Workshop Agenda

- Level of standardization (how deep GENIVI should standardize the access to Vehicle API: spec level? Tooling level? Implementation level?)
- Do we need an alternative API to Android SDK for accessing data? (discussion about the real use cases of accessing data for applications)
- Brainstorm on the metalanguage for describing the conversion between 2 specifications (VSS and Android)
- Virtualization => how can we minimize changes to Android (virtio, trout)
- Simulation => how can we simulate HW acceleration in a seamless environment together an Emulated Android
- Multiple microphones usage in Android
- Multiple devices collaboration: device centric vs car system centric.
Level of standardization

VSS vspec
- Metalanguage to describe mapping (which to which, conversion type)

Vehicle signals
- AOSP Binding def: Sig to prop mapping

Tooling to convert from Binding spec to map

Generated map [Key=VSS_id, Value=convert method]

Converter

GENIVI/W3C

OEM

License: CC BY-SA 4.0
Do we need 2nd API? PROS/CONS

Android defined properties + VSS as "vendor property"

VSS format

AOSP format
Replace xlsx to reduce human input

```cpp
/**
 * Tire pressure
 *
 * min/max value indicates tire pressure sensor range. Each tire will have a separate min/max value denoted by its areaConfig.areaId.
 *
 * @change_mode VehiclePropertyChangeMode:CONTINUOUS
 * @access VehiclePropertyAccess:READ
 * @unit VehicleUnit:KILOPASCAL
 */

TIRE_PRESSURE = (0x0309 | VehiclePropertyGroup:SYSTEM | VehiclePropertyType:FLOAT | VehicleArea:WHEEL),
```

Bind Tire.Pressure to TIRE_PRESSURE – vss layer?
Bind actual „entity“ (Row1.Wheel.Left.Tire.Pressure) to VehicleArea – vss layer?
Describe translation between units that are sometimes „tricky“ – vss layer with „mathematical language“?

```javascript
    std::placeholders::_1, VehicleProperty::TIRE_PRESSURE, (int32_t) VehicleAreaWheel::LEFT_FRONT, 1.0f, 0.0f);
```
Challenge

Vehicle.Powertrain.FuelSystem.Level
aospId: VehicleProperty::FUEL_LEVEL
aospArea: VehicleArea::Global
translation:
- complex: "INFO_FUEL_CAPACITY * _VAL_ / 100"

Vehicle.Powertrain.FuelSystem.Level
To VehicleProperty::FUEL_LEVEL
Conversion from % to milliliters – conversion dependent on other signal!

static float getFloat(VehicleHal* vhal, VehicleProperty prop) {
    VehiclePropValue request = VehiclePropValue {
        .prop = toItem(prop),
    };

    StatusCode halStatus;
    auto valPtr = vhal->get(request, &halStatus);
    float val = 0;
    if (valPtr != nullptr) {
        val = valPtr->value.floatValues[0];
    }

    return val;
}

static VehiclePropValue convertFuelLevel(std::string value, VehicleProperty id, int32_t area, float fuelCapacity) {
    VehiclePropValue prop = initializeProp(id, area);
    uint8_t percentage = std::stof(value);
    float milliliters = fuelCapacity * percentage / 100;
    prop.value.floatValues = std::vector<float> { milliliters };

    // TODO error handling
    return prop;
}

conversionMap["Vehicle.Powertrain.FuelSystem.Level"] = std::bind(convertFuelLevel,
    std::placeholders::_1, VehicleProperty::FUEL_LEVEL, toItem(VehicleArea::GLOBAL), getFloat(vhal, INFO_FUEL_CAPACITY));
static VehiclePropValue convertFloat(std::string value, VehicleProperty id, int32_t area, float K, float m) {
    VehiclePropValue prop = initializeProp(id, area);
    float v = std::stof(value);
    prop.value.floatValues = std::vector<float> { v * K + m };
    return prop;
}

static VehiclePropValue convertFuelLevel(std::string value, VehicleProperty id, int32_t area, float fuelCapacity) {
    // COMPLEX!!!!!!!!!!!!!! STUB
}

conversionMap["Vehicle.ADAS.ABS.IsActive"] = std::bind(convertBool,
    std::placeholders::_1, VehicleProperty::ABS_ACTIVE, toInt(VehicleArea::GLOBAL));
conversionMap["Vehicle.Powertrain.CombustionEngine.Engine.EOT"� = std::bind(convertFloat,
    std::placeholders::_1, VehicleProperty::ENGINE_OIL_TEMP, toInt(VehicleArea::GLOBAL), 1.0f, 0.0f);
conversionMap["Vehicle.Powertrain.FuelSystem.Level"] = std::bind(convertFuelLevel,
    std::placeholders::_1, VehicleProperty::FUEL_LEVEL, toInt(VehicleArea::GLOBAL), getFuelCapacity(vhāl));
    std::placeholders::_1, VehicleProperty::TIRE_PRESSURE, (int32_t)VehicleAreaWheel::LEFT_FRONT, 1.0f, 0.0f);
    std::placeholders::_1, VehicleProperty::TIRE_PRESSURE, (int32_t)VehicleAreaWheel::RIGHT_FRONT, 1.0f, 0.0f);
    std::placeholders::_1, VehicleProperty::TIRE_PRESSURE, (int32_t)VehicleAreaWheel::LEFT_REAR, 1.0f, 0.0f);
    std::placeholders::_1, VehicleProperty::TIRE_PRESSURE, (int32_t)VehicleAreaWheel::RIGHT_REAR, 1.0f, 0.0f);
conversionMap["Vehicle.Speed"] = std::bind(convertFloat,
    std::placeholders::_1, VehicleProperty::PERF_VEHICLE_SPEED, toInt(VehicleArea::GLOBAL), 1.0f / 3.6f, 0.0f);
BACKUP Slides
"VSS feeder"

Translator

Generated map
[Key=VSS_id,
Value=convertmethod]

Vehicle HAL

AOSP VehicleHal
common lib

Car Service

APP

signal in vss

signal in AOSP format

signal in AOSP format

License: CC BY-SA 4.0
**Tire pressure**

* min/max value indicates tire pressure sensor range. Each tire will have a separate min/max value denoted by its `areaConfig.areaId`.

* @change_mode VehiclePropertyChangeMode:CONTINUOUS
* @access VehiclePropertyAccess:READ
* @unit VehicleUnit:KILOPASCAL

```plaintext
TIRE_PRESSURE = (
    0x0309 |
    VehiclePropertyGroup:SYSTEM |
    VehiclePropertyType:FLOAT |
    VehicleArea:WHEEL),
```

### From vspec

- **Tire**
  - type: branch
description: Tire signals for wheel

- **Tire.Pressure**
  - datatype: uint8
type: sensor
unit: kPa
description: Tire pressure in kilo-Pascal

### From types.hal

- **TIRE_PRESSURE**
  - out=009
VehiclePropertyGroup:SYSTEM
VehiclePropertyType:FLOAT
VehicleArea:WHEEL,