Android Automotive »

AIDL for HALs

Android 11 introduces the ability to use AIDL for HALs in Android. This makes it possible to implement parts of Android without HIDL. HALs using AIDL to communicate between framework components must use Stable AIDL.

#android #hal #aidl

Last update: 2022-07-14

Table of Content

⚠ **This guide is written for AOSP `android-12.1.0_r8`**

AIDL for HALs

ℹ Official overview guide is at https://source.android.com/devices/architecture/aidl/aidl-hals.

AIDL has been around longer than HIDL (only from Android 8 to Android 10), and is used in many other places, such as between Android framework components or in apps. Now that AIDL has stability support, it's possible to implement an entire stack with a single IPC runtime. AIDL also has a better versioning system than HIDL.

AIDL HAL interfaces

For an AIDL interface to be used between system and vendor, the interface needs two changes:

- Every type definition must be annotated with `@VintfStability`.
- The `aidl_interface` declaration needs to include `stability: "vintf",`

AOSP Stable AIDL interfaces for HALs are in the same base directories as HIDL interfaces, in `aidl` folders.

- `hardware/interfaces`
- `frameworks/hardware/interfaces`
- `system/hardware/interfaces`

Example:

hardware/interfaces/light/aidl/Android.bp

```
aidl_interface {
    name: "android.hardware.light",
    vendor_available: true,
    srcs: [
        "android/hardware/light/*.aidl",
    ],
    stability: "vintf",
    backend: {
        java: {
            sdk_version: "module_current",
        },
        ndk: {
            vndk: {
                enabled: true,
            },
        },
```

```
    },
    versions: ["1"],
}
```

AIDL runtime library

AIDL has three different backends: Java, NDK, CPP. To use Stable AIDL, you must always use the system copy of `libbinder` at `system/lib*/libbinder.so` and talk on `/dev/binder`.

For code on the vendor image, this means that `libbinder` (from the VNDK) cannot be used. Instead, native vendor code must use the NDK backend of AIDL, link against libbinder_ndk (which is backed by system libbinder.so), and link against the -ndk_platform libraries created by aidl_interface entries.

Convert HIDL to AIDL

Build the tool `hidl2aidl` if it is not compiled:

```
m hidl2aidl
```

Create a new folder `aidl` in the HAL interface:

```
mkdir -p hardware/interfaces/invcase/aidl
```

Generate AIDL from a specific HIDL version:

```
hidl2aidl -o hardware/interfaces/invcase/aidl \
          -r android.hardware:hardware/interfaces \
          android.hardware.invcase@1.0
```

This will create `aidl/android/hardware/invcase/IInvcase.aidl` file.

AIDL

```
package android.hardware.invcase;

@VintfStability
interface IInvcase {
    String getChars();
    void putChars(in String msg);
}
```

HIDL

```
package android.hardware.invcase@1.0;

interface IInvcase {
    putChars(string msg);
    getChars() generates (string msg);
};
```

The tool also generates a makefile for the AIDL, but it is not usable at the moment:

AIDL

```
aidl_interface {
    name: "android.hardware.invcase",
    vendor: true,
    srcs: ["android/hardware/invcase/*.aidl"],
    stability: "vintf",
    owner: "vqtrong",
    backend: {
        cpp: {
            enabled: false,
        },
        java: {
            sdk_version: "module_current",
        },
    },
}
```

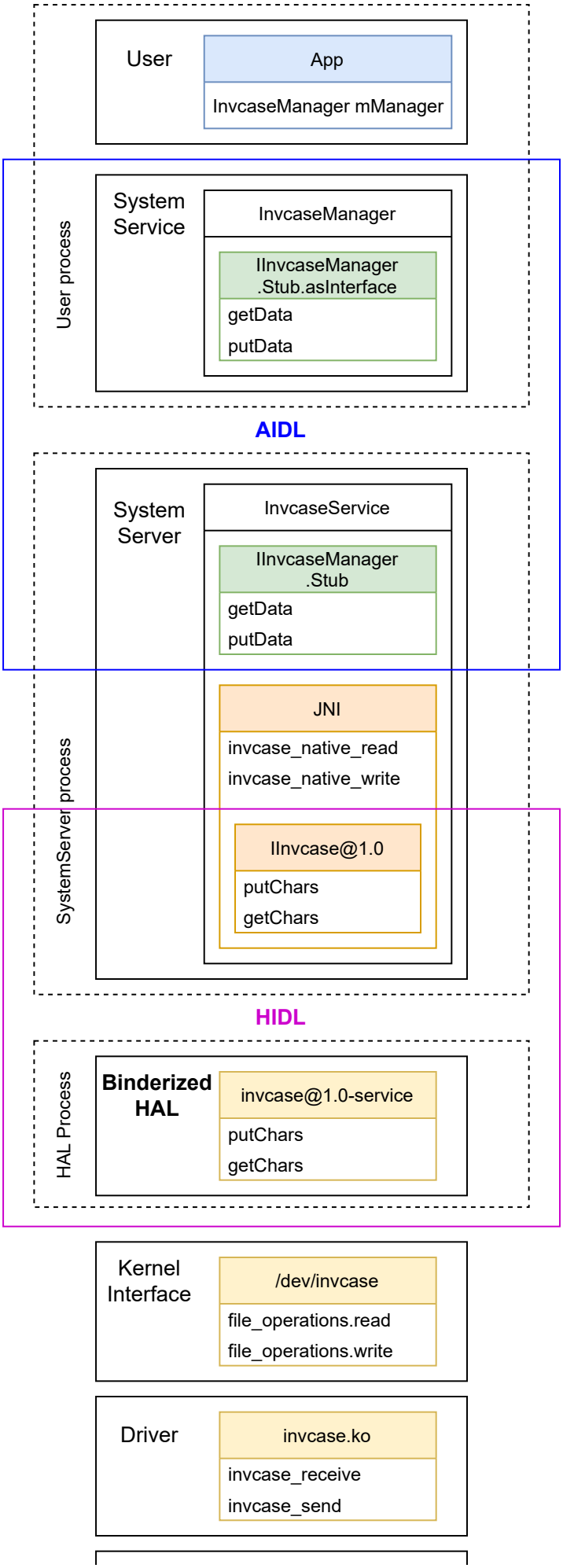
HIDL

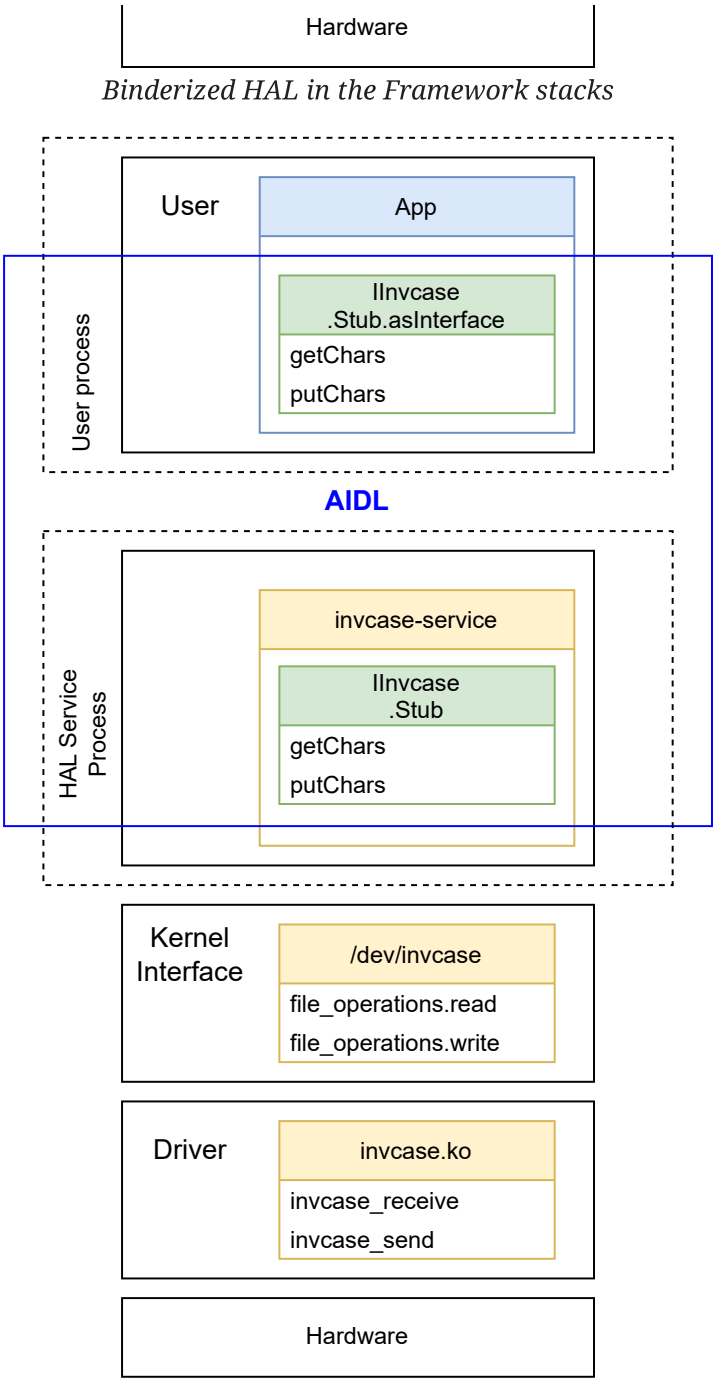
```
// This file is autogenerated by hidl-gen -Landroidbp.

hidl_interface {
    name: "android.hardware.invcase@1.0",
    root: "android.hardware",
    vndk: {
        enabled: true,
    },
    srcs: [
        "IInvcase.hal",
    ],
    interfaces: [
        "android.hidl.base@1.0",
    ],
    gen_java: false,
}
```

Framework Stacks

Differences between *Binderized HIDL for HAL* and *AIDL for HAL*:


User
App
InvcaseManager mManager
User process
System Service
InvcaseManager
IInvcaseManager .Stub.asInterface
getData
putData
AIDL
System Server
InvcaseService
IInvcaseManager .Stub
getData
putData
JNI
invcase_native_read
invcase_native_write
IInvcase@1.0
putChars
getChars
SystemServer process
HIDL
HAL Process
Binderized HAL
invcase@1.0-service
putChars
getChars
Kernel Interface
/dev/invcase
file_operations.read
file_operations.write
Driver
invcase.ko
invcase_receive
invcase_send



Binderized HAL in the Framework stacks

AIDL HAL in the Framework stacks

Implementation

aidl_hal.zip

Refer to Kernel Module to build and install a module to system. This guide assumes that `invcase` module is loaded as below:

device/generic/goldfish/init.ranchu.rc

```
+ on boot
+     insmod /system/lib/modules/invcase.ko
```

```
+     chown system system /dev/invcase
+     chmod 0600 /dev/invcase
```

Overview

▼ AOSP

▼ build

▼ make

▼ target

▼ product

▼ *base_vendor.mk*

Include new packages

```
+ PRODUCT_PACKAGES += \
+     android.hardware.invcase \
+     android.hardware.invcase-service \
+     Invcase
```

▼ packages

▼ apps

▼ Invcase

▼ src

▼ com

▼ invcase

▼ *Invcase.java*

Bind to **IInvcase** interface

```
import android.hardware.invcase.IInvcase;
class Invcase extends Activity {
    IInvcase invcaseAJ; // AIDL Java
    onCreate() {
        IBinder binder =
ServiceManager.getService(IINVCASE_AIDL_INTERFACE);
        invcaseAJ = IInvcase.Stub.asInterface(binder);
    }
}
```

▼ res

- layout
- mipmap
- values

▼ *AndroidManifest.xml*

Export main activity on Launcher

```
<manifest package="com.invcase" >
    <application>
        <activity
            android:name=".Invcase"
            android:exported="true" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

> `android:exported="true"` is mandatory on Android 12

▼ *Android.bp*

```
android_app {
    name: "Invcase",
    srcs: ["src/**/*.java"],
    platform_apis: true

    static_libs: [
        "android.hardware.invcase-V1-java"
    ]
}
```

> `platform_apis: true`: use System API when do not specify any target platform
> `android.hardware.invcase-V1-java`: Java proxy for the AIDL of HAL, directly callable from User App

▼ hardware

▼ interfaces

▼ invcase

▼ aidl

▼ *Android.bp*

▼ android

▼ hardware

▼ invcase

▼ *IInvcase.aidl*

```
@VintfStability
interface IInvcase {
    String getChars();
    void putChars(in String msg);
}
```

▼ default

▼ *Android.bp*

▼ *Invcase.h*

▼ *Invcase.cpp*

▼ *service.cpp*

▼ compatible_matrices

▼ *compatible_matrix.current.xml*

```
+ <hal format="aidl" optional="true">
+     <name>android.hardware.invcase</name>
+     <version>1</version>
+     <interface>
+         <name>IInvcase</name>
+         <instance>default</instance>
+     </interface>
+ </hal>
```

Define HAL Interface

Create a new AIDL file in the folder `hardware/interfaces/invcase/aidl`:

hardware/interfaces/invcase/aidl/android/hardware/invcase/**IInvcase**.aidl

```
package android.hardware.invcase;

@VintfStability
interface IInvcase {
    String getChars();
    void putChars(in String msg);
}
```

Configure build

- Select the backend: We will use NDK (as recommended), so declare the CPP backend as `false`.
- Set `vendor: true` and remove `vendor_available` because this is a custom vendor HAL
- Remove `vndk` section, so this HAL is located in `/vendor` only

VNDK

- VNDK Build Example
- VNDK Extension

```
aidl_interface {
    name: "android.hardware.invcase",
    vendor: true,
    srcs: ["android/hardware/invcase/*.aidl"],
    stability: "vintf",
```

```
    owner: "vqtrong",
    backend: {
        cpp: {
            enabled: false,
        },
        java: {
            sdk_version: "module_current",
        },
    },
}
```

At this time, if try to build the module with:

```
mmm hardware/interfaces/invcase/
```

you will get error about the API missing:

```
API dump for the current version of AIDL interface android.hardware.invcase does
not exist.
Run `m android.hardware.invcase-update-api`, or add `unstable: true` to the
build rule for the interface if it does not need to be versioned.
```

We need to **freeze the API** by running:

```
m android.hardware.invcase-update-api
```

> **versioning**
>
> Since there is no version 1, the current version is version 1, but once we perform a change to the API, we will upgrade the aidl. The current folder needs to be renamed to 1 and a new current folder needs to be created representing version 2.

Ok, build it again:

```
mmm hardware/interfaces/invcase/
```

Then **include the module to the system**:

build/make/target/product/base_vendor.mk

```
PRODUCT_PACKAGES += \
    android.hardware.invcase \
```

Implement HAL

We will use the `ndk_platfrom` library, therefore, let check the generated code for `ndk_platform`.

```
cd
out/soong/.intermediates/hardware/interfaces/invcase/aidl/android.hardware.invcase-V1-ndk_platform-source
```

```
find .
```

```
.
./gen
./gen/timestamp
./gen/include
./gen/include/aidl
./gen/include/aidl/android
./gen/include/aidl/android/hardware
./gen/include/aidl/android/hardware/invcase
./gen/include/aidl/android/hardware/invcase/BpInvcase.h
./gen/include/aidl/android/hardware/invcase/IInvcase.h
./gen/include/aidl/android/hardware/invcase/BnInvcase.h
./gen/android
./gen/android/hardware
./gen/android/hardware/invcase
./gen/android/hardware/invcase/IInvcase.cpp.d
./gen/android/hardware/invcase/IInvcase.cpp
```

Our interface APIs are converted to APIs as below:

IInvcase.h

```
virtual ::ndk::ScopedAStatus getChars(std::string* _aidl_return) = 0;
virtual ::ndk::ScopedAStatus putChars(const std::string& in_msg) = 0;
```

They are virtual functions and then need to be defined.

Header file

hardware/interfaces/invcase/aidl/default/Invcase.h

```
#pragma once

#include <aidl/android/hardware/invcase/BnInvcase.h>

namespace aidl {
namespace android {
namespace hardware {
namespace invcase {

class Invcase : public BnInvcase {
```

```
    public:
        //String getChars();
        ndk::ScopedAStatus getChars(std::string* _aidl_return);
        //void putChars(in String msg);
        ndk::ScopedAStatus putChars(const std::string& in_msg);
};

}  // namespace invcase
}  // namespace hardware
}  // namespace android
}  // namespace aidl
```

Implementation

hardware/interfaces/invcase/aidl/default/Invcase.cpp

```
#define LOG_TAG "Invcase"

#include <utils/Log.h>
#include <iostream>
#include <fstream>
#include "Invcase.h"

namespace aidl {
namespace android {
namespace hardware {
namespace invcase {

//String getChars();
ndk::ScopedAStatus Invcase::getChars(std::string* _aidl_return) {
    std::ifstream invcase_dev;
    invcase_dev.open("/dev/invcase");
    if(invcase_dev.good()) {
        std::string line;
        invcase_dev >> line;
        ALOGD("Invcase service: getChars: %s", line.c_str());
        *_aidl_return =  line;
    } else {
        ALOGE("getChars: can not open /dev/invcase");
        return ndk::ScopedAStatus::fromServiceSpecificError(-1);
    }
    return ndk::ScopedAStatus::ok();
}

//void putChars(in String msg);
ndk::ScopedAStatus Invcase::putChars(const std::string& in_msg) {
    std::ofstream invcase_dev;
    invcase_dev.open ("/dev/invcase");
    if(invcase_dev.good()) {
        invcase_dev << in_msg;
        ALOGD("Invcase service: putChars: %s", in_msg.c_str());
    } else {
        ALOGE("putChars: can not open /dev/invcase");
        return ndk::ScopedAStatus::fromServiceSpecificError(-1);
    }
```

```
    return ndk::ScopedAStatus::ok();
}

}  // namespace invcase
}  // namespace hardware
}  // namespace android
}  // namespace aidl
```

Implement HAL Service

The HAL service will run in its own process, like in HIDL.

Create a new folder for implementation in `hardware/interfaces/invcase/aidl/default`:

Service implementation

hardware/interfaces/invcase/aidl/default/service.cpp

```
#define LOG_TAG "Invcase"

#include <android-base/logging.h>
#include <android/binder_manager.h>
#include <android/binder_process.h>
#include <binder/ProcessState.h>
#include <binder/IServiceManager.h>
#include "Invcase.h"

using aidl::android::hardware::invcase::Invcase;
using std::string_literals::operator""s;

void logd(std::string msg) {
    std::cout << msg << std::endl;
    ALOGD("%s", msg.c_str());
}

void loge(std::string msg) {
    std::cout << msg << std::endl;
    ALOGE("%s", msg.c_str());
}

int main() {
    // Enable vndbinder to allow vendor-to-venfor binder call
    android::ProcessState::initWithDriver("/dev/vndbinder");

    ABinderProcess_setThreadPoolMaxThreadCount(0);
    ABinderProcess_startThreadPool();

    std::shared_ptr<Invcase> invcase = ndk::SharedRefBase::make<Invcase>();
    const std::string name = Invcase::descriptor + "/default"s;

    if (invcase != nullptr) {
        if(AServiceManager_addService(invcase->asBinder().get(),
```

```
name.c_str()) != STATUS_OK) {
            loge("Failed to register IInvcase service");
            return -1;
        }
    } else {
        loge("Failed to get IInvcase instance");
        return -1;
    }

    logd("IInvcase service starts to join service pool");
    ABinderProcess_joinThreadPool();

    return EXIT_FAILURE;  // should not reached
}
```

> **vndbinder**
>
> Normally, vendor processes don't open the binder driver directly and instead link against the `libbinder` userspace library, which opens the binder driver. Adding a method for `::android::ProcessState()` selects the binder driver for `libbinder`. Vendor processes should call this method before calling into `ProcessState`, `IPCThreadState`, or before making any binder calls in general. To use, place the following call after the `main()` of a vendor process (client and server):
>
> ```
> #include <binder/ProcessState.h>
> int main() {
>     android::ProcessState::initWithDriver("/dev/vndbinder");
> }
> ```
>
> **Thread management**
>
> Every instance of libbinder in a process maintains one threadpool. For most use cases, this should be exactly one threadpool, shared across all backends. The only exception to this is when vendor code might load another copy of libbinder to talk to /dev/vndbinder. Since this is on a separate binder node, the threadpool isn't shared.
>
> In the NDK backend:
>
> ```
> bool success = ABinderProcess_setThreadPoolMaxThreadCount(0);
> ABinderProcess_startThreadPool();
> ABinderProcess_joinThreadPool();
> ```

Build Service

Similar to the HIDL module, we will create a `cc_binary` module in the`Android.bp`.

AIDL has three different backends: Java, NDK, CPP. To use Stable AIDL, you must always use the system copy of `libbinder` at `system/lib*/libbinder.so` and talk on `/dev/binder`. For code on the vendor image, this means that `libbinder` (from the VNDK) cannot be used: this library has an unstable C++ API and unstable internals. Instead, native vendor code must use the NDK backend of AIDL, link against `libbinder_ndk` (which is backed by system `libbinder.so`), and link against the `-ndk_platform` libraries created by `aidl_interface` entries.

hardware/interfaces/invcase/aidl/default/Android.bp

```
cc_binary {
    name: "android.hardware.invcase-service",
    vendor: true,
    relative_install_path: "hw",
    init_rc: ["android.hardware.invcase-service.rc"],
    vintf_fragments: ["android.hardware.invcase-service.xml"],

    srcs: [
        "Invcase.cpp",
        "service.cpp",
    ],

    cflags: [
        "-Wall",
        "-Werror",
    ],

    shared_libs: [
        "libbase",
        "liblog",
        "libhardware",
        "libbinder_ndk",
        "libbinder",
        "libutils",
        "android.hardware.invcase-V1-ndk_platform",
    ],
}
```

Then include the service to the system:

build/make/target/product/base_vendor.mk

```
PRODUCT_PACKAGES += \
    android.hardware.invcase \
    android.hardware.invcase-service \
```

Run Service

We need to define the service with the `init` process, so it can start whenever the `hal` class is started. To do this, we will create a new `android.hardware.invcase-service.rc`:

hardware/interfaces/invcase/aidl/default/android.hardware.invcase-service.rc

```
service android.hardware.invcase-service
/vendor/bin/hw/android.hardware.invcase-service
        interface aidl android.hardware.invcase.IInvcase/default
        class hal
        user system
        group system
```

Expose AIDL Interface

A new VINTF AIDL object should be declared as below:

hardware/interfaces/invcase/aidl/default/android.hardware.invcase-service.xml

```
<manifest version="1.0" type="device">
    <hal format="aidl">
        <name>android.hardware.invcase</name>
        <version>1</version>
        <fqname>IInvcase/default</fqname>
    </hal>
</manifest>
```

If this is a new package, add it to the latest framework compatibility matrix. If no interface should be added to the framework compatibility matrix (e.g. types-only package), add it to the exempt list in `libvintf_fcm_exclude`.

hardware/interfaces/compatibility_matrices/compatibility_matrix.6.xml
hardware/interfaces/compatibility_matrices/compatibility_matrix.current.xml

```
<hal format="aidl" optional="true">
    <name>android.hardware.invcase</name>
    <version>1</version>
    <interface>
        <name>IInvcase</name>
        <instance>default</instance>
    </interface>
</hal>
```

Define SELinux Policy for HAL service

To make the service run at boot, HAL service needs to be registered to system under a security policy.

Declare new type

system/sepolicy/prebuilts/api/32.0/public/hwservice.te
system/sepolicy/public/hwservice.te

```
type hal_invcase_hwservice, hwservice_manager_type;
```

Set compatibility

Ignore in API 31, which also ignore in lower API:

system/sepolicy/prebuilts/api/32.0/private/compat/31.0/31.0.ignore.cil
system/sepolicy/private/compat/31.0/31.0.ignore.cil

```
(type new_objects)
(typeattribute new_objects)
(typeattributeset new_objects
    ( new_objects
        hal_invcase_hwservice
    )
)
```

Add service path

Add a new label in the:

system/sepolicy/vendor/file_contexts

```
/(vendor|system/vendor)/bin/hw/android\.hardware\.invcase-service
u:object_r:hal_invcase_service_exec:s0
```

Set service context interface:

system/sepolicy/prebuilts/api/32.0/private/hwservice_contexts
system/sepolicy/private/hwservice_contexts

```
android.hardware.invcase::IInvcase
u:object_r:hal_invcase_hwservice:s0
```

Declare attribute:

system/sepolicy/prebuilts/api/32.0/public/attributes
system/sepolicy/public/attributes

```
hal_attribute(invcase);
```

this is macro for adding below attributes:

```
attribute hal_invcase;
attribute hal_invcase_client;
attribute hal_invcase_server;mon
```

Define default domain:

system/sepolicy/vendor/hal_invcase_service.te

```
type hal_invcase_service, domain;
hal_server_domain(hal_invcase_service, hal_invcase)
type hal_invcase_service_exec, exec_type, vendor_file_type,
vendor_file_type, file_type;
init_daemon_domain(hal_invcase_service)
```

Set binder policy:

system/sepolicy/prebuilts/api/32.0/public/hal_invcase.te
system/sepolicy/public/hal_invcase.te

```
binder_call(hal_invcase_client, hal_invcase_server)
binder_call(hal_invcase_server, hal_invcase_client)
hal_attribute_hwservice(hal_invcase, hal_invcase_hwservice)
```

Declare system_server as client of HAL service:

system/sepolicy/prebuilts/api/29.0/private/system_server.te
system/sepolicy/private/system_server.te

```
hal_client_domain(system_server, hal_invcase)
```

Deliver HAL module

VNDK

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.

Refer to https://source.android.com/devices/architecture/vndk/build-system.

Include HAL service and the test app to the `PRODUCT_PACKAGES`:

build/target/product/base_vendor.mk

```
+ PRODUCT_PACKAGES += \
+     android.hardware.invcase-service \
```

This will include below files to system:

```
/vendor/lib/hw/android.hardware.invcase-service
```

User App

The User App will be very simple to test the hardware. It contains an `EditText` to get user input, a `Button` to execute commands, and a `TextView` to display the result.

Implement the User App

- Use `getSystemService(Context.INVCASE_SERVICE)` to obtain the instance of `InvcaseManager`
- Call to hardware through the `InvcaseManager` APIs

packages/apps/Invcase/src/com/invcase/Invcase.java

```
package com.invcase;

import android.content.Context;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.util.Log;
import android.os.ServiceManager;
import android.os.IBinder;
import android.hardware.invcase.IInvcase;

public class Invcase extends Activity {
    private static final String TAG = "Invcase";
    private static final String IINVCASE_AIDL_INTERFACE =
"android.hardware.invcase.IInvcase/default";
    private static IInvcase invcaseAJ; // AIDL Java

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btn = (Button)findViewById(R.id.button);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                EditText editText = (EditText)findViewById(R.id.editText);
                String txt = editText.getText().toString();
                Log.d(TAG, "App: request= " + txt);

                if(invcaseAJ != null) {
                    try {
                        invcaseAJ.putChars(txt);
                    } catch (android.os.RemoteException e) {
                        Log.e(TAG, "IInvcase-AIDL error", e);
                    }
```

```
                }

                String ret = "";

                if(invcaseAJ != null) {
                    try {
                        ret = invcaseAJ.getChars();
                    } catch (android.os.RemoteException e) {
                        Log.e(TAG, "IInvcase-AIDL error", e);
                    }
                }

                Log.d(TAG, "App: get= " + ret);

                TextView tv = (TextView)findViewById(R.id.textView);
                tv.setText(ret);
            }
        });

        IBinder binder = ServiceManager.getService(IINVCASE_AIDL_INTERFACE);
        if (binder == null) {
            Log.e(TAG, "Getting " + IINVCASE_AIDL_INTERFACE + " service
daemon binder failed!");
        } else {
            invcaseAJ = IInvcase.Stub.asInterface(binder);
            if (invcaseAJ == null) {
                Log.e(TAG, "Getting IInvcase AIDL daemon interface
failed!");
            } else {
                Log.d(TAG, "IInvcase AIDL daemon interface is binded!");
            }
        }
    }
}
```

Add User App to the Launcher

packages/apps/Invcase/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.invcase" >
    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".Invcase"
            android:exported="true"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
    </application>
</manifest>
```

> on Android 12, must use `android:exported="true"`

Build User App

packages/apps/Invcase/Android.bp

```
android_app {
    name: "Invcase",
    platform_apis: true,
    srcs: [
        "src/**/*.java"
    ],
    static_libs: [
        "android.hardware.invcase-V1-java"
    ]
}
```

Add User App to system packages:

build/target/product/base_vendor.mk

```
+ PRODUCT_PACKAGES += \
+     Invcase
```

Permission

The device `/dev/invcase` is created at boot with root permission.

The HAL Library is loaded when JNI Wrapper for Invcase Service is run, therefore, HAL code will run with `system` permission which attaches to the `system_server` process.

The Android Init Language uses `init*.rc` files to automatically do some actions when a condition is met. For the target hardware, Android Init will use `init.<hardware>.rc` file. On Emulator, it is `init.ranchu.rc`.

Add below lines to change the owner and permission on the `/dev/invcase` at boot:

device/generic/goldfish/init.ranchu.rc

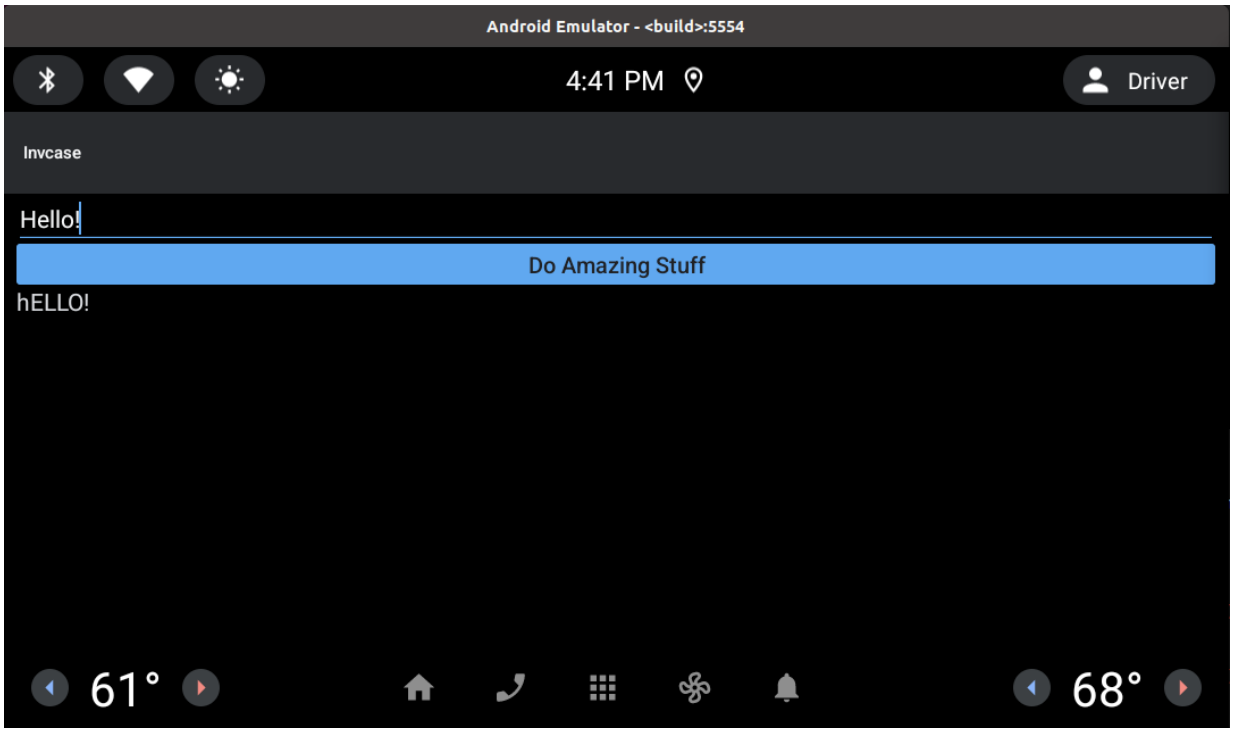
```
+ on boot
+     chown system system /dev/invcase
+     chmod 0600 /dev/invcase
```

Build and Run

The Invcase Manager exports new Service APIs, therefore, need to rebuild the list of system APIs.

```
m all -j$(nproc)
```

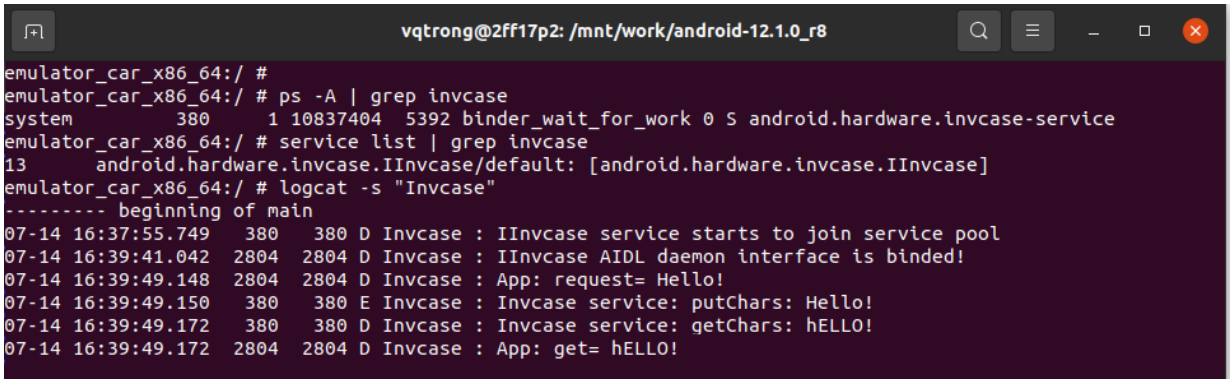
Run the Emulator and run Invcase app:



The User Test application

Start the Logcat to see debug lines:

```
logcat -s Invcase
```



Logcat shows Invcase calls

There are 2 processes:

- The HAL process runs:
  - Host the HAL implementation

- The user app process does below things:
  - Bind to HAL process through AIDL proxy (through VNDBinder)