Vehicle data exchange protocols

This page collects input requirements for the data-exchange protocols in the context of defining the *vehicle-data reference architecture* as part of the CCS project.

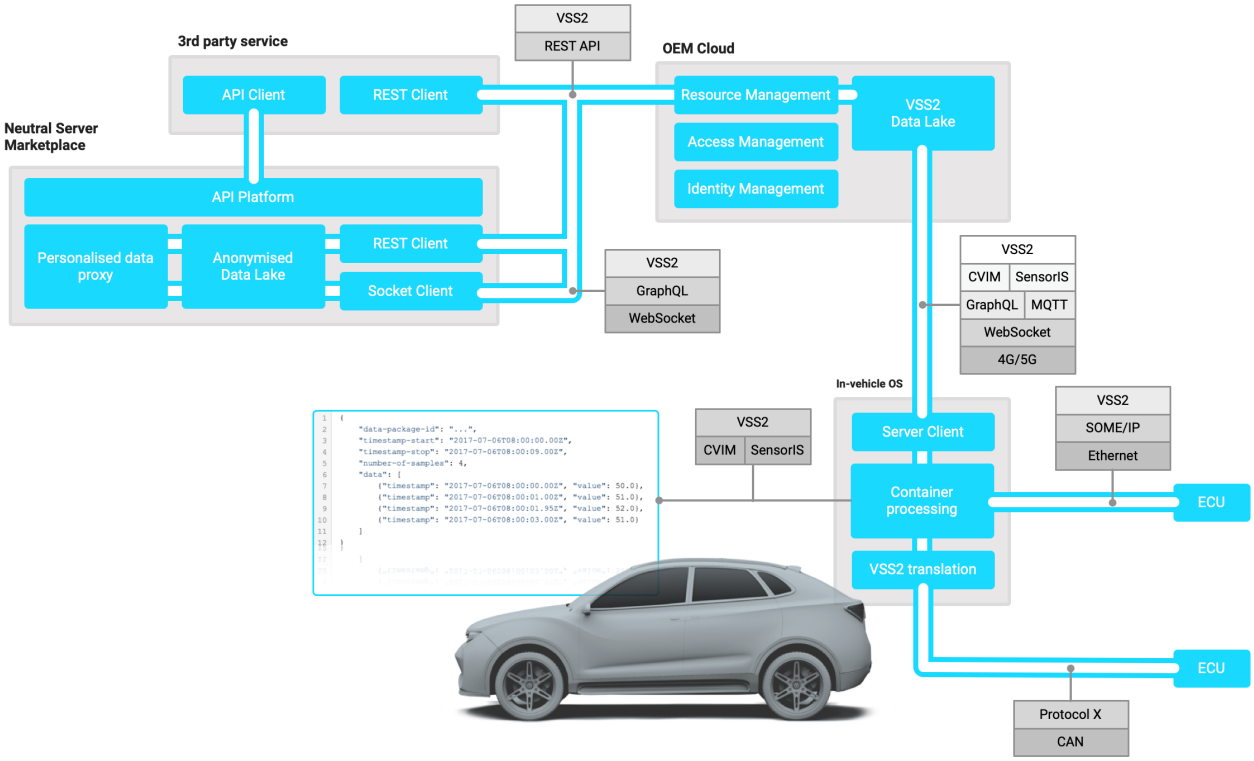
For the detailed planning of PoC development, please see CCS Reference Architecture - Work Breakdown Structure.

Purpose(s) of this page

- Define the features needed from *any* protocol, however it is realized in the end. The features may be implemented in protocol but might also be part of the defined semantics of the VSS taxonomy itself. (Semantics that are built into VSS automatically transfer to any and all access protocols including W3C-Gen2.
- Evaluate W3C "Gen2" protocol to understand its reasonable scope limits and/or add new feature requests to it.
- Understand where other protocols may complement.
- Define the list of protocols that meet the overall needs (and where each of them is to be used). This is a key piece of defining the reference architecture.

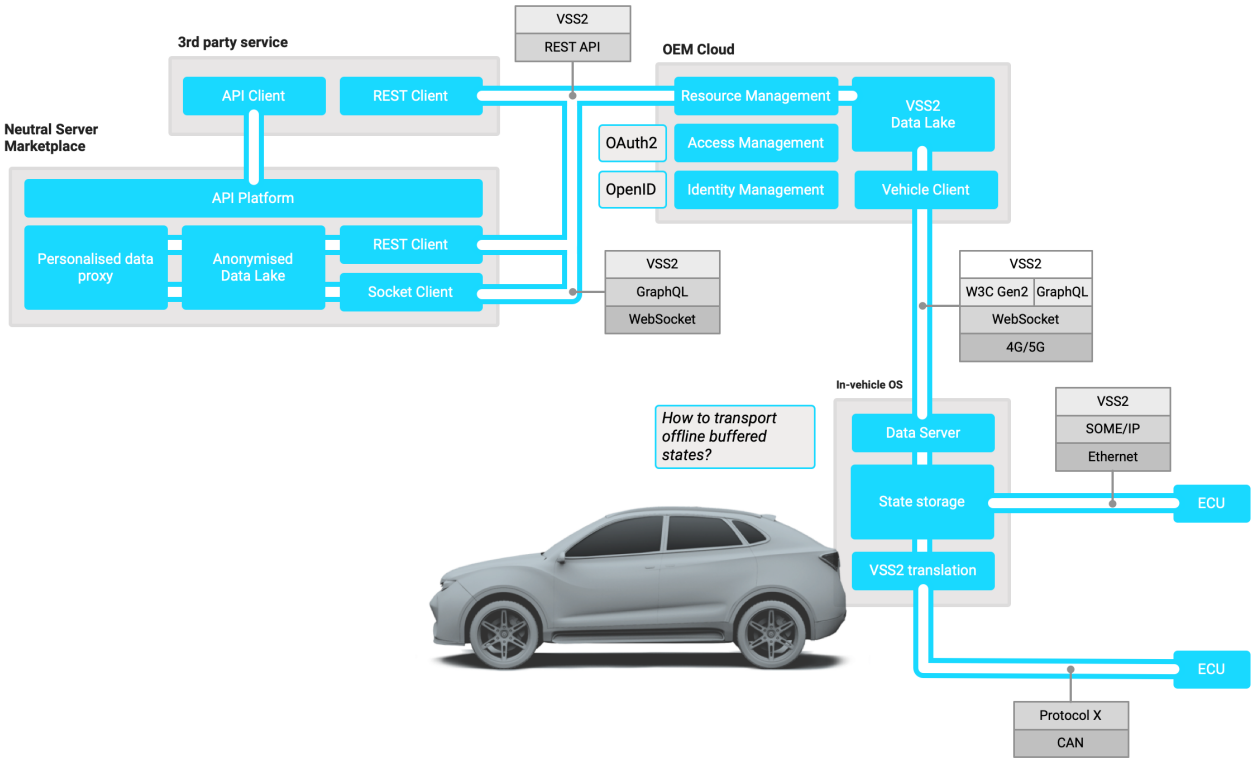
Communication framework

Communication Framework draft v1



- Starting from the in-vehicle data capturing, it is assumed that data can be transmitted by ECUs both in VSS2 and proprietary data models.
- Container processing in the vehicle is the process of encapsulating single data points into packages that can be either directly transmitted to the OEM cloud or buffered. The example data package snippet is derived from the Common Vehicle Information Model (CVIM) project where it was foreseen that a data package has meta information such as capturing frequency, start time, end time, number of samples.
- For data transmission from the vehicle to the OEM cloud a transport protocol like MQTT would be needed. Although not common in this context, GraphQL is introduced as an example, as the data transmission could potentially be implemented with a component that later can be reused for 3rd party interfaces.
- The OEM cloud includes components for Resources Management, Access Management and Identity Management in order to comply with the Extended Vehicle standardisation.
- Both 3rd parties and data marketplaces can access data from the OEM cloud using a REST API/GraphQL or alternatively socket connections. Socket connections are especially useful for marketplace interfaces with heavy traffic, both for personalised and anonymised data use cases.

Communication Framework draft v2



- "Container processing" as replaced with "State storage" in order to not confuse the term for containerised computing. In addition the vehicle interface was renamed from "Server client" to "Data server" as the vehicle is generating the data and the OEM cloud connects to the vehicle to request data.
- W3C Gen2 Core and Transport was introduced as the solution for vehicle-OEM cloud communication instead of CVIM/SensorIS and MQTT. Gen2 is designed for this purpose but opened a few requirement questions discussed in this document, especially about offline buffering.
- OAuth2 was added as the implementation guideline for OEM cloud Access Management.
- OpenID was added as the implementation guideline for OEM cloud Identity Management.

Features (ideas under development, and brainstorm)

- Queries that pre-filter on-demand or subscription data requests
    - See W3C gen 2 filtering proposal – consider additional query parameters to that list.
- A value measurements strategy for keeping more than the instantaneous current value. I.e. expect vehicles to keep a certain number of historical value measurements.
    - How is this actually defined and limited (since keeping *all* the history of *all* signals is an impossible requirement)
- A buffering mechanism and strategy for the case of temporary connectivity loss? Throw-away or buffer?
- Pre-processing (edge processing) before data is transferred
    - Statistical functions. Which ones?
        - e.g. Histogram.
    - Max/Min/Geometric Mean/Median value?
        - how to reset?
        - how to set over which time period?
- Request a sequence of measured values.
- Request a statistical result on a chunk of data.
- Discuss how to actually achieve the requests. Queries defined per-protocol (as W3C proposal)
    - or can it use VSS "virtual" node name conventions such as: Tree.Path.SignalX.MAX implies the maximum measured value of SignalX so far?
    - And/or what goes into VSS metadata?
    - Any application of VSS layers?

Requirements (well defined, this is what is required of a protocol)

- ...

Work to do

- Define (or reuse see CVIM, SENSORIS) a data values container format. I.e. the JSON format (presumably) that stores multiple measured values. The VSS defines only the entities that can be measured. W3C-Gen2 and VISSv1 defines a container format for one single value.
- Define (or reuse see SENSORIS) request formats for a "measurement job". Unlike the real-time request for a current value such as W3C-Gen2 /VISSv1 provides, or the continuous delivery of current values (Publish/Subscribe), this is something like "If the value X is higher than 5, measure value Y for the next 3 hours".
- For each of the feature ideas, consider how it can be met by:
    - protocol query parameters
    - VSS node path addressing
    - VSS node metadata
    - or other protocol feature