CAR API, HAL, ANDROID VHAL

COMMON VEHICLE DATA APIS

Jan Kubovy, BMW

COVESA

Accelerating the future of connected vehicles

How to get Car Data to the Apps

- **Full custom solution**
  - fast API changes, high maintenance, proprietary, unmanaged API, no backward compatibility, apps prepare data, high usage requirements, no-3rd-parties
- **Vendor API**
  - fast API changes, high maintenance, proprietary, managed API, possible backward compatibility, common API layer may prepare at least some data, high usage requirements, no-3rd-parties
- **Standard API**
  - slower API changes, lower maintenance, standard, managed API, mostly backward compatibility, API layer prepares data, lower usage requirements, 3rd-parties difficult (SDK, Appstore, License)
- **AOSP VHAL with Vendor Properties**
  - slow API changes, low maintenance, standard, managed API, partially backward compatibility, API layer prepares data, lower usage requirements, no-3rd-parties
- **AOSP VHAL with Standard Properties**
  - slow API changes, low maintenance, standard, managed API, backward compatibility, API layer prepares data, low usage requirements, 3rd-parties

Android HAL & VHAL

- **Android HAL (Hardware Abstraction Layer)**
  - **standard interface** between the Android framework and **device-specific hardware**, allowing the Android system to communicate with hardware components via vendor-specific implementations (i.e. driver)

- **Android VHAL (Vehicle Hardware Abstraction Layer)**
  - **specialized HAL** in Android Automotive, enabling communication between the Android Automotive framework and the **vehicle's hardware**, such as sensors, actuators, and control systems (also a driver but with limited options to implement)
  - employs **VHAL properties** - data points or control interfaces representing various vehicle attributes or functions.

Android VHAL properties

- is identified by a unique **ID** (**`0xGATTDDDD`**)
  - **G**roup: 1 nibble
  - **A**rea: 1 nibble
  - **T**ype 1 byte
  - i**D**entifier: 2 bytes - as sequence (max 65535!)
    - Google starts at 0x0100 (VIN)
    - VSS Mapper* starts at 0x8000
- has associated metadata
- access type (read, write)
- update frequency

```
enum VehiclePropertyGroup {
    SYSTEM     = 0x10000000,
    VENDOR     = 0x20000000,
    BACKPORTED = 0x30000000,
    MASK       = 0xf0000000,
}
```

```
enum VehiclePropertyType {
    STRING     = 0x00100000,
    BOOLEAN    = 0x00200000,
    INT32      = 0x00400000,
    INT32_VEC  = 0x00410000,
    INT64      = 0x00500000,
    INT64_VEC  = 0x00510000,
    FLOAT      = 0x00600000,
    FLOAT_VEC  = 0x00610000,
    BYTES      = 0x00700000,
    MIXED      = 0x00e00000,
    MASK       = 0x00ff0000,
}
```

```
/**
 * Declares all vehicle properties. VehicleProperty has a bitwise structure.
 * Each property must have:
 *  - a unique id from range 0x0100 - 0xffff
 *  - associated data type using VehiclePropertyType
 *  - property group (VehiclePropertyGroup)
 *  - vehicle area (VehicleArea)
 *
 * Vendors are allowed to extend this enum with their own properties. In this
 * case they must use VehiclePropertyGroup:VENDOR flag when the property is
 * declared.
 *
 * When a property's status field is not set to AVAILABLE:
 *  - IVehicle#set may return StatusCode::NOT_AVAILABLE.
 *  - IVehicle#get is not guaranteed to work.
 *
 * Properties set to values out of range must be ignored and no action taken
 * in response to such ill formed requests.
 */
```

Standard and Vendor properties differ by 2 bits

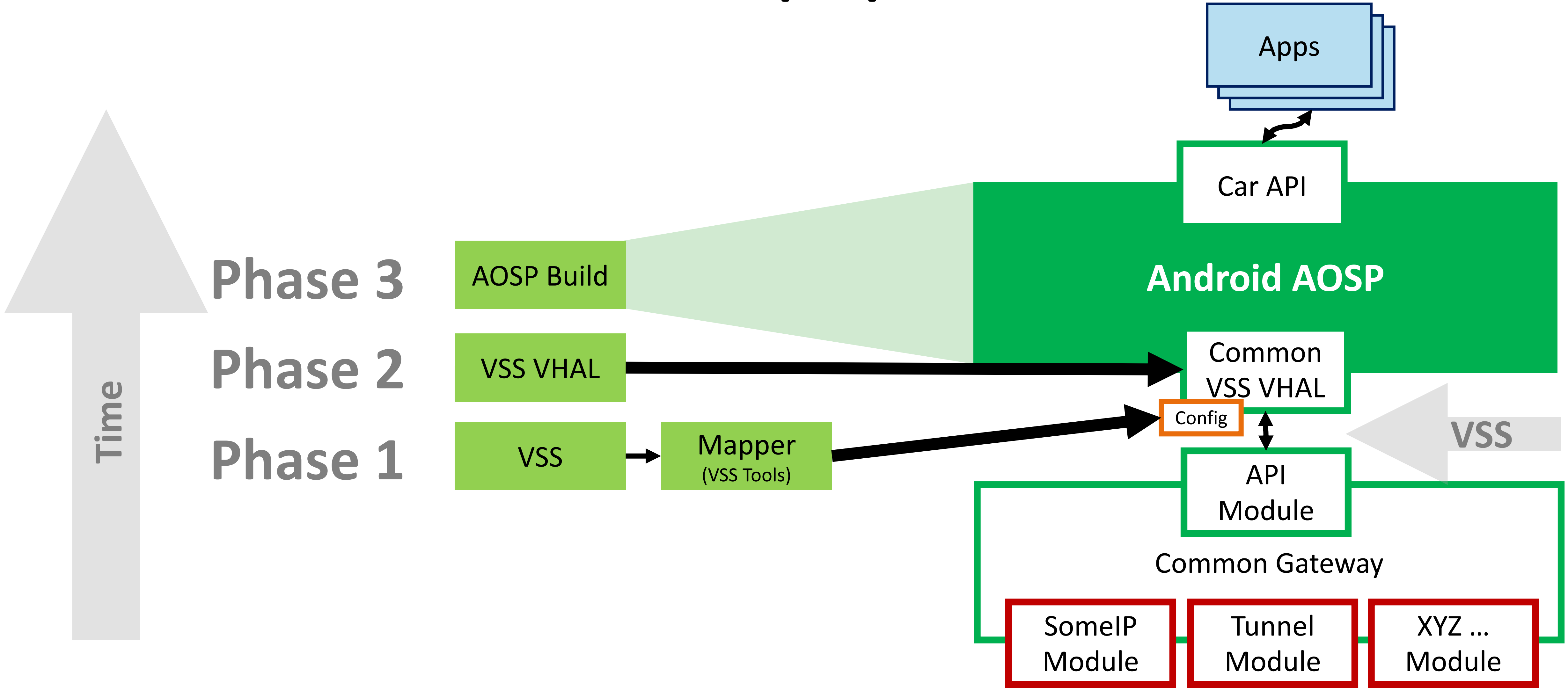
Source: https://cs.android.com/android/platform/superproject/main/+/main:hardware/interfaces/automotive/vehicle/aidl_property/android/hardware/automotive/vehicle/VehicleProperty.aidl

Android Standard vs Vendor Properties

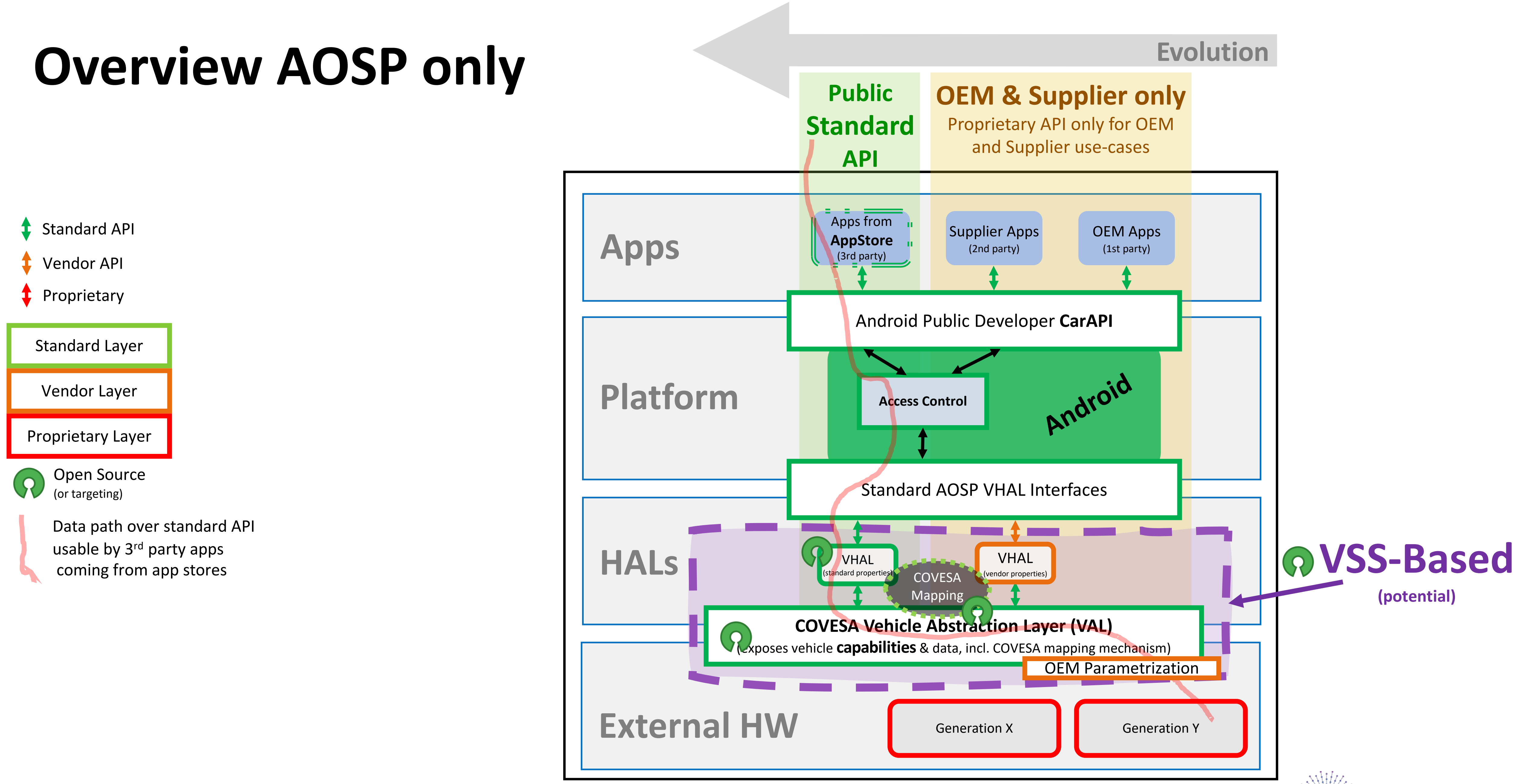
- **Standard Properties**
  - **predefined and standardized** by Android Automotive
  - to ensure **interoperability** across different vehicles
  - covering **only commonly used vehicle data** like speed and fuel level
- **Vendor Properties**
  - **custom, vendor-specific** extensions defined by manufacturers
  - to **expose additional vehicle data** or features not covered by the standard properties
  - **not standardized across vehicles and OEMs**, limiting interoperability and requiring tailoring apps for specific manufacturers
  - **accessible only to system-level apps** or those with special permissions, reducing their utility for general app developers
  - future updates to the Android Automotive framework may not support certain vendor-specific implementations

The **Android Car API** provides developers with a framework to build apps for Android Automotive, enabling them to **access vehicle-related data**. This data is supplied through the **Vehicle HAL (VHAL)**, which acts as the bridge between the Android Automotive framework and the **underlying vehicle hardware**, <u>translating API requests into hardware-specific interactions</u>.

Common standard VSS VHAL proposal



Overview AOSP only



Evolution
Public Standard API
OEM & Supplier only
Proprietary API only for OEM and Supplier use-cases
Apps
Apps from AppStore (3rd party)
Supplier Apps (2nd party)
OEM Apps (1st party)
Android Public Developer CarAPI
Platform
Access Control
Android
Standard AOSP VHAL Interfaces
HALs
VHAL (standard properties)
COVESA Mapping
VHAL (vendor properties)
COVESA Vehicle Abstraction Layer (VAL)
(exposes vehicle capabilities & data, incl. COVESA mapping mechanism)
OEM Parametrization
External HW
Generation X
Generation Y
VSS-Based
(potential)
Standard API
Vendor API
Proprietary
Standard Layer
Vendor Layer
Proprietary Layer
Open Source (or targeting)
Data path over standard API usable by 3rd party apps coming from app stores

Goal

- Relevant VSS subset is mapped deterministically to Android VHAL properties
  - No extra alignment on mapping each time VSS is amended
  - Ability to update mapping over the vehicle's lifetime
- No extra SDK, no additional requirements for end-app-developer
- Access per property and per app
  - Ability to change access over the vehicle's lifetime
- Transition path between today and target
  - Things have to work now and in the same way when target is reached without changes for OEMs

How do we get there?

Phase 1: VSS Mapper VSS Tools

- Deterministically generating VSS to VHAL Properties mapping
  - Alignment on mapper tooling (one-time) rather then each property (continuous)
- Main challenge
  - Vendor property ID clash due to small space (2 bytes)
- Goal
  - Establishing topic within community
  - Standardization - alignment on basics
  - **No technical use**

Phase 2: Standard VSS VHAL Building Block

- Standard, cross-OEM common VHAL implementation
  - Configurable with the output of VSS VHAL Mapper
- Main challenge
  - Dynamic access management per app changeable over the vehicle's lifetime
  - Mapping updateable over the vehicle's lifetime
- Goal
  - Common standard open-source implementation
  - Cross-OEM VHAL properties
  - **Only vendor VHAL properties possible**
    - **1st and 2nd party apps, no 3rd party apps**
    - **Backward compatibility not guaranteed**

Phase 3: AOSP Patch

- Extending standard properties for standard, cross-OEM common VHAL implementation
- Main challenge
  - Compatibility with future AOSP updates
  - Contributing to AOSP upstream
  - Maintaining patched AOSP till upstream contribution is accepted
- Goal
  - No need to change AOSP code in the future, just VHAL and mapping configuration.
  - **VSS data available as standard VHAL properties**
    - **3rd party app from app store can access any vehicle property which OEM allows**

The Patch

```
project device/generic/car/
diff --git a/hardware/interfaces/automotive/vehicle/aidl_property/android/hardware/automotive/vehicle/VehiclePropertyGroup.aidl
index xxxxxx..xxxxxxx 100644
--- a/hardware/interfaces/automotive/vehicle/aidl_property/android/hardware/automotive/vehicle/VehiclePropertyGroup.aidl
+++ b/hardware/interfaces/automotive/vehicle/aidl_property/android/hardware/automotive/vehicle/VehiclePropertyGroup.aidl
@@ -21,6 +21,7 @@ enum VehiclePropertyGroup {
     SYSTEM     = 0x10000000,
     VENDOR     = 0x20000000,
     BACKPORTED = 0x30000000,
+    VSS        = 0x40000000,
     MASK       = 0xf0000000,
 }
```

The solution may be as simple as this ;-)

… and then one can use the full 2byte ID space!!!

COVESA

Accelerating the future of connected vehicles