes* are operating system tasks spawned from *Executables*.
- *Framework*-qualified terms refer to the concepts related to the **system** partition.
- *Vendor*-qualified terms refer to the concepts related to **vendor** partitions.

For example:

- *Framework Executables* refer to executables in `/system/bin` or `/system/xbin`.
- *Framework Shared Libraries* refer to shared libraries under `/system/lib[64]`.
- *Framework Modules* refer to both *Framework Shared Libraries* and *Framework Executables*.
- *Framework Processes* are processes spawned from *Framework Executables* (e.g. `/system/bin/app_process`).
- *Vendor Executables* refer to executables in `/vendor/bin`
- *Vendor Shared Libraries* refer to shared libraries under `/vendor/lib[64]`.
- *Vendor Modules* refer to both *Vendor Executables* and *Vendor Shared Libraries*.

- *Vendor Processes* are processes spawned from *Vendor Executables* (e.g. `/vendor/bin/android.hardware.camera.provider@2.4-service`).