# VSS In-Vehicle Access API

Sebastian Schildt, Sven Erik Jeroschewski

COVESA DEG Workshop, September 24th 2024

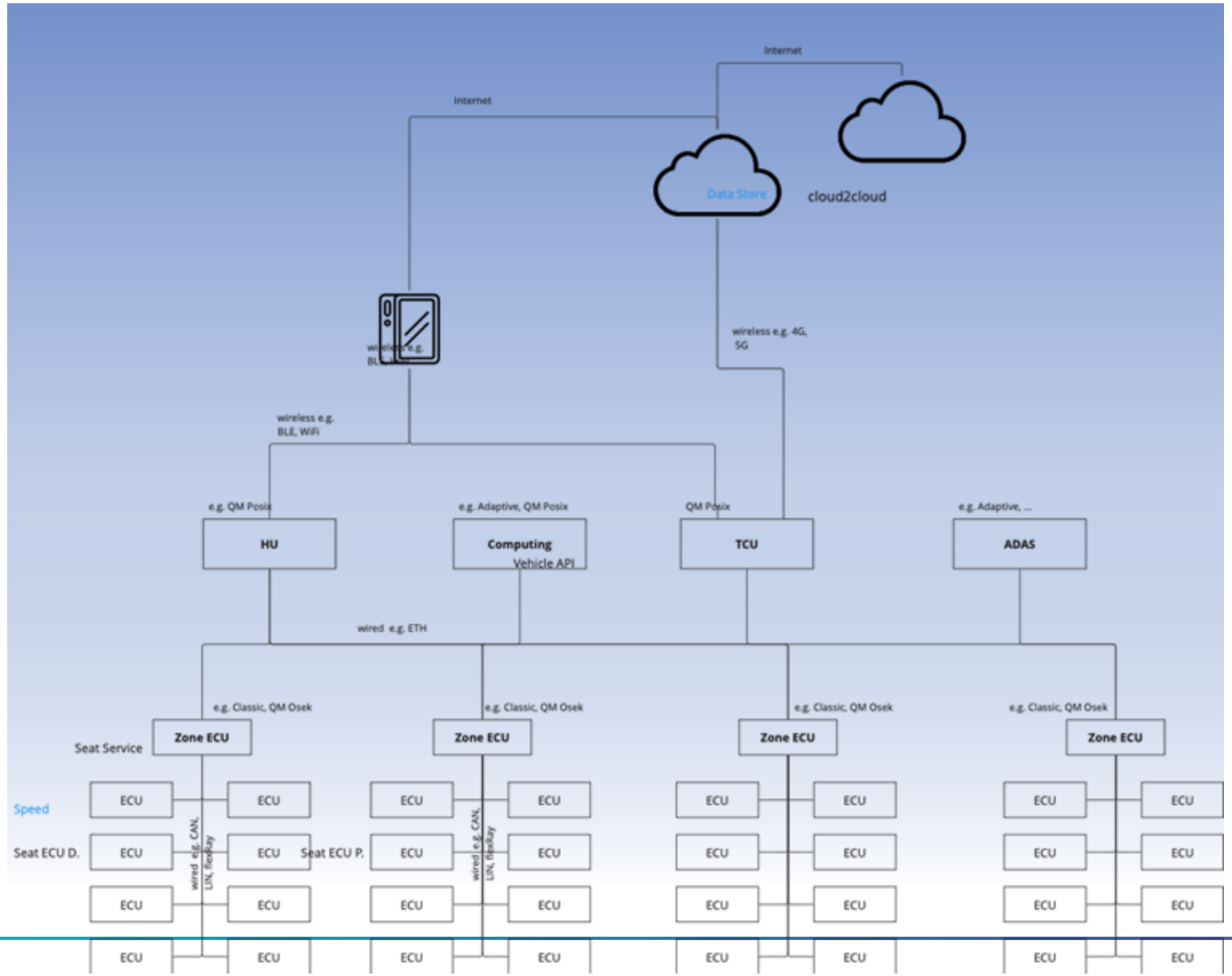
COVESA

Accelerating the future of connected vehicles

# VIVA

VSS In-Vehicle Access

# VSS scope




Vehicle Signal Specification

Mobile

Zone controller

Cloud

Edge

in-vehicle

SDV

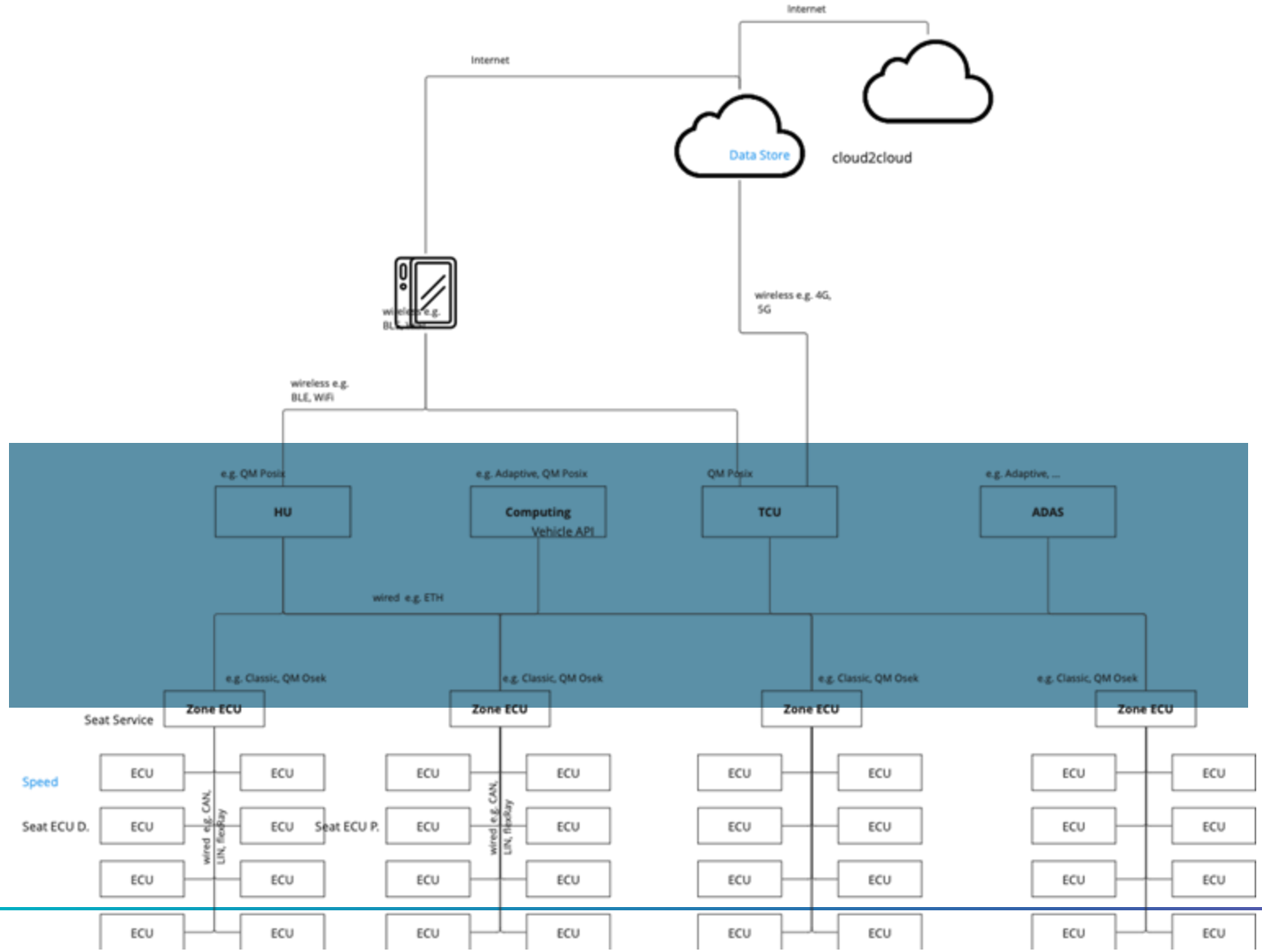
Vehicle Computers

ECU?

infotainment

COVESA

# Our scope




Making

Vehicle Signal Specification

usable

**in-vehicle**

COVESA

# Why another API ?

## Core principals of VIVA *(a VISSv3 ext)*

### Performant

- ✓ Providing thousands of vehicle signals concurrently
- ✓ RPC type communication for embedded implementation
- ✓ Lightweight transport protocol friendly
- ✓ Optimized to inter-connect with in-vehicle technologies (VSS server Southbound focus)

### Error handling

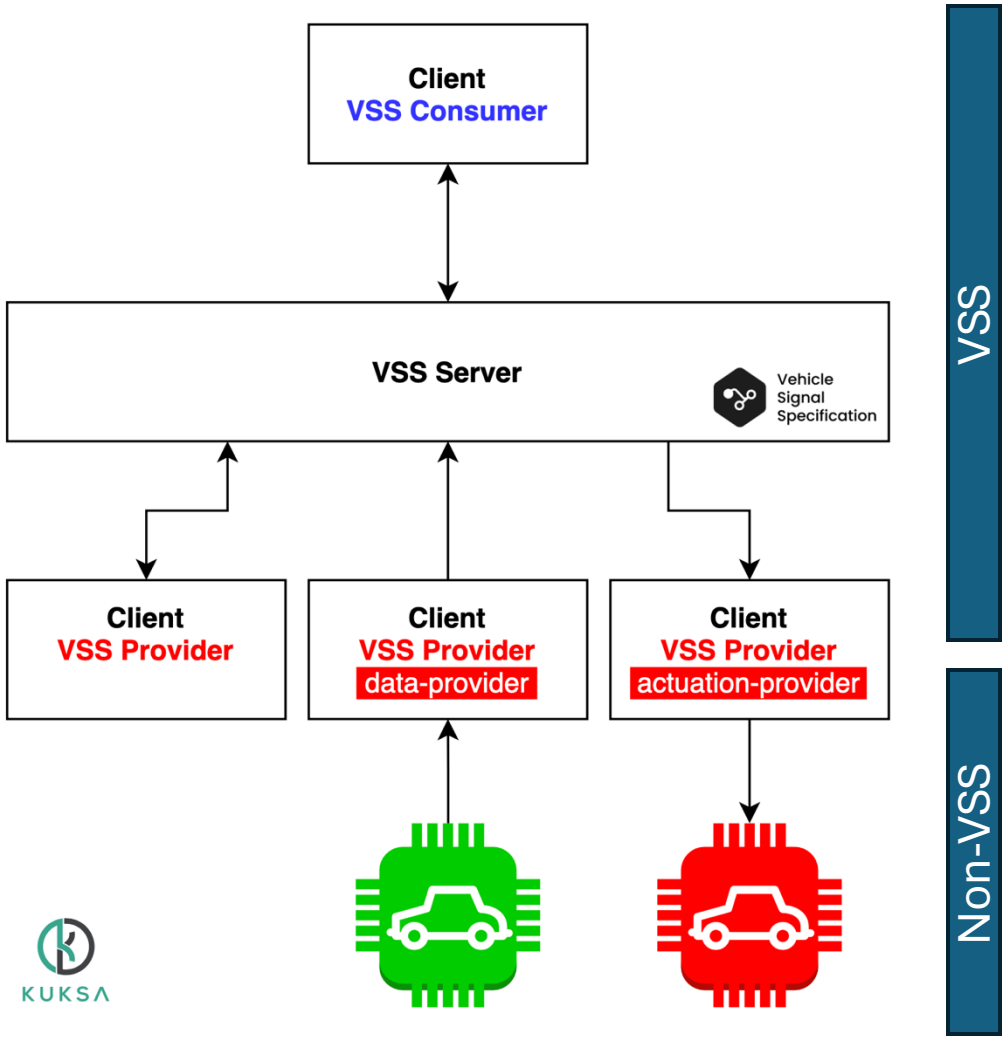
- ✓ need better error handling down to the data providers in a VSS server/API
- ✓ Need detailed responses to handle failed actuations
- ✓ Handle response appropriately to avoid unexpected side effects

### Secure

- ✓ Hardened API with minimal surface for in-vehicle operations
- ✓ Avoid unnecessary complexity

COVESA

# Basic architecture for VSS in-vehicle




- Apps/clienbts Interacts with Vehicle represented by the VSS model

- Holds current vehicle state in VSS format
- Provides an API to interact with VSS signals

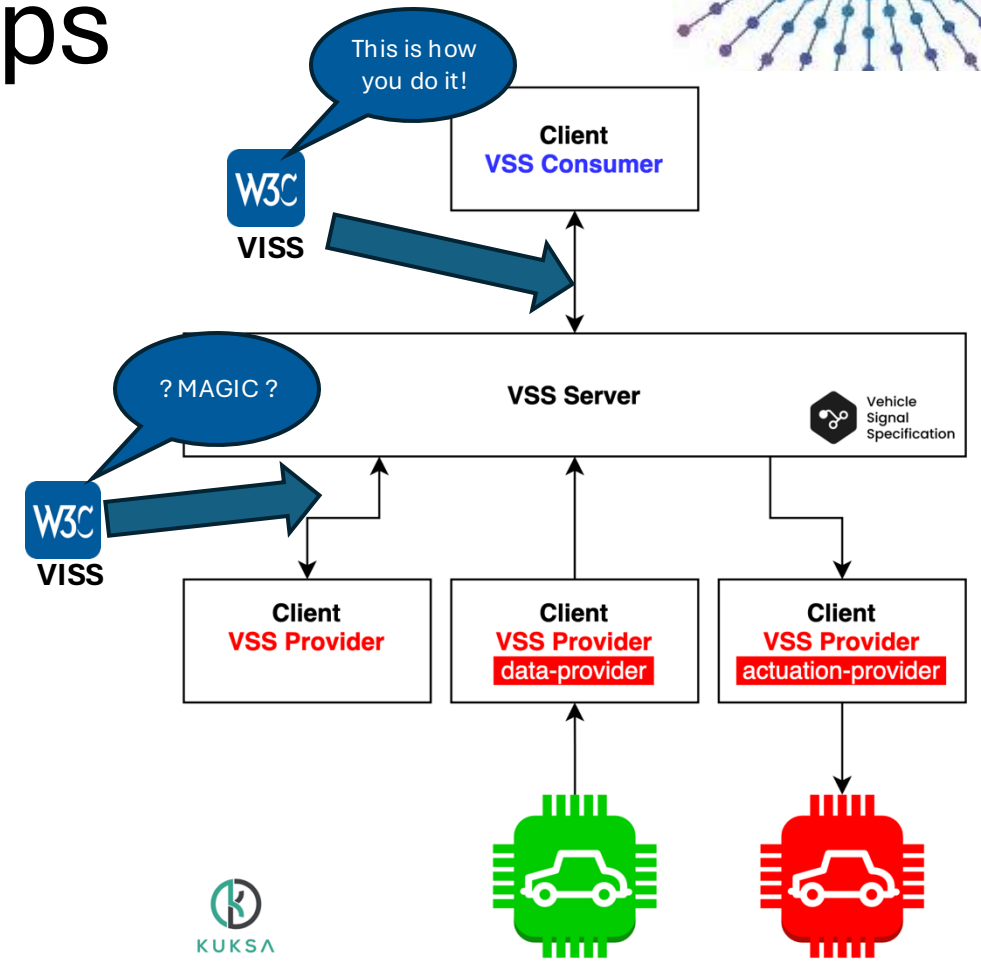
- VSS provider syncs of the vehicle with VSS model of the server
  - **data-provider** makes sure that the actual state of a vehicle is represented in VSS (historically known as “feeder”)
  - **actuation-provider** makes ensure that the target value of a VSS actuator is reflected by the actual state of a vehicle

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

# What is there?
# VISS (without “R”) & KUKSA (API)

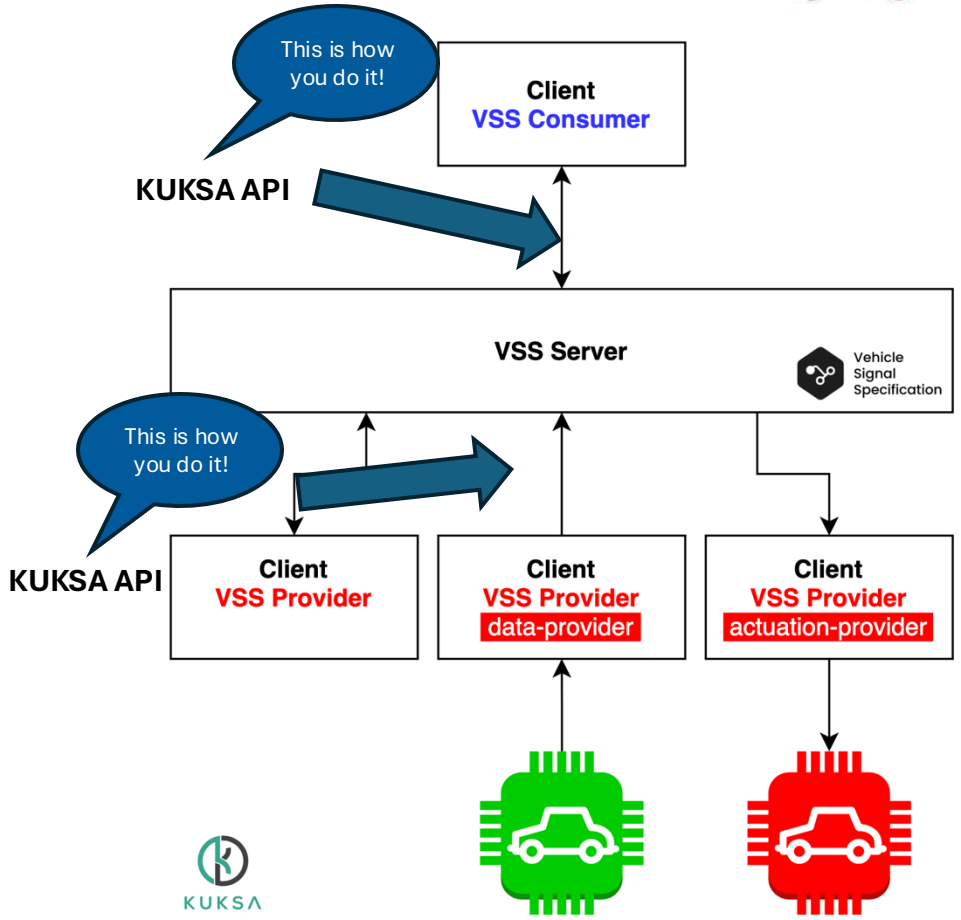
# VISS is designed to serve Apps

- VISS developed with an ”Application mindset”
  - Not much focus on providing data to server – which is fundamental in KUKSA
- JSON + Websocket (HTTP/MQTT) very suitable for “highlevel” App stacks (think “nodejs”) focusing on development speed
- Scope of features in VISSv2/3 grew much beyond what you may want to support in a small, efficient in-vehicle application



# KUKSA is designed for Vehicle integration

- KUKSA scope is supporting collecting and transforming data to VSS from in-vehicle Systems
  - A lot of focus on providing VSS to server – which is fundamental in KUKSA
  - Aim to work in today's vehicle computers with reasonable overhead
- Has a limited scope: Efficiently providing VSS signals and influencing a vehicle's state via VSS



# KUKSA+VISS

**VISS is a ~~W3C~~ COVESA API to**

- Access VSS data via websocket (VISS V1 and V2) or HTTP/MQTT (v2) orGRPC (v3)

**VISS & KUKSA have a turbulent past**

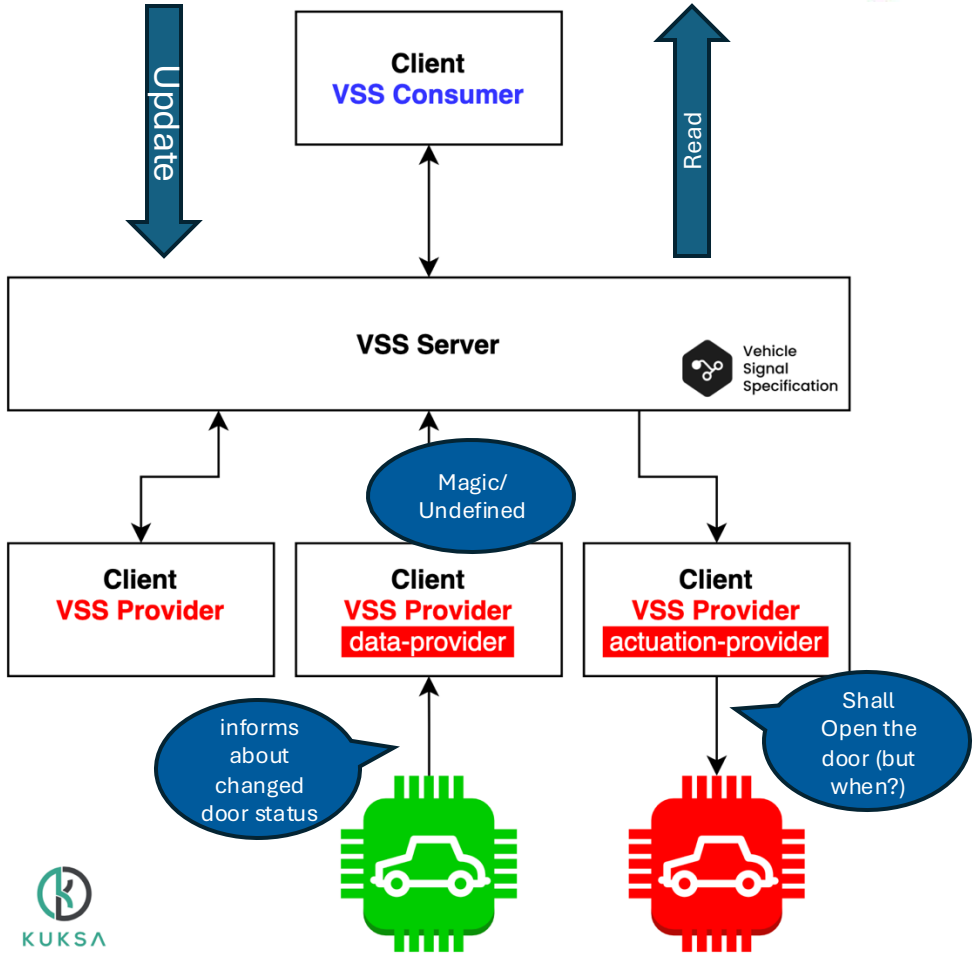
- Legacy KUKSA val-server is a C++ VSS server that started supporting only VISSv1
  - It extended VISSv1 , later supported a small subset of VISS V2
- Current KUKSA databroker initially supported no VISS, instead is used a GRPC based API
  - We felt –*for our use cases*- it brought too much complexity/features and not enough performance
- Later/current databroker versions support minimal VISS V2 subset over websocket

**So let's talk about this again.**
**In more detail.**

# Let’s take a (slightly) deeper look
# API comparison

# Actuation – VISSv3

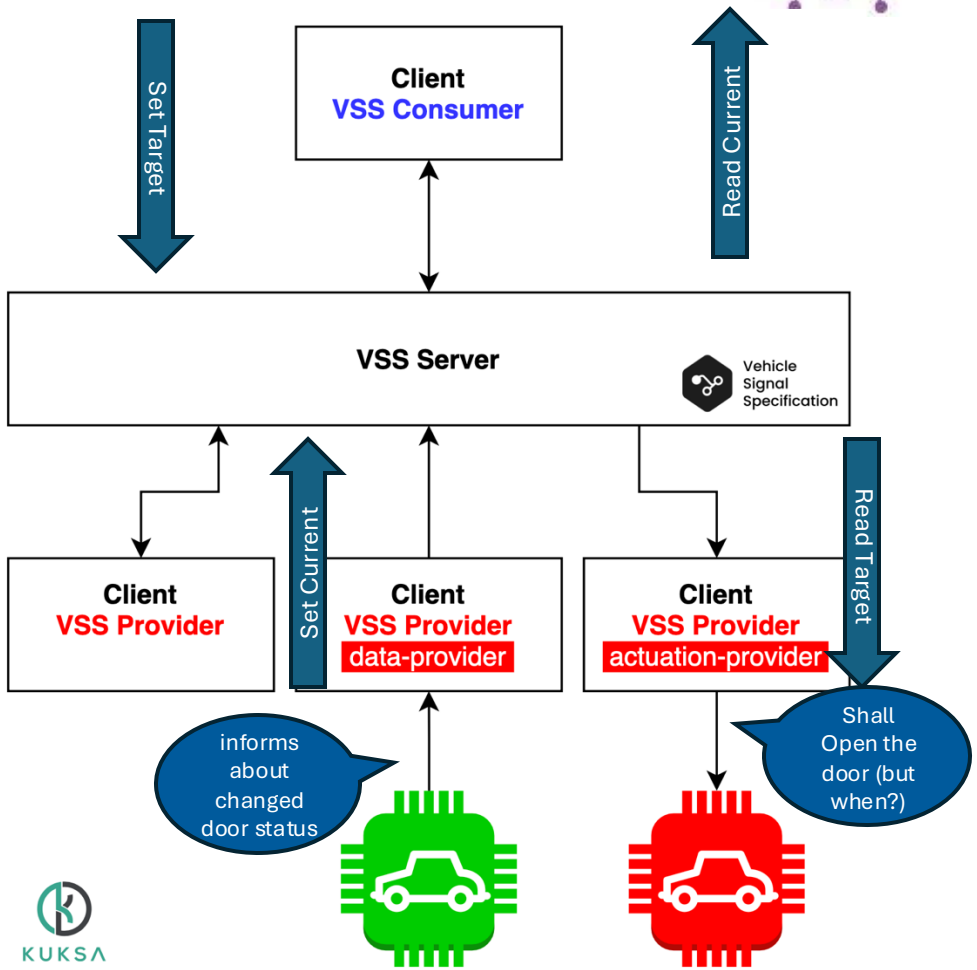
- VISSv3 only exposes **current** values and allows to update with **target values** which are not exposed through the API
- Interaction between providers and VSS server undefined/magic
- Unclear when actuation provider should perform actuation (When Update is executed, when provider wakes up again)
- Limited Error Handling (Signal in Read/Subscribe has not changed but Why?)




COVESA

# Actuation – kuksa.val.v1

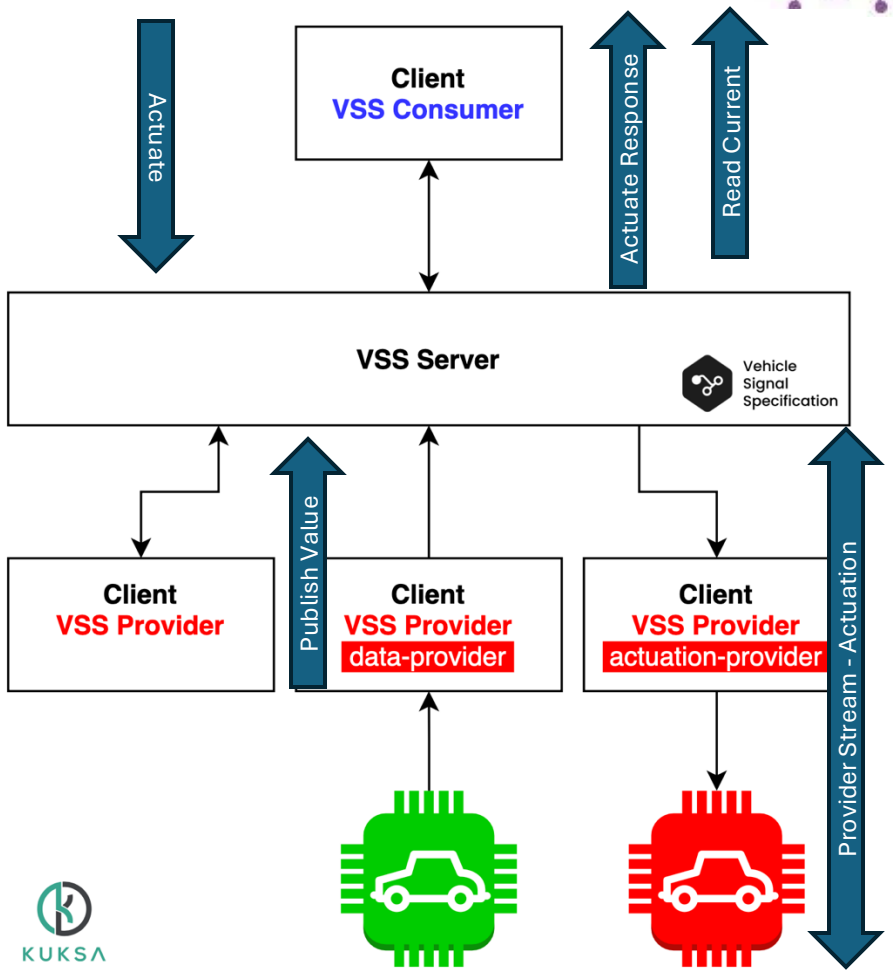
- Set, Read, Subscribe cover both **Current** or **Target** value
- Interaction between providers and VSS server uses same API
- Unclear when actuation provider should perform actuation (When Update is executed, when provider wakes up again)
- Limited Error Handling (Signal in Read/Subscribe has not changed but Why?)




COVESA

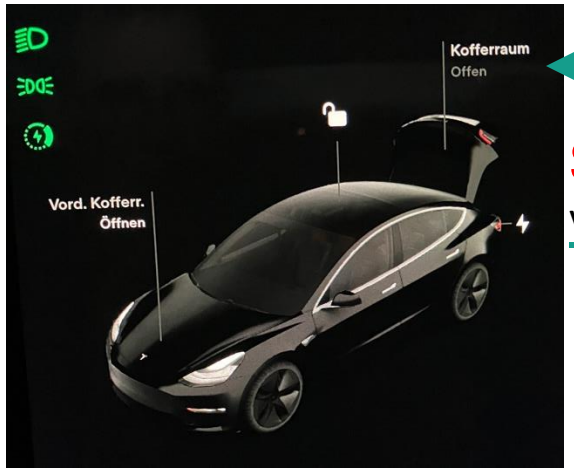
# Actuation – kuksa.val.v2 Draft

- Broker only exposes current value
- Can set target value. Through actuation but gets dropped if no provider is previously subscribed ->stateless actuation
- Actuate has error response that can be influenced by provider
- Interaction between providers and VSS server uses same API
- Todo: Limited Error Handling (Signal in Read/Subscribe has not changed but Why?)
- Todo: Advantage of this approach is that the Error handling can be extended down to the provider




COVESA

# Sensors & Actuators in KUKSA.val.v1




Application

Subscribe current
Vehicle.Body.Trunk.Rear.IsOpen

Set target:
Vehicle.Body.Trunk.Rear.IsOpen

Subscribe target
Vehicle.Body.Trunk.Rear.IsOpen

KUKSA.val data broker

Vehicle Signal Specification

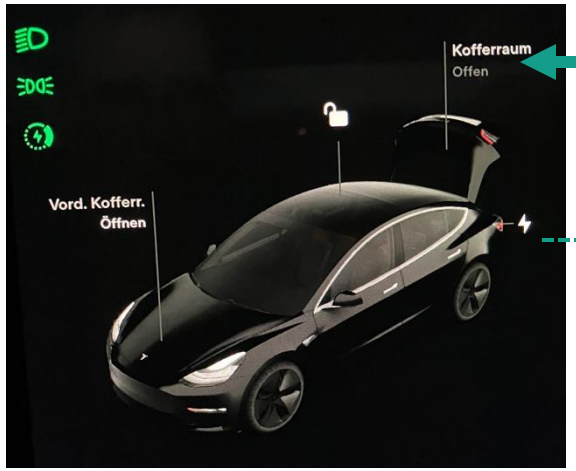
Set current
Vehicle.Body.Trunk.Rear.IsOpen

VSS Trunk data provider

VSS Trunk actuation provider

E/E network (CAN, SOME/IP, LIN, DDS, etc.) or Autosar (adaptive) platfo

# Sensors & Actuators in KUKSA.val v2




Application

Subscribe

Vehicle.Body.Trunk.Rear.IsOpen

Actuate:

Vehicle.Body.Trunk.Rear.IsOpen

KUKSA.val data broker

Vehicle Signal Specification

Provider Stream: Actuation

Vehicle.Body.Trunk.Rear.IsOpen

Provider Stream: Publish

Vehicle.Body.Trunk.Rear.IsOpen

VSS Trunk provider

E/E network (CAN, SOME/IP, LIN, DDS, etc.) or Autosar (adaptive) platform

# VIVA

## In-Vehicle Access

~~VSS~~

VISS³

- Create an in-vehicle API catering to southbound aspects of a VSS server
- This is not just “a transport” for VISS but its own API
- Allows each API variant to cater to its use cases without becoming a large “one size fits all” Frankenstein Middleware
- Make it simple enough there can realistically be more implementations (not “only” VissR, KUKSA)
- **May** or **may not** be under the VISS3 project (we suggest it is)
- **Should** be in COVESA

# DETAILS



Here come 20+ pages of the nitty gritty details


# API Updates – kuksa.val.v1

```
service VAL {
        rpc Get(GetRequest) returns (GetResponse);

        rpc Set(SetRequest) returns (SetResponse);

        rpc Subscribe(SubscribeRequest) returns (stream SubscribeResponse);

        rpc GetServerInfo(GetServerInfoRequest) returns (GetServerInfoResponse);
}
```

# API Updates – kuksa.val.v1

```
service VAL {
        rpc Get(GetRequest) returns (GetResponse);

        rpc Set(SetRequest) returns (SetResponse);

        rpc StreamedUpdate(stream StreamedUpdateRequest) returns (stream
StreamedUpdateResponse);

        rpc Subscribe(SubscribeRequest) returns (stream SubscribeResponse);

        rpc GetServerInfo(GetServerInfoRequest) returns (GetServerInfoResponse);
}
```

# Kuksa.val.v1 -> kuksa.val.v2

- Services (Get, Set, Subscribe, GetInfo) may seem easy to understand but complexity is in transported messages with multitude of functionalities
  - Add more rpc to make messages easier

- Not performing actuation has number of non intuitive or not wanted consequences
  - How long is the actuation valid? (Do not open door after ten minutes)
  - provider not available – do not wait forever for execution of actuation

- Singular rpc call less efficient for continuous operations
  - Introduce more streaming rpcs (e.g., in publishing sensor values)

# API Updates – kuksa.val.v2 (under development)

rpc **Get**(GetRequest) returns (GetResponse)

rpc **GetValue**(GetValueRequest) returns (GetValueResponse);

// Returns all requested signals or PERMISSION_DENIED if access is denied for any requested signalrpc

**GetValues**(GetValuesRequest) returns (GetValuesResponse);

// Only return values of signals that the user is allowed to read (everything else is ignored).

rpc **ListValues**(ListValuesRequest) returns (ListValuesResponse);

# API Updates – kuksa.val.v2 (under development)

rpc **Set**(SetRequest) returns (SetResponse);

//Returns (GRPC error code): (...)

// UNAVAILABLE if there is no provider currently providing the actuator

rpc **Actuate**(ActuateRequest) returns (ActuateResponse);

rpc **BatchActuate**(BatchActuateRequest) returns (BatchActuateResponse);

rpc **PublishValue**(PublishValueRequest) returns (PublishValueResponse);

// Open a stream used to provide actuation and/or publishing values using a streaming interface.

rpc **OpenProviderStream**(stream OpenProviderStreamRequest) returns (stream OpenProviderStreamResponse);

# API Updates – kuksa.val.v2 (under development)

rpc **Subscribe**(SubscribeRequest) returns (stream SubscribeResponse);

rpc **GetServerInfo**(GetServerInfoRequest) returns (GetServerInfoResponse);

rpc **Subscribe**(SubscribeRequest) returns (stream SubscribeResponse);

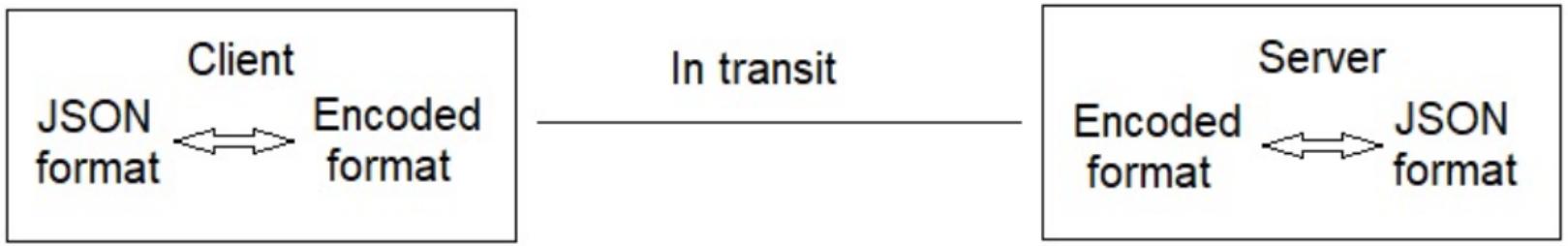
rpc **GetServerInfo**(GetServerInfoRequest) returns (GetServerInfoResponse);

# API Updates – kuksa.val.v2 (under development)

```
service VAL {

        rpc GetValue(GetValueRequest) returns (GetValueResponse);

        rpc GetValues(GetValuesRequest) returns (GetValuesResponse);

        rpc ListValues(ListValuesRequest) returns (ListValuesResponse);

        rpc Subscribe(SubscribeRequest) returns (stream SubscribeResponse);

        rpc Actuate(ActuateRequest) returns (ActuateResponse);

        rpc BatchActuate(BatchActuateRequest) returns (BatchActuateResponse);

        rpc ListMetadata(ListMetadataRequest) returns (ListMetadataResponse);

        rpc PublishValue(PublishValueRequest) returns (PublishValueResponse);

        rpc OpenProviderStream(stream OpenProviderStreamRequest) returns (stream
OpenProviderStreamResponse);

        rpc GetServerInfo(GetServerInfoRequest) returns (GetServerInfoResponse);

}
```

# VISSv3

- VISSv3 enforces JSON format on both sides of the communication but keeps it free between server and client (e.g., Protobuf)




Protocol or “Middleware”?

# VISSv3 vs kuksa.val.v1

```
service VISS {

rpc GetRequest (GetRequestMessage) returns (GetResponseMessage);


rpc SetRequest (SetRequestMessage) returns (SetResponseMessage);


rpc SubscribeRequest (SubscribeRequestMessage) returns (stream
SubscribeStreamMessage);


rpc UnsubscribeRequest (UnsubscribeRequestMessage) returns
(UnsubscribeResponseMessage);

}
```

```
service VAL {

rpc Get(GetRequest) returns (GetResponse);


rpc Set(SetRequest) returns (SetResponse);


rpc StreamedUpdate(stream StreamedUpdateRequest) returns (stream
StreamedUpdateResponse);


rpc Subscribe(SubscribeRequest) returns (stream SubscribeResponse);


rpc GetServerInfo(GetServerInfoRequest) returns (GetServerInfoResponse);

}
```

COVESA

# VISSv3 vs kuksa.val.v1 – Get Request

```
message GetRequestMessage {
 string Path = 1;
 optional FilterExpressions Filter = 2;
 optional string Authorization = 3;
 optional string RequestId = 4;
}

message FilterExpressions {
 message FilterExpression {
  enum FilterVariant {
   PATHS = 0;
   TIMEBASED = 1;
   (…)
  }
  FilterVariant Variant = 1;
  message FilterValue {
    optional PathsValue ValuePaths = 1;
    optional TimebasedValue ValueTimebased = 2;
    optional CurvelogValue ValueCurvelog = 5;
    (…)
}
FilterValue Value = 2;
 }
repeated FilterExpression FilterExp = 1; }
```

```
message GetRequest {
  repeated EntryRequest entries = 1;
}

message EntryRequest {
  string path           = 1;
  View view             = 2;
  repeated Field fields = 3;
}
enum View {
  VIEW_CURRENT_VALUE = 1;
  VIEW_TARGET_VALUE  = 2;
  VIEW_METADATA      = 3;
  VIEW_ALL           = 20;
 (…)
}
enum Field {
  FIELD_PATH                    = 1;
  FIELD_VALUE                   = 2;
  FIELD_ACTUATOR_TARGET         = 3;
  FIELD_METADATA                = 10;
(…)
}
```

# VISSv3 vs kuksa.val.v1 – Get Response

```
message GetResponseMessage {
  ResponseStatus Status = 1;
  optional SuccessResponseMessage
SuccessResponse = 2;
  optional ErrorResponseMessage
ErrorResponse = 3;
  optional string RequestId = 4;
  string Ts = 5;
  optional string Authorization = 6;}

enum ResponseStatus {
 SUCCESS = 0;
 ERROR = 1; }
message SuccessResponseMessage {
    optional DataPackages DataPack = 1;
    optional string Metadata = 2; }
```

```
message DataPackages {
 message DataPackage {
  string Path = 1;
  message DataPoint {
    string Value = 1;
     string Ts = 2; }
  repeated DataPoint Dp = 2; }
repeated DataPackage Data = 1;}

message ErrorResponseMessage {
  string Number = 1;
  optional string Reason = 2;
  optional string Message = 3; }
```

```
message GetResponse {
 repeated DataEntry entries     = 1;
  repeated DataEntryError errors = 2;
  Error error                    = 3; }

message DataEntry {
  string path = 1;
  Datapoint value = 2;
  Datapoint actuator_target = 3;
  Metadata metadata = 10; }

message Datapoint {
  google.protobuf.Timestamp timestamp = 1;
  oneof value {
    string string          = 11;
    bool bool              = 12;
    (…) } }
```

```
message DataEntryError {
  string path = 1;  // vss path
  Error error = 2;
}
// Should follow VISSv2 codes
message Error {
  uint32 code    = 1;
  string reason  = 2;
  string message = 3;}

message Metadata {
  (…)
  oneof entry_specific {
    Actuator actuator   = 20;
    Sensor sensor       = 30;
    Attribute attribute = 40; }
}
```

COVESA

# VISSv3 vs kuksa.val.v1 – Set Request

```
message SetRequestMessage {
  string Path = 1;
  string Value = 2;
  optional string Authorization = 3;
  optional string RequestId = 4;}
```

```
message SetRequest {
  repeated EntryUpdate updates = 1;}

message EntryUpdate {
  DataEntry entry       = 1;
  repeated Field fields = 2;}

 message DataEntry {
  string path = 1;
  Datapoint value = 2;
  Datapoint actuator_target = 3;
  Metadata metadata = 10; }

enum Field {
  FIELD_VALUE                    = 2;
  FIELD_ACTUATOR_TARGET          = 3;
  FIELD_METADATA                 = 10;
(…) }
```

```
message Datapoint {
  google.protobuf.Timestamp
timestamp = 1;
  oneof value {
    string string            = 11;
    bool bool                = 12;
    (…) } }
```

# VISSv3 vs kuksa.val.v1 – Set Response

```
message SetResponseMessage {
  ResponseStatus Status = 1;
  optional ErrorResponseMessage ErrorResponse = 2;
  optional string RequestId = 3;
  string Ts = 4;
  optional string Authorization = 5;}

enum ResponseStatus {
  SUCCESS = 0;
  ERROR = 1;}

message ErrorResponseMessage {
  string Number = 1;
  optional string Reason = 2;
  optional string Message = 3;}
```

```
message SetResponse {
  Error error                    = 1;
  repeated DataEntryError errors = 2;
}

message Error {
  uint32 code    = 1;
  string reason  = 2;
  string message = 3;
}

message DataEntryError {
  string path = 1;  // vss path
  Error error = 2;
}
```

# VISSv3 vs kuksa.val.v1 – Subscribe Request

```
message SubscribeRequestMessage {
 string Path = 1;
 optional FilterExpressions Filter = 2;
 optional string Authorization = 3; string RequestId = 4; }
```

```
message SubscribeRequest {
  repeated SubscribeEntry entries = 1;
}

message SubscribeEntry {
  string path           = 1;
  View view             = 2;
  repeated Field fields = 3;
}
enum View {
  VIEW_CURRENT_VALUE = 1;
  VIEW_TARGET_VALUE  = 2;
  VIEW_METADATA      = 3;
  VIEW_ALL           = 20;
 (…)
}
enum Field {
  FIELD_VALUE                    = 2;
  FIELD_ACTUATOR_TARGET          = 3;
  FIELD_METADATA                 = 10;
(…) }
```

# VISSv3 vs kuksa.val.v1 – Subscribe Response

```
message SubscribeStreamMessage {
  SubscribeResponseType MType = 1;
  ResponseStatus Status = 2;
  message SubscribeResponseMessage {
    optional ErrorResponseMessage
ErrorResponse = 1;
    optional string SubscriptionId = 2;
    string RequestId = 3;
    string Ts = 4;
    optional string Authorization = 5;}
  optional SubscribeResponseMessage
Response = 3;
  message SubscribeEventMessage {
    string SubscriptionId = 1;
    message SuccessResponseMessage {
      DataPackages DataPack = 1; }
    optional SuccessResponseMessage
SuccessResponse = 2;
    optional ErrorResponseMessage
ErrorResponse = 3;
    string Ts = 4;}
  optional SubscribeEventMessage Event
= 4;}
```

```
message ErrorResponseMessage {
  string Number = 1;
  optional string Reason = 2;
  optional string Message = 3;}
enum SubscribeResponseType {
  RESPONSE = 0;
  EVENT = 1; }
message DataPackages {
  message DataPackage {
    string Path = 1;
    message DataPoint {
      string Value = 1;
      string Ts = 2; }
    repeated DataPoint Dp = 2;}
  repeated DataPackage Data = 1;
}
enum ResponseStatus {
  SUCCESS = 0;
  ERROR = 1; }
```

```
message SubscribeResponse {
  repeated EntryUpdate updates = 1;
}

message EntryUpdate {
  DataEntry entry       = 1;
  repeated Field fields = 2;}

 message DataEntry {
  string path = 1;
  Datapoint value = 2;
  Datapoint actuator_target = 3;
  Metadata metadata = 10; }

enum Field {
  FIELD_VALUE           = 2;
  FIELD_ACTUATOR_TARGET = 3;
  FIELD_METADATA        = 10;
  (…)}
```

```
message Datapoint {
  google.protobuf.Timestamp timestamp = 1;
  oneof value {
    string string            = 11;
    bool bool                = 12;
    (…) } }
message Metadata {
  (…)
  oneof entry_specific {
    Actuator actuator   = 20;
    Sensor sensor       = 30;
    Attribute attribute = 40; }
}
```

COVESA

# VISSv3 vs kuksa.val.v2

```
service VISS {

rpc GetRequest (GetRequestMessage) returns (GetResponseMessage);

rpc SetRequest (SetRequestMessage) returns (SetResponseMessage);

rpc SubscribeRequest (SubscribeRequestMessage) returns (stream
SubscribeStreamMessage);

rpc UnsubscribeRequest (UnsubscribeRequestMessage) returns
(UnsubscribeResponseMessage);

}
```

```
service VAL {

 rpc GetValue(GetValueRequest) returns (GetValueResponse);

  rpc GetValues(GetValuesRequest) returns (GetValuesResponse);

  rpc ListValues(ListValuesRequest) returns (ListValuesResponse

  rpc Subscribe(SubscribeRequest) returns (stream SubscribeResponse)

  rpc SubscribeId(SubscribeRequestId) returns (stream SubscribeResponseId);

  rpc Actuate(ActuateRequest) returns (ActuateResponse);

  rpc BatchActuate(BatchActuateRequest) returns (BatchActuateResponse);

  rpc ListMetadata(ListMetadataRequest) returns (ListMetadataResponse);

  rpc PublishValue(PublishValueRequest) returns (PublishValueResponse

  rpc OpenProviderStream(stream OpenProviderStreamRequest) returns (stream
OpenProviderStreamResponse);

  rpc GetServerInfo(GetServerInfoRequest) returns (GetServerInfoResponse);

}
```

COVESA

# VISSv3 vs kuksa.val.v2 – Get Request

```
message GetRequestMessage {
 string Path = 1;
 optional FilterExpressions Filter = 2;
 optional string Authorization = 3;
 optional string RequestId = 4;
}

message FilterExpressions {
 message FilterExpression {
  enum FilterVariant {
   PATHS = 0;
   TIMEBASED = 1;
   (…)
  }
  FilterVariant Variant = 1;
  message FilterValue {
    optional PathsValue ValuePaths = 1;
    optional TimebasedValue ValueTimebased = 2;
    optional CurvelogValue ValueCurvelog = 5;
    (…)
}
FilterValue Value = 2;
 }
repeated FilterExpression FilterExp = 1; }
```

```
message GetValuesRequest {
  SignalID signal_ids = 1;
}

message SignalID {
  oneof signal {
    int32 id = 1;
    string path = 2;
  }
}
```

COVESA

# VISSv3 vs kuksa.val.v2 – Get Response

```
message GetResponseMessage {
  ResponseStatus Status = 1;
  optional SuccessResponseMessage
SuccessResponse = 2;
  optional ErrorResponseMessage
ErrorResponse = 3;
  optional string RequestId = 4;
  string Ts = 5;
  optional string Authorization = 6;}

enum ResponseStatus {
 SUCCESS = 0;
 ERROR = 1; }
message SuccessResponseMessage {
    optional DataPackages DataPack = 1;
    optional string Metadata = 2; }
```

```
message DataPackages {
 message DataPackage {
  string Path = 1;
  message DataPoint {
    string Value = 1;
     string Ts = 2; }
  repeated DataPoint Dp = 2; }
repeated DataPackage Data = 1;}

message ErrorResponseMessage {
  string Number = 1;
  optional string Reason = 2;
  optional string Message = 3; }
```

```
message GetValueResponse {
  Datapoint data_point = 1; }

message Datapoint {
  google.protobuf.Timestamp
timestamp = 1;
  oneof value_state {
    ValueFailure failure = 2;
    Value value = 3; } }

enum ValueFailure {
  UNSPECIFIED = 0;
  INVALID_VALUE = 1;
  NOT_PROVIDED = 2;
  UNKNOWN_SIGNAL = 3;
  ACCESS_DENIED = 4;
  INTERNAL_ERROR = 5; }
```

```
message Value {
  oneof typed_value {
    string string = 11;
    bool bool = 12;
    (…)  } }
```

# VISSv3 vs kuksa.val.v2 – Set Request

```
message SetRequestMessage {
  string Path = 1;
  string Value = 2;
  optional string Authorization = 3;
  optional string RequestId = 4;}
```

```
message ActuateRequest {
  SignalID signal_id = 1;
  Value value = 2; }

message SignalID {
  oneof signal {
    int32 id = 1;
    string path = 2; } }

message Value {
  oneof typed_value {
    string string = 11;
    bool bool = 12;
    (…)  } }
```

```
message PublishValueRequest {
  SignalID signal_id = 1;
  Datapoint data_point = 2;
}
message Datapoint {
  google.protobuf.Timestamp timestamp = 1;
  oneof value_state {
    ValueFailure failure = 2;
    Value value = 3; } }

enum ValueFailure {
  UNSPECIFIED = 0;
  (…) }
```

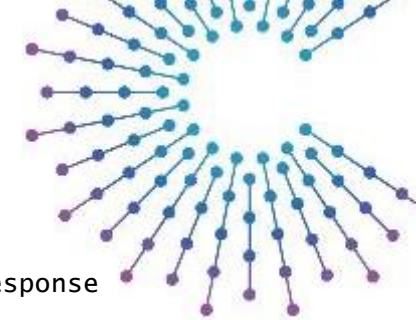
COVESA

# VISSv3 vs kuksa.val.v2 – Set Response

```
message SetResponseMessage {
  ResponseStatus Status = 1;
  optional ErrorResponseMessage ErrorResponse = 2;
  optional string RequestId = 3;
  string Ts = 4;
  optional string Authorization = 5;
}
```

```
message ActuateResponse {
}
```

```
message PublishValueResponse
{ }
```



# VISSv3 vs kuksa.val.v2 – Subscribe Request

```
message SubscribeRequestMessage {
 string Path = 1;
 optional FilterExpressions Filter = 2;
 optional string Authorization = 3;
 string RequestId = 4;
}
```

```
message SubscribeRequest {
  repeated string signal_paths = 1;
}
```

# VISSv3 vs kuksa.val.v2 – Subscribe Response

```
message SubscribeStreamMessage {
  SubscribeResponseType MType = 1;
  ResponseStatus Status = 2;
  message SubscribeResponseMessage {
    optional ErrorResponseMessage
ErrorResponse = 1;
    optional string SubscriptionId = 2;
    string RequestId = 3;
    string Ts = 4;
    optional string Authorization = 5;}
  optional SubscribeResponseMessage
Response = 3;
  message SubscribeEventMessage {
    string SubscriptionId = 1;
    message SuccessResponseMessage {
      DataPackages DataPack = 1; }
    optional SuccessResponseMessage
SuccessResponse = 2;
    optional ErrorResponseMessage
ErrorResponse = 3;
    string Ts = 4;}
  optional SubscribeEventMessage Event
= 4;}
```

```
message ErrorResponseMessage {
  string Number = 1;
  optional string Reason = 2;
  optional string Message = 3;}
enum SubscribeResponseType {
  RESPONSE = 0;
  EVENT = 1; }
message DataPackages {
  message DataPackage {
    string Path = 1;
    message DataPoint {
      string Value = 1;
      string Ts = 2; }
    repeated DataPoint Dp = 2;}
  repeated DataPackage Data = 1;
}
enum ResponseStatus {
  SUCCESS = 0;
  ERROR = 1; }
```

```
message SubscribeResponse {
  map<string, Datapoint> entries =
1; }
message Datapoint {
  google.protobuf.Timestamp
timestamp = 1;
  oneof value_state {
    ValueFailure failure = 2;
    Value value = 3; } }

enum ValueFailure {
  UNSPECIFIED = 0;
  INVALID_VALUE = 1;
  NOT_PROVIDED = 2;
  UNKNOWN_SIGNAL = 3;
  ACCESS_DENIED = 4;
  INTERNAL_ERROR = 5; }
```

```
message Value {
  oneof typed_value {
    string string = 11;
    bool bool = 12;
    (…)  } }
```

COVESA

# VIVA

## In-Vehicle Access

~~VSS~~

VISS³

- Create an in-vehicle API catering to southbound aspects of a VSS server
- This is not just “a transport” for VISS but its own API
- Allows each API variant to cater to its use cases without becoming a large “one size fits all” Frankenstein Middleware
- Make it simple enough there can realistically be more implementations (not “only” VissR, KUKSA)
- **May** or **may not** be under the VISS3 project (we suggest it is)
- **Should** be in COVESA