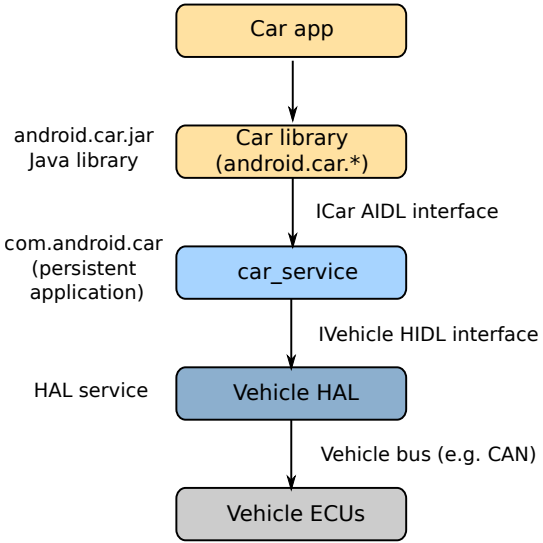
Extending the Vehicle HAL with vendor properties

Chris Simmonds, 2net

Architecture of Android Automotive



The Android Hardware Abstraction Layer

- The Hardware Abstraction Layer (HAL) sits between Android and hardware
- Divided into c. 50 interfaces
- Interfaces are written in HIDL(*) (deprecated) or Stable AIDL(**)
- Most HALs implemented as a daemon (background) process

(*) HIDL = Hardware Interface Definition Language
(**) AIDL = Android Interface Definition Language

The Vehicle HAL

- The Vehicle HAL (VHAL) mediates between Android and the vehicle
- Allows apps and the Android framework to
  - Monitor variables, e.g. speed
  - Control variables, e.g. side window position
- Vehicle variables are represented as **vehicle properties**
- Properties have names such as `PERF_VEHICLE_SPEED` and `WINDOW_POS`

The VHAL IVehicle

- The VHAL is defined as a HIDL interface in
  `hardware/interfaces/automotive/vehicle/2.0`

```
IVehicle.hal                 The functions, get(), set(), subscribe(), etc
IVehicleCallback.hal         Callback when a property changes
types.hal                    Types used by IVehicle and IVehicleCallback
```

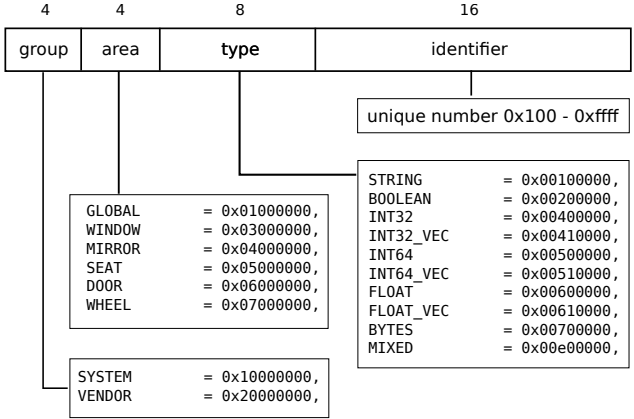
System and vendor vehicle Properties

- The VHAL defines two groups of properties:
- **SYSTEM**: c. 150 properties defined in `types.hal`
- **VENDOR**: defined by OEM, functions defined as needed

The SYSTEM properties and all the types associated with them are defined in file `hardware/interfaces/automotive/vehicle/2.0/types.hal`

Property identifier

A vehicle property is identified by a 32-bit number with this format:



System Property Identifiers

- **System** property identifiers are marked with `VehiclePropertyGroup:SYSTEM`
- In S/12, `types.hal` defines over 150 of them, for example:

```
enum VehicleProperty: int32_t {
[...]
/**
 * Speed of the vehicle
 *
 * @change_mode VehiclePropertyChangeMode:CONTINUOUS
 * @access VehiclePropertyAccess:READ
 * @unit VehicleUnit:METER_PER_SEC
 */
PERF_VEHICLE_SPEED = (
    0x0207
    | VehiclePropertyGroup:SYSTEM
    | VehiclePropertyType:FLOAT
    | VehicleArea:GLOBAL),
[...]
```

Adding a new HIDL root

- The vendor properties will be defined in an extension to `types.hal`
- HAL extensions are usually put in `hardware/[company name]`, e.g. `hardware/google`
- Need a HIDL root to map HIDL package name to a path name

```
$ croot
$ mkdir hardware/example
```

Create `hardware/example/Android.bp`

```
hidl_package_root {
    name: "vendor.example",
}
```

Now HIDL packages starting `vendor.example` will be found in `hardware/example`

Adding a HIDL extension

- The new HIDL package, `vendor.example.automotive.vehicle@2.0`, extends `android.hardware.automotive.vehicle@2.0`
- The pathname to the extended HAL is `hardware/example/automotive/vehicle/2.0`
- Create types.h and define a `VehicleProperty` that extends `android.hardware.automotive.vehicle@2.0::VehicleProperty`

```
package vendor.example.automotive.vehicle@2.0;

import android.hardware.automotive.vehicle@2.0::VehicleProperty;
enum VehicleProperty : android.hardware.automotive.vehicle@2.0::VehicleProperty {
[...]
```

`hardware/example/automotive/vehicle/2.0/types.h`

Adding a HIDL extension

Define vendor property types, importing additional types from the base HAL

```
package vendor.example.automotive.vehicle@2.0;

import android.hardware.automotive.vehicle@2.0::VehicleProperty;
import android.hardware.automotive.vehicle@2.0::VehiclePropertyGroup;
import android.hardware.automotive.vehicle@2.0::VehiclePropertyType;
import android.hardware.automotive.vehicle@2.0::VehicleArea;
enum VehicleProperty : android.hardware.automotive.vehicle@2.0::VehicleProperty {
    VENDOR_EXAMPLE = (
        0x1001
        | VehiclePropertyGroup:VENDOR
        | VehiclePropertyType:INT32
        | VehicleArea:GLOBAL),
};
```

`hardware/example/automotive/vehicle/2.0/types.h`

Create an Android.bp

Create an Android.bp to build the interface:

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

hidl_interface {
    name: "vendor.example.automotive.vehicle@2.0",
    root: "vendor.example",
    system_ext_specific: true,
    srcs: [
        "types.hal",
    ],
    interfaces: [
        "android.hardware.automotive.vehicle@2.0",
        "android.hidl.base@1.0",
    ],
    gen_java: true,
}
```

hardware/example/automotive/vehicle/2.0/Android.bp

Android modules generated

After building (m) you will see several new Android modules:

```
$ allmod | grep vendor.example.automotive
[...]
vendor.example.automotive.vehicle-V2.0-java      <--- Java interface
[...]
vendor.example.automotive.vehicle@2.0            <--- C++ interface
[...]
```

The generated C++ code is in
`out/soong/.intermediates/hardware/example/automotive/vehicle/2.0/vendor.example.automotive.vehicle@2.0_genc++_headers/gen/vendor/example/automotive/vehicle/2.0/types.h`

The generated Java code is in
`out/soong/.intermediates/hardware/example/automotive/vehicle/2.0/vendor.example.automotive.vehicle-V2.0-java_gen_java/gen/srcs/vendor/example/automotive/vehicle/V2_0/VehicleProperty.java`

Initializing vendor properties

We need a library where the property configuration and initial values are defined

`vhal-sirius.cpp`

```
// Based on hardware/interfaces/automotive/vehicle/2.0/default/impl/vhal_v2_0/VehicleHalServer.cpp

const ConfigDeclaration kVehicleProperties[]{
    {.config =
        {
            .prop = static_cast < int32_t > (vendor::example::automotive::vehicle::V2_0::VehicleProperty::VENDOR_EXAMPLE),
            .access = VehiclePropertyAccess::READ_WRITE,
            .changeMode = VehiclePropertyChangeMode::ON_CHANGE,
         },
    .initialValue = {.int32Values = {0}},
    },
};

void vhalSiriusInit(VehiclePropertyStore* store)
{
[...]
}
```

Initializing vendor properties

vhal-sirius.h

```
/*
 * Iniitialize the property store for sirius/marvin
 */
using namespace android::hardware::automotive::vehicle::V2_0;
void vhalSiriusInit(VehiclePropertyStore* store);
```

Initializing vendor properties

Android.bp

```
cc_library_static{
    name: "vhal-sirius",
    vendor: true,
    srcs: ["vhal-sirius.cpp"],
    export_include_dirs: ["."],

    // for VehiclePropertyStore.h and log/log.h
    header_libs: ["vhal_v2_0_common_headers", "liblog_headers"],

    shared_libs: [
        // For android/hardware/automotive/vehicle/2.0/IVehicle.h
        "android.hardware.automotive.vehicle@2.0",

        // For vendor/example/automotive/vehicle/2.0/types.h
        "vendor.example.automotive.vehicle@2.0",
    ],
}
```

Initializing vendor properties

We need to call the library from the default VHAL

`hardware/interfaces/automotive/vehicle/2.0/default/VehicleService.cpp`

```
[...]
#include <vhal_v2_0/VehicleHalManager.h>

#include <vhal-sirius.h>                                        <--- new code

using namespace android;
[...]
int main(int /* argc */, char* /* argv */ []) {
[...]
    auto emulator = std::make_unique<impl::VehicleEmulator>(hal.get());
    vhalSiriusInit(store.get());                                <---  new code
    auto service = std::make_unique<VehicleHalManager>(hal.get());
```

Initializing vendor properties

Need to link with our vhal-sirius library

`hardware/interfaces/automotive/vehicle/2.0/default/Android.bp`

```
[...]
cc_binary {
    name: "android.hardware.automotive.vehicle@2.0-service",
    defaults: ["vhal_v2_0_target_defaults"],
    vintf_fragments: [
        "android.hardware.automotive.vehicle@2.0-service.xml",
    ],
    init_rc: ["android.hardware.automotive.vehicle@2.0-service.rc"],
    vendor: true,
    relative_install_path: "hw",
    srcs: ["VehicleService.cpp"],
    shared_libs: [
        "libbase",
        "libjsoncpp",
        "libprotobuf-cpp-lite",
    ],
    static_libs: [
        "android.hardware.automotive.vehicle@2.0-manager-lib",
        "android.hardware.automotive.vehicle@2.0-default-impl-lib",
        "android.hardware.automotive.vehicle@2.0-libproto-native",
        "vhal-sirius",                                          <--- new code
    ],
}
[...]
```

Using the C++ interface

```
#include <hidl/LegacySupport.h>
#include <utils/misc.h>
#include <utils/Log.h>
#include <hidl/HidlSupport.h>
#include <hidl/Status.h>
#include <stdio.h>

#include <android/hardware/automotive/vehicle/2.0/IVehicle.h>
#include <vendor/example/automotive/vehicle/2.0/types.h>

sp < IVehicle > mVehicle;

int main(void)
{
    mVehicle = IVehicle::getService();
    StatusCode status = StatusCode::TRY_AGAIN;

    VehiclePropValue mVendor;
    mVendor.prop = static_cast < int32_t >
           (vendor::example::automotive::vehicle::V2_0::VehicleProperty::VENDOR_EXAMPLE);
    mVehicle->get(*pRequestedPropValue,[pRequestedPropValue, &status]
                  (StatusCode s, const VehiclePropValue & v) {
                  status = s; if (s == StatusCode::OK) {
                  *pRequestedPropValue = v;}
                  });
    printf("VENDOR_EXAMPLE: %d\n", mVendor.value.int32Values[0]);
    return 0;
}
```

Using the C++ interface

Android.bp

```
cc_binary {
    name: "vehiclehaltest",
    defaults: ["hidl_defaults"],
    vendor: true,
    srcs: ["vehiclehaltest.cpp"],

    shared_libs: [
        "android.hardware.automotive.vehicle@2.0",
        "vendor.example.automotive.vehicle@2.0",
        "libhidlbase",
        "liblog",
        "libutils",
    ],
}
```

Using the Java interface

ExampleCarActivity.java

```
package com.example.car;
[...]
// Property IDs, including vendor extensions
import vendor.example.automotive.vehicle.V2_0.VehicleProperty;
[...]
public class ExampleCarActivity extends Activity
[...]

    int vendorExample = mPropertyManager.getIntProperty(VehicleProperty.VENDOR_EXAMPLE, 0);
    Log.i(TAG, "Got VENDOR_EXAMPLE" + vendorExample);
[...]
```

Using the Java interface

Android.bp

```
android_app {
    name: "examplecar",

    srcs: ["src/**/*.java"],

    // This is needed to be able to use android.car.permission.CAR_VENDOR_EXTENSION
    privileged: true,
    certificate: "platform",

    platform_apis: true,

    // For VehicleProperty with vendor extensions
    static_libs: [
        "vendor.example.automotive.vehicle-V2.0-java",
    ],
    libs: ["android.car"],
    required: ["allowed_privapp_com.example.car"],
}

prebuilt_etc {
    name: "allowed_privapp_com.example.car",
    sub_dir: "permissions",
    src: "com.example.car.xml",
    filename_from_src: true,
}
```