Treble VNDK
Design Principles and Practical Migration
Status

- VNDK is partially implemented in **Android Oreo**
  - Only VNDK-SP (for SP-HAL) is enforced
- VNDK is fully implemented in **Android Oreo-MR1**
  - VNDK-SP (for SP-HAL) is enforced
  - Enabling VNDK is recommended
- In the future, VNDK will be fully enforced and ineligible libraries won’t be accessible by vendor modules at build-time and run-time
Agenda

- VNDK overview
- Dynamic linker support
- Build system support
- VNDK definition tool
- JNI libraries in bundled APKs
VNDK overview
Modular Android -- in view of lib dependency

- Vendor modules should not depend on system modules
  - Except: VNDK
- Framework do not depend on vendor modules
  - Except: Same-process HAL (SP-HAL)
What is VNDK?

- **VNDK** is a set of shared libraries for vendors to implement vendor modules.
  - Part of VINTF
  - Versioned, stable
Same-Process HAL

- **SP-HAL** -- Several time-critical HALs are not binderized
  - android.hardware.renderscript@1.0-impl
  - android.hardware.graphics.mapper@1.0-impl
  - android.aidl.memory@1.0-impl
  - libEGL_${chipset}
  - libGLES_${chipset}
  - vulkan.${chipset}

- What about its dependency?

  * To be precise, both SP-HAL and their dependencies applies.
VNDK categories

- **LLNDK** (LL-NDK + SP-NDK)
  - Shared libraries with stable APIs and loosely coupled with the framework
  - Both system and vendor share the same file

- **VNDK**
  - A specialized variant built for vendor modules.
  - There may be a FWK-ONLY counterpart with the same name.

- **VNDK-SP**
  - Same as VNDK
  - Can be used by SP-HALs
  - May be loaded into framework process (to be explained in the next slide and dynamic linker namespace section)
VNDK-SP: Dependency of Same-Process HAL

- SP-HAL must only depend on LLNDK or VNDK-SP
  - VNDK-SP and its FWK-ONLY counterpart (shared lib with same name) may be loaded into the same process.

![Diagram showing dependencies between libraries](image)
Other categories

- **FWK-ONLY**
  - Other shared libraries on the system partition
  - Vendor modules must not depend on these libraries

- **VND-ONLY**
  - Other (i.e., non-SP-HAL) shared libraries on the vendor partition
  - Framework modules must not depend on these libraries

Any cross-partition dependencies must be in LLNDK, VNDK, VNDK-SP and SP-HAL.
- This is recommended in O-MR1 and will be enforced in P.
Glimpse of categories 1/3

System Partition
- FWK-ONLY
- NDK
- LLNDK

Vendor Partition
- VND-ONLY
- VNDK
- VNDK-SP
- SP-HAL
- SP-HAL-Deps
Glimpse of categories

**LLNDK**

- libEGL
- libGLESv1_CM
- libGLESv2
- libGLESv3
- libRS
- libandroid_net#
- libc
- libdl
- liblog
- libm
- libnativewindow
- libsync
- libvndksupport#

**VNDK-SP**

- android.hardware.graphics_allocator@2.0
- android.hardware.graphics_common@1.0
- android.hardware.graphics_mapper@2.0
- android.hardwarerenderscript@1.0
- android.hidel.memory@1.0
- android.hidel.memory@1.0-impl
- libRSCpuRef
- libRSDriver
- libRS_internal
- libbacktrace
- libbase
- libbcinfo
- libblas
- libc++

- libcompiler_rt
- libcutils
- libhardware
- libhidlbase
- libhidlmemory
- libhidltransport
- libhwcomposer
- libiw
- liblzma
- libunwind
- libunwindstack
- libutils
- libz*

* In some configurations, libz belongs to LLNDK but there should be no differences.
# These are LLNDK but not NDK.

The definitive list can be found in /system/etc/ld.config.txt.
Glimpse of categories 3/4

- **Eligible list** is a list of shared libraries that have been reviewed.
  - Can be found in:
    - `${AOSP}/development/vndk/tools/definition-tool/datasets/eligible-list*.csv`
Glimpse of categories 4/4

Some NDK libs not visible to vendor modules

libandroid.so
libbaudio.so
libcamera2ndk.so
libicui18n.so
libicuuc.so
libjnigraphics.so
libmediandk.so
libneuralnetworks.so
libOpenMAXAL.so
libOpenSLES.so
libstdc++.so*
libvulkan.so
libwebviewchromium_plat_support.so

These libraries are **highly coupled** with the framework, thus they do not belong to LLNDK.

Vendor modules must **not** depend on these shared libraries.

* Use libc++ instead of libstdc++.
Vendor modules may need **extra APIs** or **extra functionalities** from the VNDK libraries.

VNDK can be extended, but they must remain **ABI compatible** to the AOSP VNDK.

- Symbols must not be removed.
- Exposed structures must not be altered (including struct/class layout and vtable)
- The idea is to make sure all extensions are drop-in replacements of the AOSP VNDK shared libraries.

```c
struct Example {
    int a_
    int bias_
};

Example *example_create(int a) {
    Example *e = (Example *)malloc(sizeof(Example));
    e->a_ = a;
    e->bias_ = rand();
    return e;
}

int example_get_a(Example *e) {
    return e->a_ + e->bias_
}

/* Extensions */
void example_set_bias(Example *e, int b) {
    e->bias_ = b;
}
```
Extended VNDK libraries must be installed into 
/vendor/lib[64]/{,vndk,vndk-sp}

- Otherwise, vendor modules will fail VTS tests on GSI, which is required to pass compliance.
- Use this as a last resort because extended VNDK shared libraries are not framework-only OTA updatable.
- VNDK definition tool can provide some preliminary set (will be introduced later)
Degenerated VNDK (O) vs. Treble VNDK (O-MR1)

In Android O, **degenerated VNDK directory layout** is adopted:

- VNDK-SP libraries have extra copies in `/system/lib[64]/vndk-sp`
- Both framework and vendor modules are using shared libraries in `/system/lib[64]`

In Android O-MR1, **Treble VNDK directory layout** is adopted:

- VNDK-SP libraries have extra copies in `/system/lib[64]/vndk-sp`
- VNDK libraries have extra copies `/system/lib[64]/vndk`
- Vendor modules are only using `/system/lib[64]/{vndk,vndk-sp}`
- Framework modules are only using `/system/lib[64]`
## Directory layout

<table>
<thead>
<tr>
<th></th>
<th>Android O</th>
<th>Android O-MR1</th>
<th>Independent system updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>FWK-ONLY</td>
<td>/system/lib[64]</td>
<td>/system/lib[64]</td>
<td>Everything may change</td>
</tr>
<tr>
<td>LLNDK</td>
<td>/system/lib[64]</td>
<td>/system/lib[64]</td>
<td>New APIs or implementation</td>
</tr>
<tr>
<td>VNDK-SP</td>
<td>/system/lib[64]/vndk-sp</td>
<td>/system/lib[64]/vndk-sp</td>
<td>Old APIs with security fixes</td>
</tr>
<tr>
<td>VNDK-SP-EXT</td>
<td>/vendor/lib[64]/vndk-sp</td>
<td>/vendor/lib[64]/vndk-sp</td>
<td>N/A</td>
</tr>
<tr>
<td>VNDK</td>
<td>/system/lib[64] (degenerated)</td>
<td>/system/lib[64]/vndk</td>
<td>Old APIs with security fixes (only O-MR1)</td>
</tr>
<tr>
<td>VNDK-EXT</td>
<td>/vendor/lib[64]</td>
<td>/vendor/lib[64]/vndk</td>
<td>N/A</td>
</tr>
<tr>
<td>VND-ONLY</td>
<td>/vendor/lib[64]</td>
<td>/vendor/lib[64]</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Dynamic linker support
Motivation: Isolate SP-HAL and VNDK-SP

- SP-HAL from the vendor partition will be loaded into framework processes.
- SP-HAL may depend on VNDK-SP.
- Framework modules may depend on FWK-ONLY counterpart (shared lib with same name with VNDK-SP).
- Loading two shared libraries with the same soname causes some problems.
  - They may have different symbols after updates.
- Enforced in Android Oreo (PRODUCT_FULL_TREBLE:=true)
Motivation: Isolate system and vendor

- Isolate the shared library dependencies
  - Dynamic linker should not load shared libraries from the other partition except VNDK or SP-HAL.

- Requirement
  - Not in Oreo
  - Recommended in Oreo-MR1
  - Enforced in Pi

- In Oreo-MR1, if BOARD_VNDK_VERSION is specified, then this will be enforced by default.
  - To disable this, add BOARD_VNDK_RUNTIME_DISABLE:=true
Dynamic linker namespace

- **Dynamic linker** /system/bin/linker[64] is a part of Bionic that loads and links ELF shared objects at run-time.
  a. It is the first program being run after the kernel maps the executable into memory.
  b. It is responsible to load DT_NEEDED entries and resolve undefined symbols.
  c. It implements dlopen() and android_dlopen_ext().

- **Dynamic linker namespace** is a mechanism to
  a. Isolate shared libraries
  b. Provide fine-grained control on dynamic shared libraries resolution
  c. Provide fine-grained control on symbol resolution

- Dynamic linker namespace is the underlying mechanism to isolate SP-HALs and VNDK-SP.
Framework process linker namespaces (Oreo-MR1)

- **default namespace**
  - `<search.paths>>
  - `/system/lib[64]
  - `/vendor/lib[64] (removed)

- **sphal namespace**
  - `<search.paths>>
  - `/vendor/lib[64]/egl
  - `/vendor/lib[64]/hw
  - `/vendor/lib[64]

- **vndk namespace**
  - `<search.paths>>
  - `/system/lib[64]/vndk-sp
  - `/vendor/lib[64]/vndk-sp

- **rs namespace**
  - `<search.paths>>
  - `/system/lib[64]/vndk-sp
  - `/vendor/lib[64]/vndk-sp
  - `/vendor/lib[64]

Note: In Android O-MR1, `/vendor/lib[64]` is removed from the default linker namespace when `BOARD_VNDK_RUNTIME_DISABLE` is not defined.
Vendor process linker namespaces (Oreo)

**default namespace**

`<<search.paths>>
/vendor/lib[64]
/system/lib[64]`

Note: `/system/lib[64]` is in the default linker namespace of vendor processes in Android O. However, it will be removed in O-MR1 if `BOARD_VNDK_RUNTIME_DISABLE` is not defined.
Vendor process linker namespaces (Oreo-MR1)

**default namespace**

```
<<search.paths>>
/vendor/lib[64]
/vendor/lib[64]/vndk
/system/lib[64]/vndk
/vendor/lib[64]/vndk-sp
/system/lib[64]/vndk-sp
```

**system namespace**

```
<<search.paths>>
/system/lib[64]
```

Note: /system/lib[64] will be removed from the default linker namespace of vendor processes in Android O-MR1 if BOARD_VNDK_RUNTIME_DISABLE is not defined.
Dynamic linker namespace is configured by /system/etc/ld.config.txt

- **INI file format**
  - Source code is at ${AOSP}/system/core/rootdir/etc/ld.config*.txt

**ld.config.txt must **not be modified.**
- There will be a CTS checking whether this file is intact.
- However, learning the file format can help understanding how does VNDK work.
dir.name assignments specify the section which will be chosen.

- For example, [system] section will be chosen if the main executable of the process resides in /system/bin.

Each section represents a graph with (1) several linker namespaces as nodes and (2) several links for fallback lookups.

dir.system = /system/bin
dir.vendor = /vendor/bin

[system]
additional.namespaces = sphal,vndk,rs
namespace.default.isolated = true
namespace.default.search.paths = ...
namespace.default.permitted.paths = ...

namespace.sphal.isolated = true
namespace.sphal.visible = true
namespace.sphal.search.paths = ...
namespace.sphal.permitted.paths = ...
namespace.sphal.link.default.shared_libs =

[vendor]
For each section

- `additional.namespaces` specifies the names of other linker namespace in addition to the `default` namespace.

For each linker namespace

- `isolated` specifies whether `permitted.paths` is enforced
- `permitted.paths` specifies the permitted path (in addition to `search.paths`) when `isolated` is true.
- `search.paths` specifies the directories which will be search for when dynamic linker is resolving an `soname`.
- `visible` specifies whether the namespace can be found by `android_get_exported_namespace()`

```
dir.system = /system/bin
dir.vendor = /vendor/bin

[system]
additional.namespaces = sphal,vndk,rs

namespace.default.isolated = true
namespace.default.search.paths = ...
namespace.default.permitted.paths = ...

namespace.sphal.isolated = true
namespace.sphal.visible = true
namespace.sphal.search.paths = ...
namespace.sphal.permitted.paths = ...
namespace.sphal.link.default.shared_libs = ...

[vendor]
```

* If someone pass a full path to `dlopen()`, then `search.paths` is irrelevant.
ld.config.txt: Fallback links

- `namespace.${name}.link.${another}.shared_libs` specifies the `soname` that can go through the fallback link to the linker namespace `${another}`.
  - If an `soname` cannot be resolved in linker namespace `${name}` and the `soname` is one of the property value, then the dynamic linker will try to resolve the `soname` in the linker namespace `${another}`.
  - For example, if `/vendor/lib/hw/vulkan.${chipset}.so` depends on `libc.so` but `libc.so` is neither in `/vendor/lib/hw` nor `/vendor/lib`, thus the dynamic linker will try to find `libc.so` in the `default` namespace.

```system
dir.system = /system/bin

[system]
additional.namespaces = sphal,vndk,rs

namespace.default.search.paths = /system/${LIB}

namespace.sphal.search.paths = /vendor/${LIB}/hw:/vendor/${LIB}

namespace.sphal.link.default.shared_libs = libc.so:libm.so
```
Build system support
Motivations

● **Duplicate shared libraries when necessary**
  ○ Build vendor variant for users in vendor partitions
  ○ VNDK, VNDK-SP may be duplicated when needed

● **Make the build dependencies explicit**
  ○ Check whether headers, static libraries, shared libraries are available
  ○ Define the VNDK libraries that have to be installed into GSI
  ○ Generate VNDK snapshots for cross version system image development
Recap: VNDK categories

- **LLNDK**  (LL-NDK + SP-NDK)
  - Shared libraries with stable APIs and loosely coupled with the framework
  - Both system and vendor share the same file

- **VNDK**
  - A specialized variant built for vendor modules.
  - There may be a FWK-ONLY counterpart with the same name.

- **VNDK-SP**
  - Same as VNDK
  - Can be used by SP-HALs
  - May be loaded into framework process (to be explained in the next slide and dynamic linker namespace section)
Build system support (Oreo)

- To move a module to vendor partition:
  - Add `LOCAL_VENDOR_MODULE:=true` to `Android.mk` (or `LOCAL_PROPRIETARY_MODULE`)
  - Add `vendor:true` to `Android.bp` (or proprietary)

- To install a module to both system and vendor partitions, you will need tricky build rules on the right or in the document:
  - Basically, this trick assigns intermediate files to `LOCAL_PREBUILT_MODULE_FILE`.

```makefile
define define-vndk-lib
include $$(@CLEAR_VARS)
LOCAL_MODULE := $1.$2
LOCAL_MODULE_CLASS := SHARED_LIBRARIES
LOCAL_PREBUILT_MODULE_FILE := $$(@TARGET_OUT_INTERMEDIATE_LIBRARIES)/$1.so
LOCAL_STRIP_MODULE := false
LOCAL_MULTLIB := first
LOCAL_MODULE_TAGS := optional
LOCAL_INSTALLED_MODULE_STEM := $1.so
LOCAL_MODULE_SUFFIX := .so
LOCAL_MODULE_RELATIVE_PATH := $3
LOCAL_VENDOR_MODULE := $4
include $$(@BUILD_PREBUILT)
endif # TARGET_TRANSLATE_2ND_ARCH is not true
endif # TARGET_2ND_ARCH is not empty
endef
```
Build system support (Oreo-MR1) 1/4

- “BOARD_VNDK_VERSION := current” enables full VNDK support.

- If “BOARD_VNDK_VERSION := current” is specified in BoardConfig.mk, then build system will:
  - Check the header search path (and removes global default search paths)
  - Check the link types of the shared libraries (i.e. vendor module can only either link to LLNDK or vendor_available)
  - Build vendor-specific VNDK libraries and install them to /system/lib[64]/vndk, vndk-sp
  - Build vendor-specific libraries and install them to /vendor/lib[64]

- VNDK-related properties in Android.bp:
  - vendor: true
  - vendor_available: true
  - vndk.enabled: true
  - vndk.support_system_process: true
Build system support (Oreo-MR1) 2/4

- **vendor** specifies whether an Android.bp module is a vendor module or not.
  - If this value is false, then it cannot depend on the module with vendor equals to true.
  - If this value is true, then it can only depend on LLNDK or the module with vendor_available equals to true.

- **vendor_available** specifies whether an Android.bp module (header lib, static lib, or shared lib) is available to vendor.
  - If this value is true and a framework module uses this module, then this module will be installed into the system partition.
  - If this value is true and a vendor module uses this module, then the vendor variant will be built. If vndk.enabled is false (or not defined), then this will be installed to /vendor/lib[64]; otherwise, this module will be installed to /system/lib[64]/vndk or /system/lib[64]/vndk-sp.
Build system support (Oreo-MR1)

- **vndk section**
  - `vndk.enabled` specifies whether an Android.bp module is a VNDK library or not. It is a prerequisite to set `vendor_available` to true.
  - `vndk.support_system_process` specifies whether an Android.bp module is a VNDK-SP library or not. Both `vendor_available` and `vndk.enabled` are prerequisites.
Build system support (Oreo-MR1)

- **target.vendor** specifies vendor-specific build options.
  - Use the **exclude_srcs** property if you would like to exclude framework-specific source files.
  - Use the **exclude_shared_libs** property if you would like to exclude framework-specific shared libraries.

```plaintext
cc_library {
    name: "libvnd_specific_example",
    vendor_available: true,
    target: {
        vendor: {
            exclude_srcs: ["framework_only.c"],
            exclude_shared_libs: ["libfwk_only"],
            cflags: ["-DEXTRA_VND_C_FLAGS"],
            cppflags: ["-DEXTRA_VND_CPP_FLAGS"],
        },
    },
}
```
Sum up

- Define a vendor module which must be installed to vendor partition
  - LOCAL_VENDOR_MODULE := true (Android.mk)
  - vendor: true (Android.bp)

- Enable full VNDK build-time support (O-MR1)
  - BOARD_VNDK_VERSION := current (BoardConfig.mk)
  - Build two variants: vendor_available: true
  - VNDK: vndk.enabled: true
  - VNDK-SP: vndk.support_system_process: true

- Disable run-time dynamic linker isolation between framework and vendor (O-MR1)
  - BOARD_VNDK_RUNTIME_DISABLE := true (BoardConfig.mk)
VNDK definition tool
VNDK definition tool

- **VNDK definition tool** scans the shared library dependencies, computes VNDK sets, and checks the existence of dependency violations.
  
  - Source: 
    ```
    ${AOSP}/development/vndk/tools/definition-tool/vndk_definition_tool.py
    ```
VNDK definition tool - commands

- **vndk** -- List VNDK libraries and other libraries that should be copied to vendor partitions.
- **check-dep** -- Check whether there are violations in the shared library dependencies.
- **deps** -- Print all resolved dependencies of shared libraries.
- **deps-insight** -- Create a HTML to show the shared library dependencies.
VNDK definition tool: vndk command

- **vndk** command lists VNDK-SP libraries and other libraries that should be copied to vendor partitions.

- Command line options:
  - `--system`: Path to your system partition directory
  - `--vendor`: Path to your vendor partition directory
  - `--aosp-system`: Path to GSI system partition directory. Convert image with simg2img then mount the image.
  - `--tag-file`: Path to eligible list CSV file
  - `--load-extra-deps`: Path to a file specifies extra shared library dependencies.
  - `--full`: List all categories (for debugging)
VNDK definition tool: check-dep command

- **check-dep** command checks the dependencies and list the violating shared libraries and symbols.

- Command line options: (in addition to vndk command)
  - **--module-info**: Path to `${ANDROID_PRODUCT_OUT}/module-info.json`

- For each violations, following information will be printed:
  - Violating module and its source path
  - Ineligible dependencies and its source path
  - The imported symbols from the ineligible dependencies.

```bash
ccheck-dep \--system path/system \--vendor path/vendor \--aosp-system path/gsi/system \--tag-file eligible-list.csv \--load-extra-deps deps.txt \--module-info module-info.json

/vendor/lib/libviolating.so
  MODULE_PATH: libviolating/source
/system/lib/libineligible1.so
  MODULE_PATH: ineligible1/source
  symbol_a
  symbol_b
/system/lib/libineligible2.so
  MODULE_PATH: ineligible2/source
  symbol_c
```
VNDK definition tool: deps and deps-insight command

- **deps** and **deps-insight** are debugging commands which will print all dependencies of shared libraries.
  - deps prints plain text output
  - deps-insight generates HTML for interactive investigation.
- Command line options are similar to vndk/check-dep command.

```bash
vndk_definition_tool.py deps \
  --system path/system \
  --vendor path/vendor \
  --load-extra-deps deps.txt \
  --module-info module-info.json

vndk_definition_tool.py deps-insight \
  --system path/system \
  --vendor path/vendor \
  --aosp-system path/gsi/system \
  --tag-file eligible-list.csv \
  --load-extra-deps deps.txt \
  --module-info module-info.json
```
JNI libraries in bundled APKs
# JNI libraries in bundled apps (O-MR1)

<table>
<thead>
<tr>
<th>Shared libraries location</th>
<th>Bundled system app /system/app</th>
<th>Bundled vendor app /vendor/app</th>
<th>Downloaded app /data/app</th>
</tr>
</thead>
<tbody>
<tr>
<td>/system/lib[64]</td>
<td>All</td>
<td>/system/etc/public.libraries.txt (NDK) + LLNDK</td>
<td>/system/etc/public.libraries.txt (NDK)</td>
</tr>
<tr>
<td>/vendor/lib[64]</td>
<td>/vendor/etc/public.libraries.txt</td>
<td>All</td>
<td>/vendor/etc/public.libraries.txt</td>
</tr>
<tr>
<td>/system/lib[64]/vndk-sp</td>
<td>x</td>
<td>Public VNDK-SP</td>
<td>x</td>
</tr>
<tr>
<td>/system/lib[64]/vndk</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Q&A

Full Documentation:
https://source.android.com/devices/architecture/vndk/
Q & A

If you have more questions, please talk to your Technical Account Manager, 3PL or SoC POC. You may also contact us at: android-treble-questions@google.com