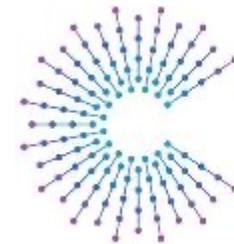


# VSS in-vehicle

KUKSA State of the Union – Android and the Return of VISS

Sebastian Schildt, ETAS GmbH,  
COVESA AMM, October 12<sup>th</sup> 2023



# COVESA

Accelerating the future of connected vehicles

# About: What are we talking about

We like



Vehicle  
Signal  
Specification

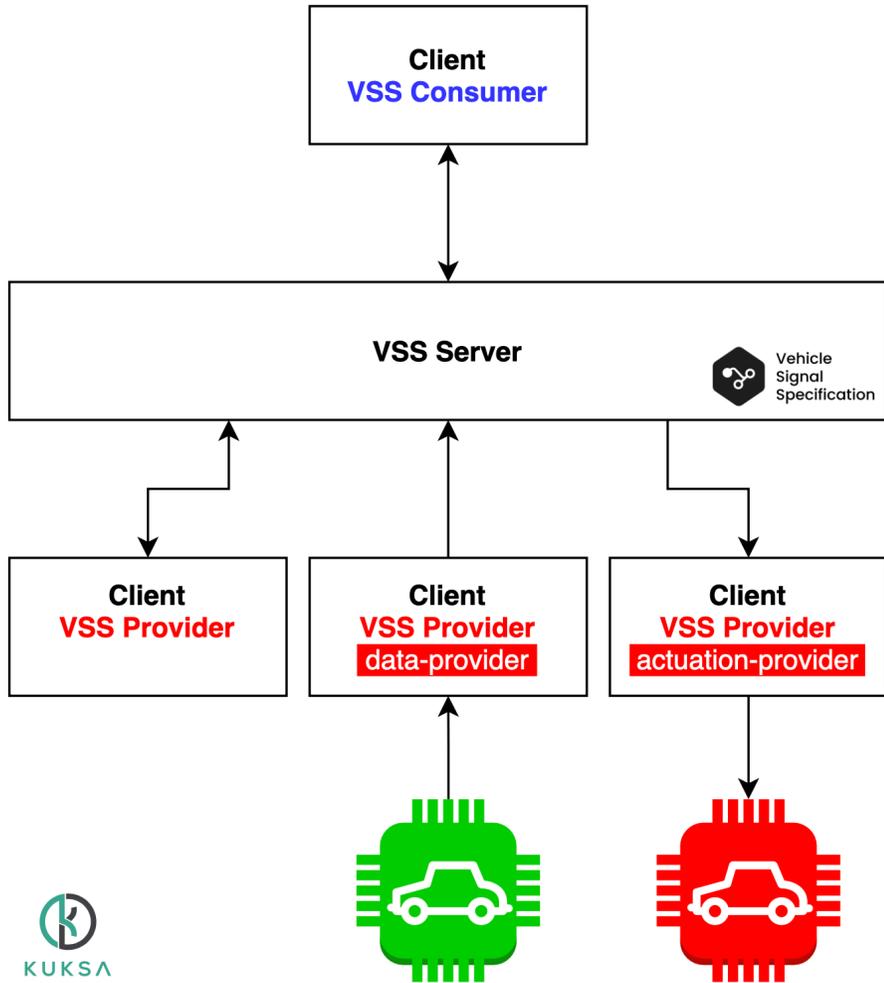
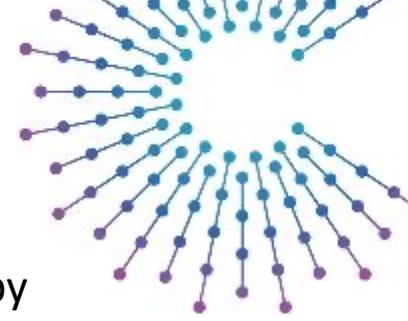
which can be used for

in-vehicle charging B2C Fleet management  
cloud edge cross-domain SDV  
quality B2B infotainment diagnostics

While this is *all amazing*,

*we are most passionate about* providing and using VSS in-vehicle

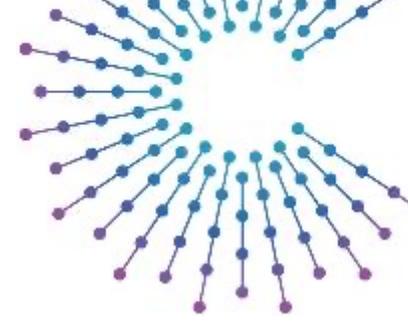
# Taxonomy of in-vehicle VSS components



- Interacts with Vehicle represented by the VSS model
  - Vehicle Computer function
  - IVI App
  - External consumer device
- Holds current vehicle state in VSS format
- Provides an API to interact with VSS signals
- VSS provider syncs of the vehicle with VSS model of the server
  - **data-provider** makes sure that the actual state of a vehicle is represented in VSS (historically known as “feeder”)
  - **actuation-provider** makes ensure that the target value of a VSS actuator is reflected by the actual state of a vehicle

<https://github.com/eclipse/kuksa.val/blob/master/doc/terminology.md>

# The previous episodes...

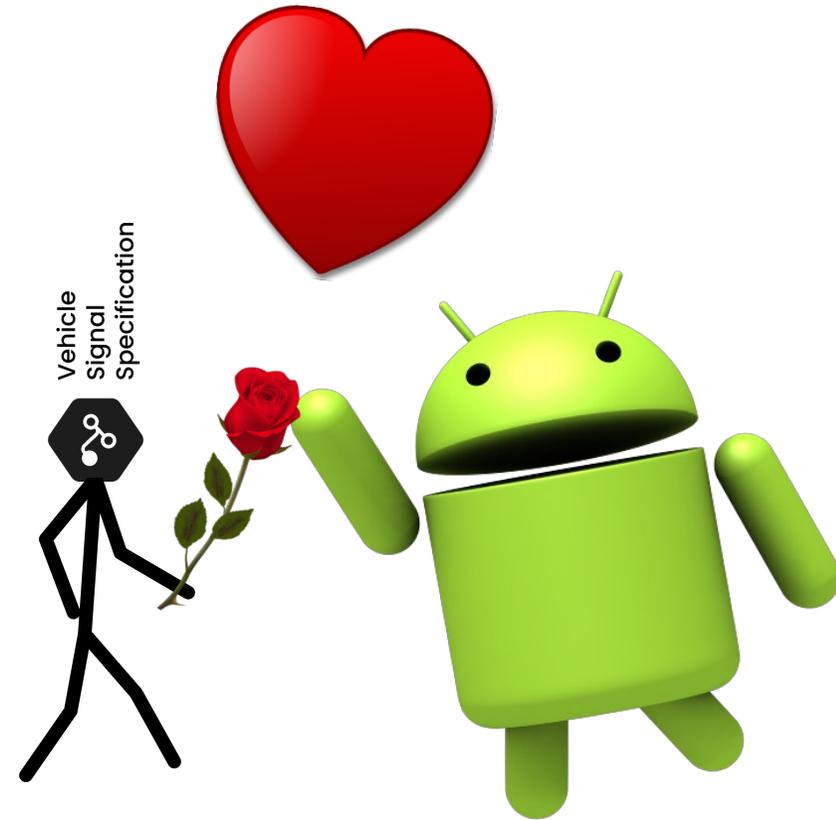
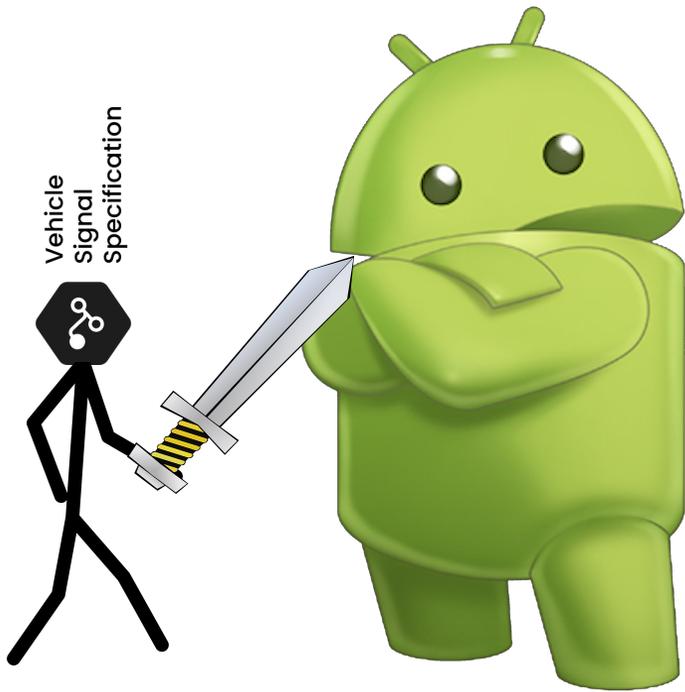
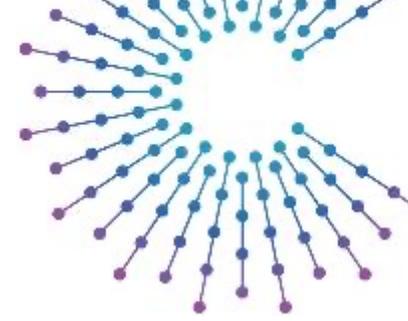


- **COVESA All members Meeting October 2022**  
[Applying VSS In-Vehicle](#)
  - Why in-vehicle is a good place to start deploying VSS
- **FOSDEM 2023, February 2023**  
[KUKSA.val Vehicle Abstraction In-vehicle access to standardized VSS Vehicle Signals](#)
  - A deeper dive into KUKSA architecture and usage
- **COVESA All members Meeting April 2023**  
[Deployment Options and Security Architecture using VSS in-vehicle](#)
  - Deployment blueprints using VSS in-vehicle (how open/dynamic do you need your system?)
  - Security considerations
- **Automotive Grade Linux All Member Meeting Summer 2023**  
[Evolving VSS Usage in AGL \(slides, video\)](#)
  - Status of Automotive Grade Linux adopting VSS and integrating KUKSA

A decorative graphic at the top of the slide consists of a network of interconnected nodes and lines. The nodes are represented by small circles, and the lines are thin, connecting the nodes in a complex, web-like pattern. The color of the nodes and lines transitions from a dark blue on the left to a light blue on the right, with a white background in the center.

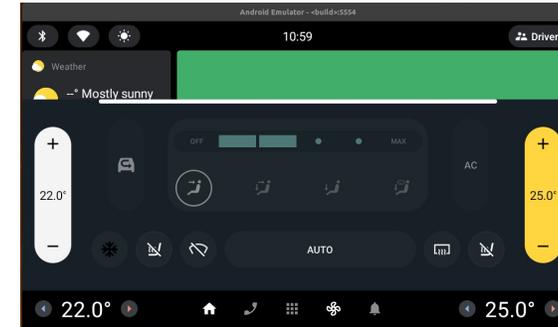
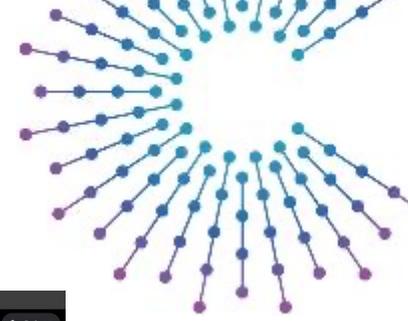
# VSS and Android

# What about Android?



# A closer look: Android (for) Automotive

- Android Automotive is common choice for infotainment systems and often also a base for “third party” functions
- Android Automotive offers the “VHAL” mechanism to access certain Vehicle data



VHAL

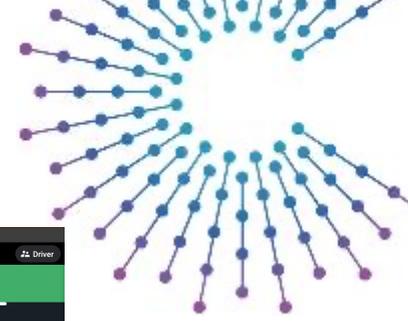
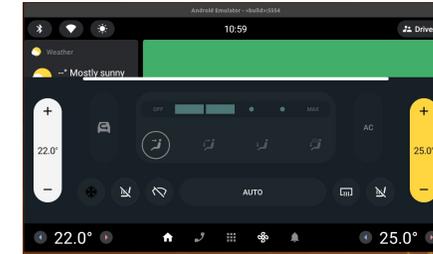


- Using Android Automotive `_without_` VHAL, is a *dumb* idea
  - You are losing the ecosystem benefit, that likely lead to the decision to use Android in the first place
- Good news: If you are a VSS user, you definitely can use a subset of your VSS data to provide a VHAL!
  - This pattern has been shown in recent COVESA meetings, i.e. [here](#) and [here](#)



# VHAL limitations

- You might want to use more datapoints for you internal functionalities that you provide via VHAL
- You might not want to entrust Android Security and ACL model with all your data



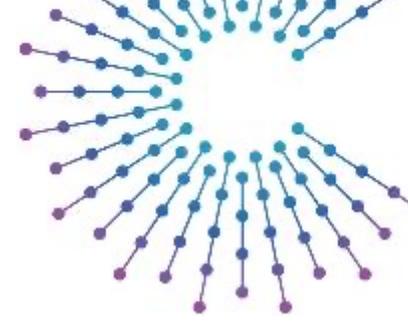
VHAL



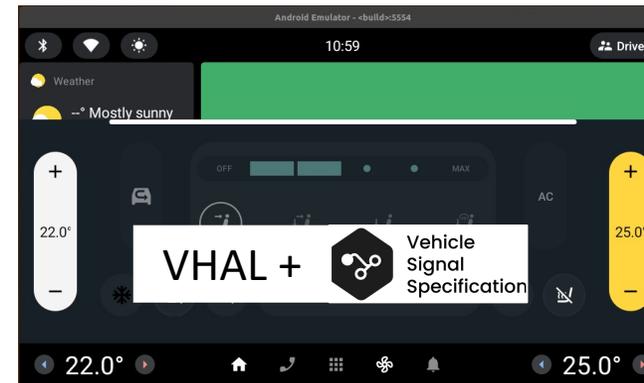
- The VHAL implementation is a vendor component that baked into your Android system image, it is *harder to update* than just an app
  - Changing or extending VHAL -> Android system update
- What about your customer's smartphones?
  - Not VHAL there



# KUKSA: Enable a direct VSS interface besides VHAL



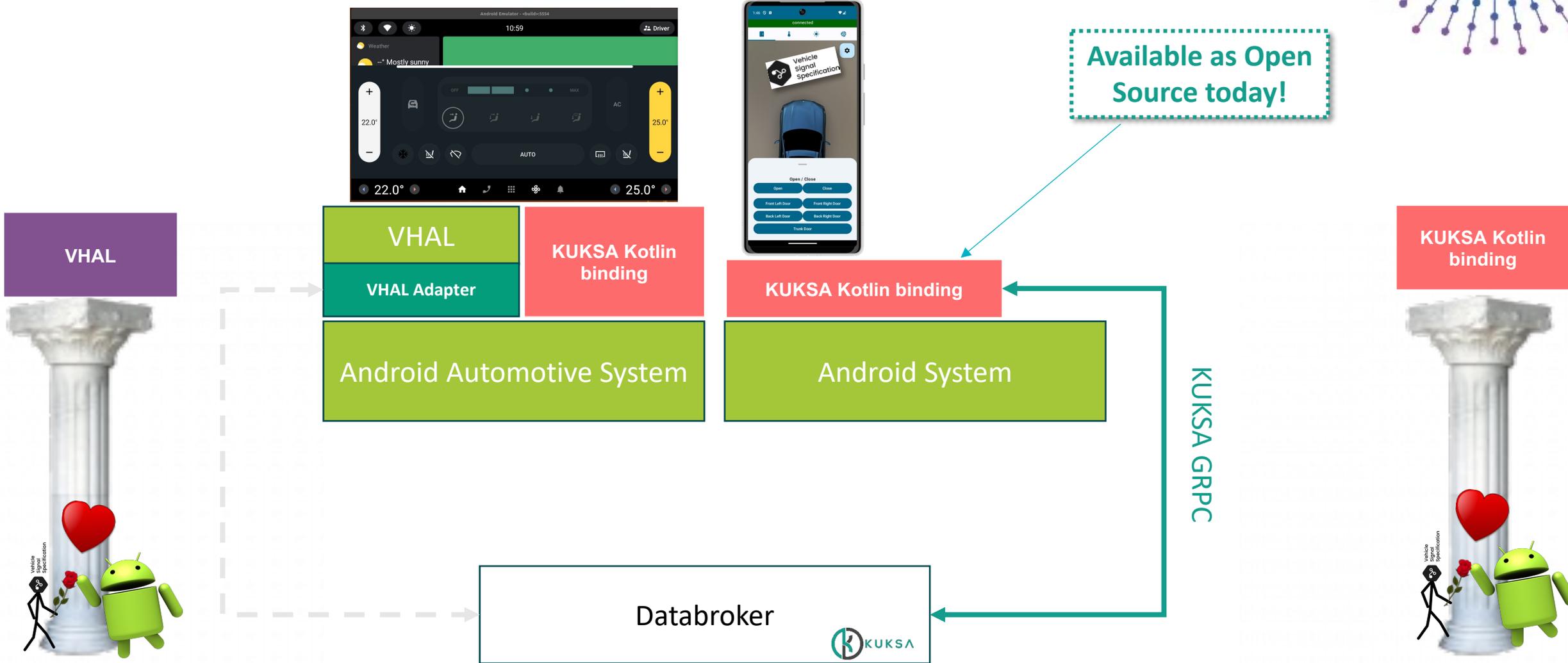
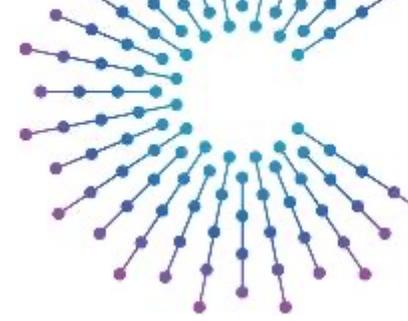
- A Kotlin (the programming language used on the Android platform) library wrapping the standard KUKSA GRPC API (i.e. like the Python binding)
- Run on any Android system, usable from Kotlin and Java language applications
- You are independent from Android system updates to support new data points
  - The lib is embedded into the application
- When developing a non-Automotive Android App, or accessing non-standard OEM-specific datapoints without needing to modify the Android system/vendor image



KUKSA Kotlin binding



# Enable a direct VSS interface besides VHAL

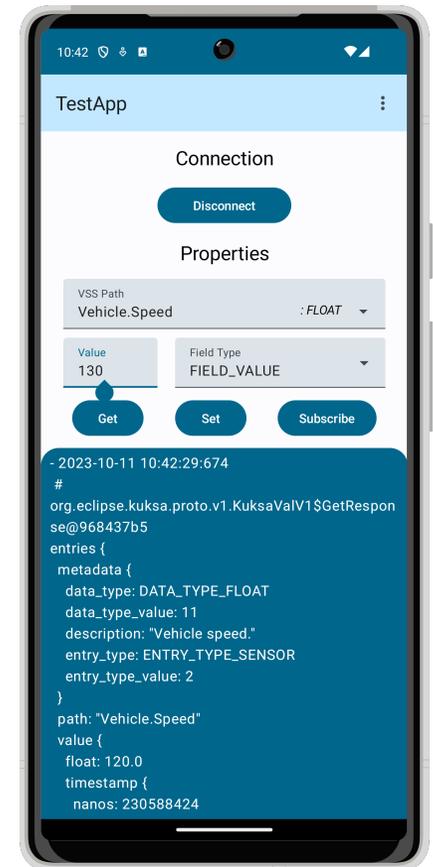


# Freshly backed: kuksa-android-sdk

- Get an initial version at <https://github.com/eclipse-kuksa/kuksa-android-sdk>
- Write Kotlin (or Java) apps gains KUKSA databroker leveraging your complete VSS model

```
val managedChannel = ManagedChannelBuilder.forAddress(host, port)
    .usePlaintext()
    .build()
val connector = DataBrokerConnector(managedChannel)
dataBrokerConnection = connector.connect()

val fields = listOf(Types.Field.FIELD_VALUE)
val property = Property("Vehicle.Speed", fields)
val response = dataBrokerConnection.fetch(property)
val value = response.entriesList[0].value
```



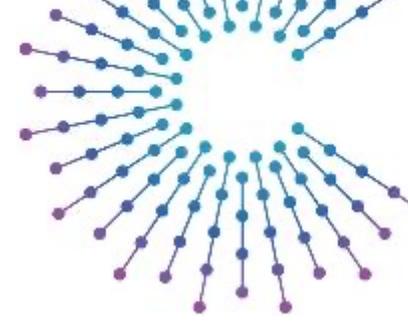
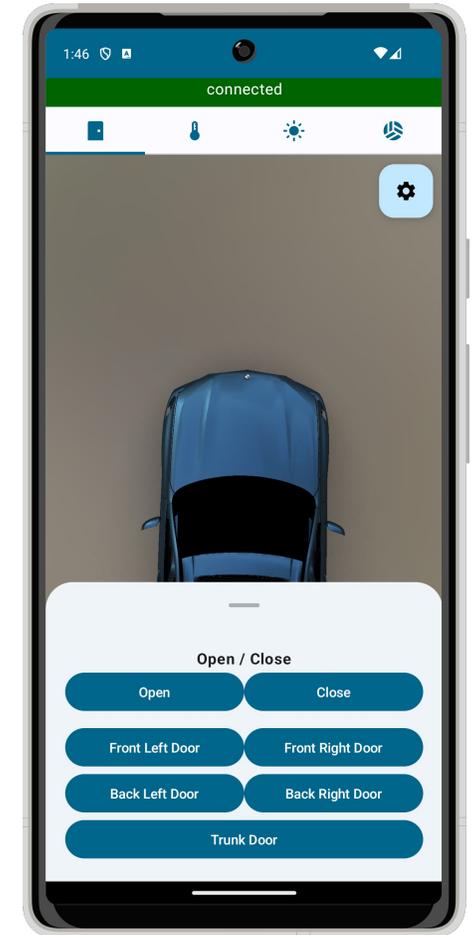
# Sneak Peak: Android Example App

Showcase how to build an Android App based on VSS data using KUKSA databroker

Serves as a blueprint for your own apps

See it live at next Eclipse SDV hackathon November 28 - 30, 2023 in Munich: <https://sdv.eclipse.org/sdv-hackathon-2023/>

Code online shortly before or after





# KUKSA and VISS

# Other APIs: VISS

## VISS is a W3C API to

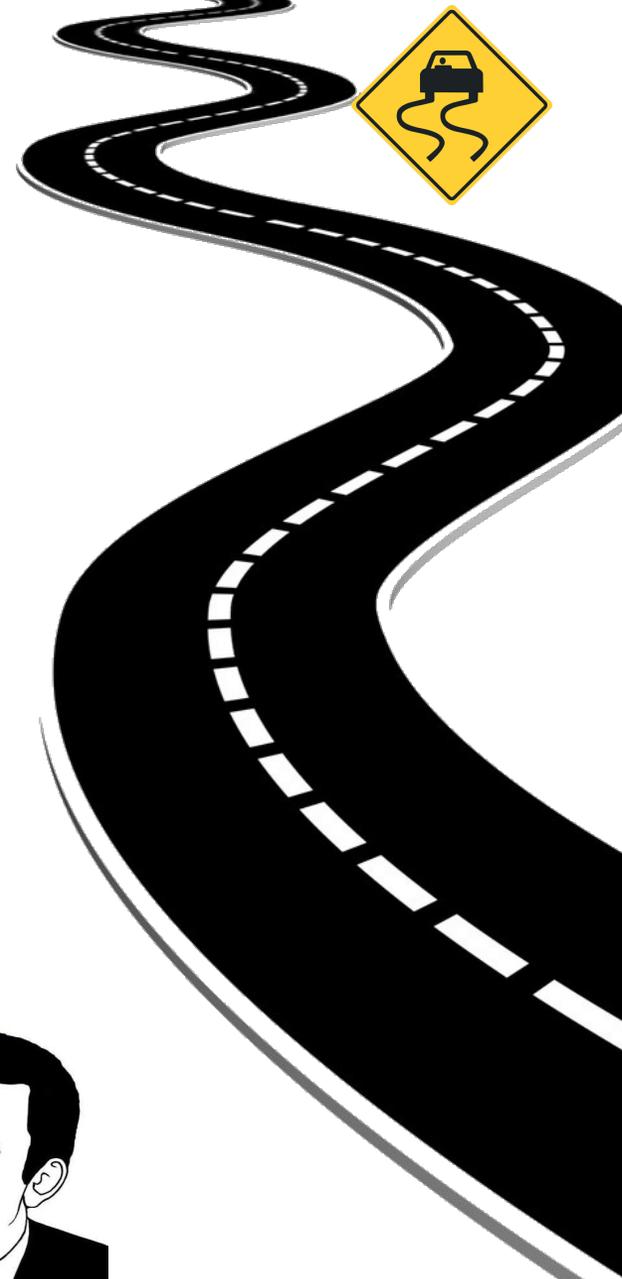
- Access VSS data via websocket (VISS V1 and V2) or HTTP/MQTT

## VISS & KUKSA have a turbulent past

- Legacy KUKSA val-server is a C++ VSS server that started supporting only VISSv1
  - It extended VISS
  - It supported a small subset of VISS V2
  - There were – never finished moves – to change from VISS
- Current RUST based VSS server databroker never supported VISS, instead it uses a GRPC based API
  - We felt –*for our use cases*- it brought too much complexity/features and not enough performance

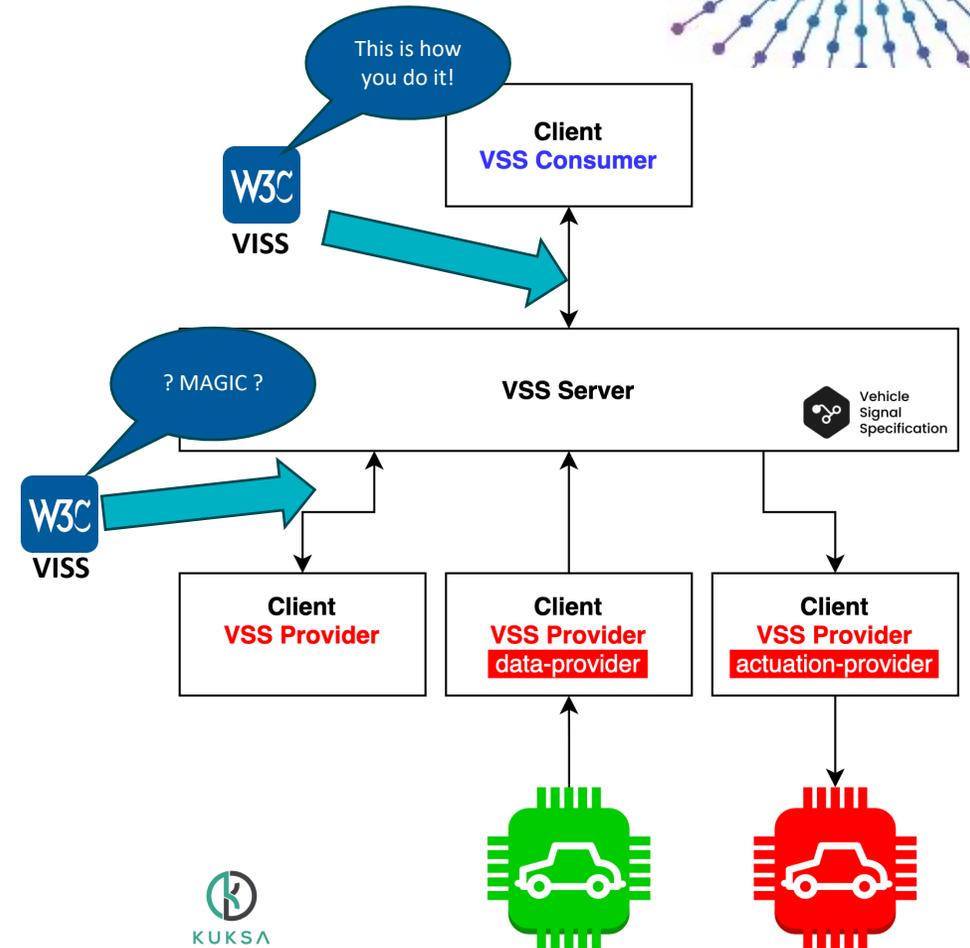
We are currently looking into VISS again

<https://github.com/w3c/automotive>



# Reasons No

- VISS predominantly developed with an "Application mindset"
  - Not much focus on providing data – which is fundamental in KUKSA
  - Was extended in earlier KUKSA version
- JSON + Websocket (HTTP/MQTT) is not the best technology in a vehicle, when you expect
  - mediocre "last generation Pi" processing power
  - Likely want to work with compiled languages such as C++, Rust
- Scope of features in VISSv2 grew much beyond what we want to support in a small, efficient in-vehicle application

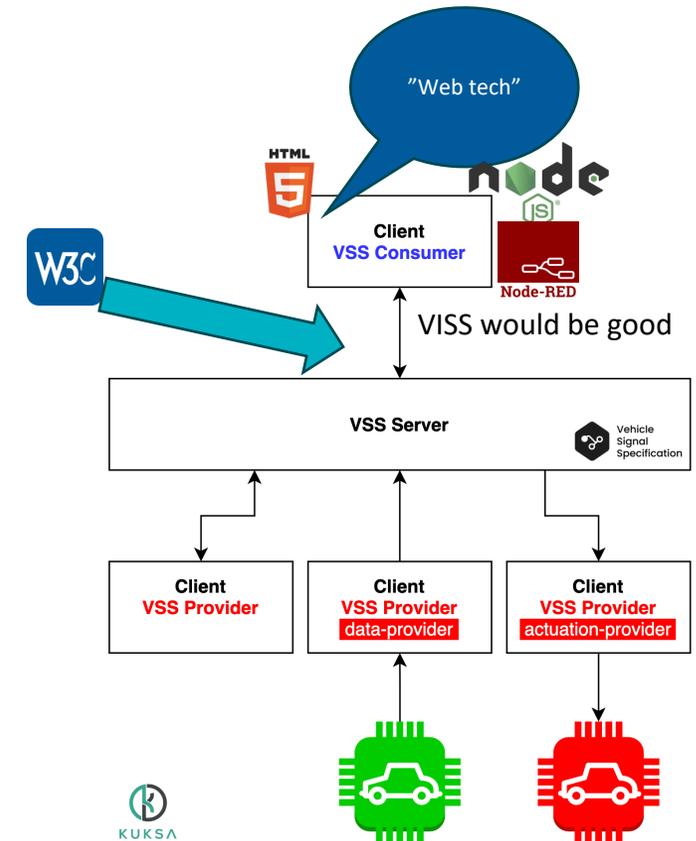


# Reasons Yes

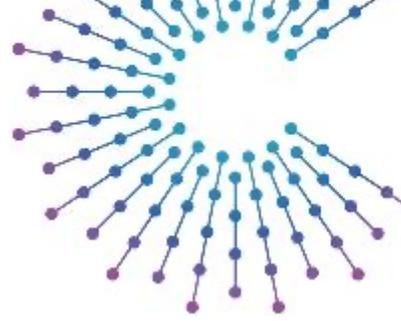
- Using JavaScript based stacks (nodeJS), writing PWA (“HTML5 apps”)
  - Websockets+JSON much more accessible than GRPC
- Most applications need only very basic functions(get/set (actuators) / subscribe (sensors))

## We currently have two experiments

- Implement it as an external component on top of KUKSA GRPC interface:  
<https://github.com/eclipse-kuksa/kuksa-viss>
  - Keep databroker lean
- Implement it in databroker: <https://github.com/eclipse/kuksa.val/pull/642>
  - Can be compiled optionally
  - More efficient



# VISS+KUKSA way forward



- We still feel
  - Our GRPC interface is the way to go for providers, and in vehicle functions (written in compiled languages)
  - For applications based on more “web” tech stacks, as you may find on infotainments, or user device’s something like Websocket-based VISS may be more accessible
- We will make up our mind, whether we want an external version, or databroker integrated one
  - You can help us make up our mind. We are open source after all

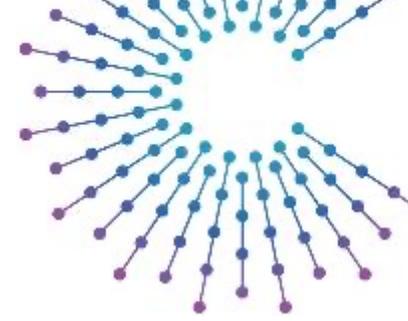
We will – as always – try to implement what is best for our specific use case/product and our users, we intend our subset to be compatible with VISS V2.

We believe it might be a good idea - on W3C VISS side to define a very minimal base level of “VISS-compliance”, i.e. call what we are aiming for “VISS level 1 compliance”, and potentially define other tiers  
-> Happy to discuss

A decorative graphic at the top of the slide consists of a network of interconnected nodes and lines. The nodes are represented by small circles, and the lines are thin, connecting the nodes in a complex, web-like pattern. The color of the nodes and lines transitions from a dark blue on the left to a light blue on the right.

# VSS Mocking

# Mock Service



- Testing business logic against VSS is not always easy
- In KUKSA you could manually set some values using the test clients → tedious
- You could connect a complete vehicle/bus simulation → complex
- Solution: KUKSA Mock Service
  - Registers as KUKSA provider
  - Describe behaviour using simple DSL based on VSS

```
mock_datapoint(  
  path="Vehicle.Speed",  
  initial_value=0.0,  
  behaviors=[  
    create_behavior(  
      trigger=ClockTrigger(0),  
      action=create_animation_action(  
        duration=10.0,  
        repeat_mode=RepeatMode.REPEAT,  
        values=[0, 30.0, 50.0, 70.0, 100.0, 70.0, 50.0, 30.0, 0.0],  
      ),  
    ],  
)
```



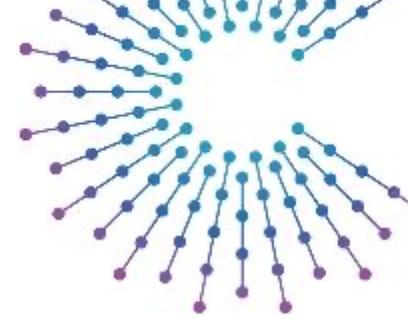
```
mock_datapoint(  
  path="Vehicle.Body.Windshield.Front.Wiping.System.Mode",  
  initial_value="STOP_HOLD",  
  behaviors=[  
    create_behavior(  
      trigger=create_event_trigger(EventType.ACTUATOR_TARGET),  
      action=create_set_action("$event.value"),  
    ),  
  ],  
)
```

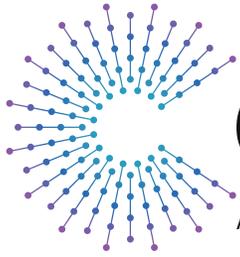


[https://github.com/eclipse/kuksa.val.services/tree/main/mock\\_service](https://github.com/eclipse/kuksa.val.services/tree/main/mock_service)

# Summary

- VSS in vehicle is a good idea
- You can get more out of Android (Automotive) with VSS
  - VSS can be the base for VHAL
  - KUKSA Kotlin library available today to enable you complete VSS models in any Linux
- A base subset of VISS is useful for apps written using web technologies
- Experiment with and/or contribute to KUKSA VSS server
- Engage in VSS online meetings and see you next AMM





# COVESA

Accelerating the future of connected vehicles

## Thank you

## Contact & Information

KUKSA



<https://eclipse.github.io/kuksa.website/>

ETAS OSS manifesto



<https://www.etas.com/en/open-source-software.php>

Me



<http://sdv.expert>

Eclipse SDV



<https://sdv.eclipse.org>

COVESA VSS



[https://covesa.github.io/vehicle\\_signal\\_specification/](https://covesa.github.io/vehicle_signal_specification/)