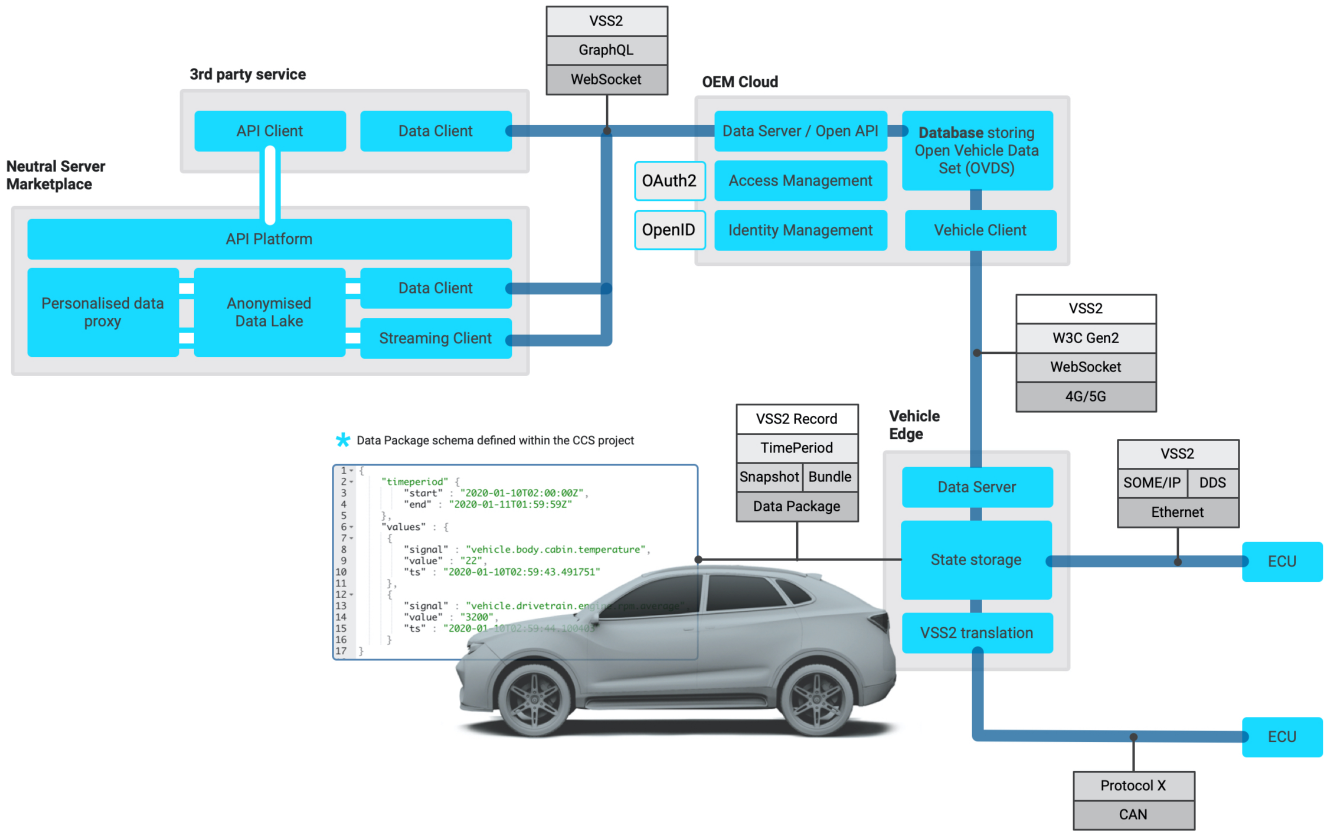
CCS Reference Architecture - Work Breakdown Structure

Implementation roadmap

- milestone 1 - Spring GENIVI Virtual Tech Summit (4-7 May 2021)
- milestone 2 - internal milestone (early Q3 - mid-July)
- milestone 3 - Fall All Member Meeting (Virtual ?) October ?
- milestone 4 - January 2022 – (planned for CES ?)

Communication framework architecture

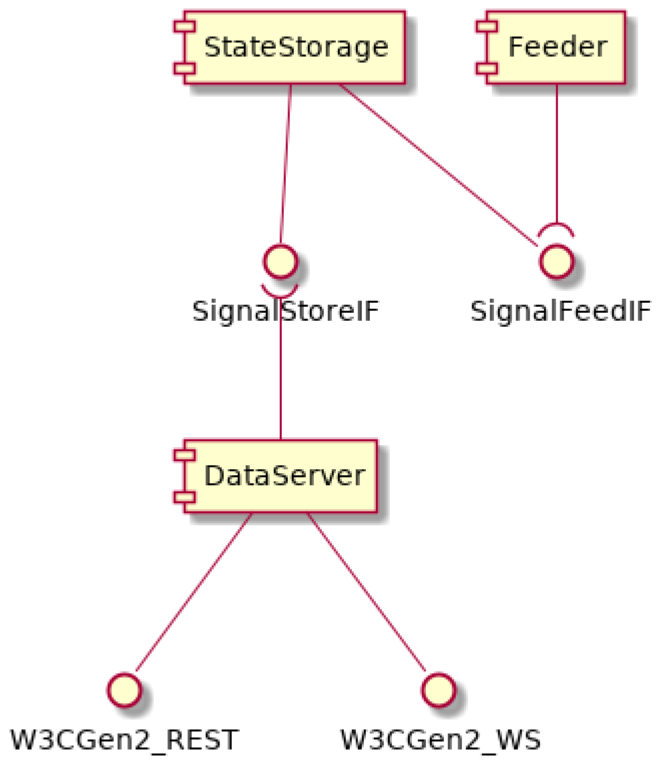
This page lists the different components and tasks to be completed for the architecture discussed on the Vehicle data exchange protocols page. The table references to the following diagram, with components that are considered in scope for the PoC highlighted with green color.



```
{
    "timeperiod" {
        "start" : "2020-01-10T02:00:00Z",
        "end" : "2020-01-11T01:59:59Z"
    },
    "values" : {
     {
        "signal" : "vehicle.body.cabin.temperature",
        "value" : "22",
        "ts" : "2020-01-10T02:59:43.491751"
     },
     {
        "signal" : "vehicle.drivetrain.engine.rpm.average",
        "value" : "3200",
        "ts" : "2020-01-10T02:59:44.100403"
     }
}
```

Components

Vehicle

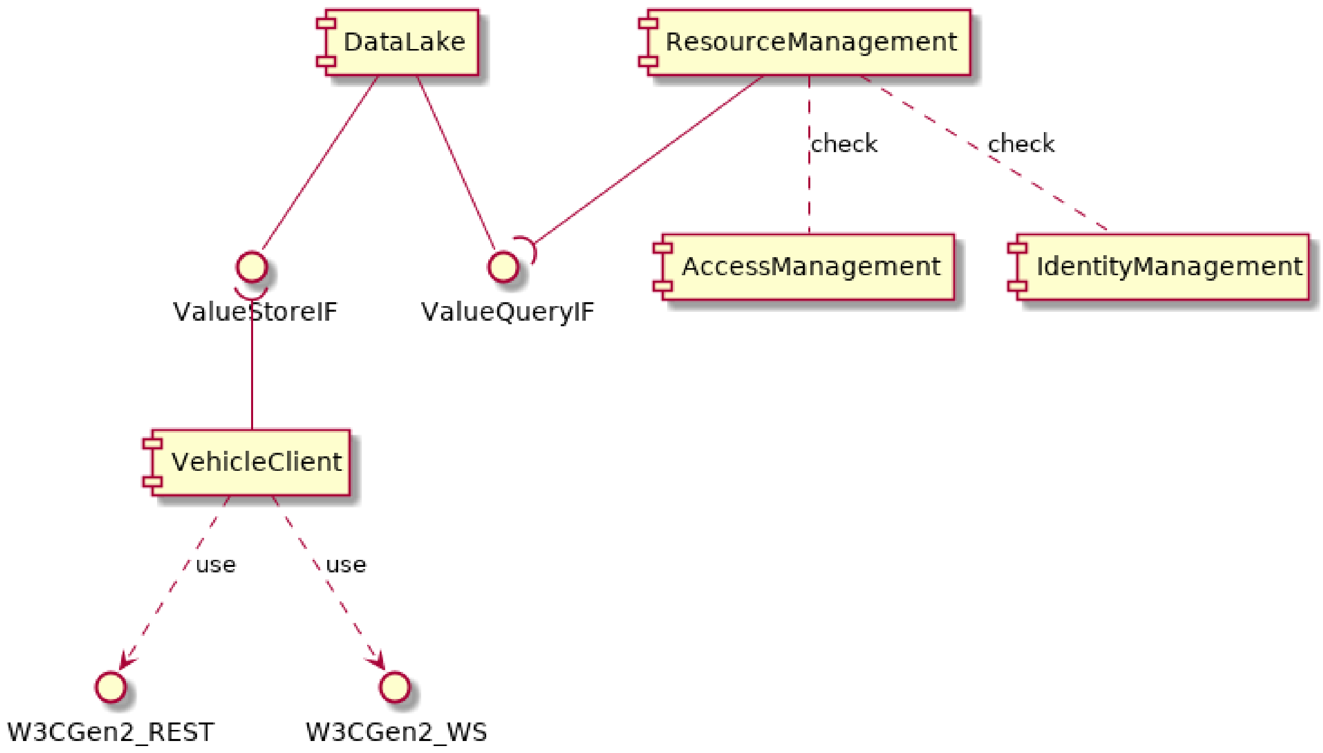


Definitions:

StateStorage and VSSFeeder is expected to be combined into one component in the implementation.

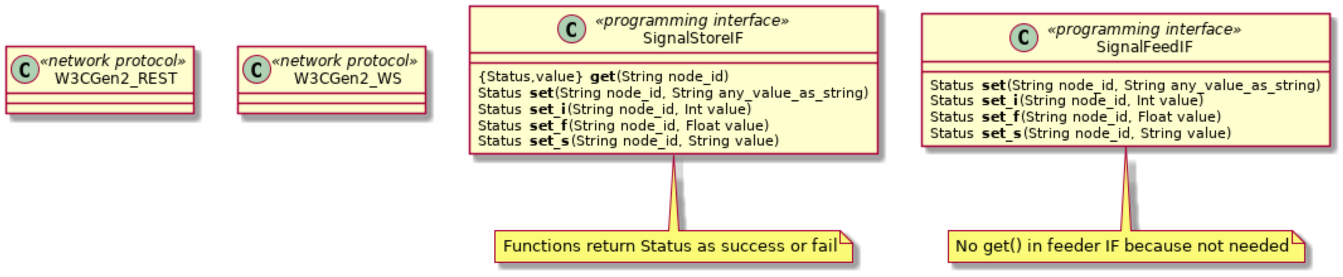
SignalStoreIF : Done by reading/writing **sqlite** database transactions, and using the notification feature to notify components that update occurred.
StateStorage: Essentially implemented by **sqlite** instance
SignalFeedIF: This is an internal implementation detail now since StateStorage/VSSFeeder is essentially one program

OEM Cloud



ValueStoreIF: ?
ValueQueryIF: ?

Interface details


«network protocol»
W3CGen2_REST
«network protocol»
W3CGen2_WS
«programming interface»
SignalStoreIF
{Status,value} **get**(String node_id)
Status **set**(String node_id, String any_value_as_string)
Status **set_i**(String node_id, Int value)
Status **set_f**(String node_id, Float value)
Status **set_s**(String node_id, String value)
«programming interface»
SignalFeedIF
Status **set**(String node_id, String any_value_as_string)
Status **set_i**(String node_id, Int value)
Status **set_f**(String node_id, Float value)
Status **set_s**(String node_id, String value)
Functions return Status as success or fail
No get() in feeder IF because not needed

(OEM Cloud) GraphQLDatabaseIF: (Interface between VSS2-Database Resource Mgmt (GraphQL server)

The interface is expected to be accessing the database itself. In order to give the GraphQL server full freedom to query data intelligently, it should connect directly to the SQL database.

Description of work

Apache License is more preferred by participants.

#	Component	Plan / Activities	Status of chosen software components (Implementation details of PoC)	Alt. SW components / Implementation for Production Systems ⭐ (WIP, future)	Owner
1	In-vehicle State storage (laptop simulator)	• Custom code + in-memory database (+ persistence) + connection to feeder • Either locally cached during uptime of system or • Stored and indexed in a database to allow access to historical data ◦ Selection of database ◦ Implementation of database schema ~~CCS-106~~ - Develop in-vehicle state storage DONE	• ~~Custom control code (Python or C++)~~ • ~~+ sqlite binding, OK for PoC~~ ✓ Implemented by a shared sqlite database file. The "API" is simply interacting with the database using SQL statements and/or sqlite bindings. Note: statestorage executable in a one-shot thing that sets up the database. The gen2-server accesses the actual database directly using sqlite binding. ~~CCS-107~~ - Develop proof-of-concept version of in-vehicle storage DONE	• Custom code ~~CCS-108~~ - Develop production-grade version of in-vehicle storage DONE	~~Keith~~ ~~CCS-109~~ - Reach out Kuksa project for in-vehicle state storage components DONE

2	In-vehicle VSS2 translation (a.k.a. VSS feeder) (laptop simulator)	• Implementation of SOME/IP client? • or simple simulator • or Vehicle Driving Simulator (LG? OpenDS? or GENIVI GitHub version?) ◦ = set up simulator, make it drive around track, get list of available signals, write code to convert signals to VSS... ◦ Signals are fed over DDS to AutoWare or Apollo autonomous driving stacks. ◦ Sim needs a high-end graphics machine • Look if VSI or VSD should play a part • New: live_simulator which is feeding a real-time playback of existing timestamped data. • Implement CAN + VSS translation support (e.g. reuse from Bosch code project) CCS-110 - Develop In-vehicle VSS2 translation (a.k.a. VSS feeder) IN PROGRESS	• ✓ live-simulator implemented by Ulf feeds data taken from an existing "ovds" database and feeds it into statestorage. • Example ovds database to feed live simulator now exists. Additional options: • ~~Custom code, feeding simple simulated data~~ • ⚠ Implement CAN + VSS translation support (e.g. reuse from Bosch code project) • Then (future) run full vehicle driving simulator continue driving-simulator track and "playback" simulator in parallel. CCS-111 - Develop proof-of-concept version of in-vehicle VSS2 translation IN PROGRESS		*not assigned* ~~CCS-112~~ - Reach out Kuksa project for in-vehicle VSS2 translation / VSS feeder components DONE CCS-149 - Analyze how to run a driving simulator to generate vehicle data IN PROGRESS
3	In-vehicle Data Package	• Collecting VSS2 data into snapshots and bundles according to Data serialization / value formats • Presenting data packages to the Data server • Possibly closely related to the State storage schema ~~CCS-113~~ - Develop in-vehicle data packaging DONE	~~*Start with results of Apache NiFi track to fulfil this need.*~~ *(no resource / no concrete plan for NiFi at the moment)* ❌ *No recent progress* CCS-114 - Develop proof-of-concept version of in-vehicle data packaging TO DO	✓ VISS specification defines how to fetch "historical" collected data. ✓ VISS server W3C_VehicleSignalInterfaceImpl implementation will record data (if an interface is triggered from the vehicle, use case is "going out of mobile service area"). Also, the messages for fetching data (according to VISS) is implemented. CCS-115 - Develop production-grade version of in-vehicle data packaging TO DO	~~Gunnar~~ Ulf
4	In-vehicle Data server (laptop simulator)	• Data server implementation for W3C Gen2 ◦ Reference implementation exists in GitHub MEAE-GOT ◦ Ulf can work together with someone how to connect to an existing API of "state storage" ◦ (also talk to Kuksa project - VISS+REST server) CCS-116 - Develop in-vehicle data server TO DO	Use server directory from W3C_VehicleSignalInterfaceImpl ~~CCS-117~~ - Develop proof-of-concept version of in-vehicle data server DONE Gen2 implementation now uses ovds.db file. ✓ On-demand requests are fetched from database. Timed subscriptions are also supported, i.e. send updates at regular intervals. ⚠ There is not yet implementation of a trigger to the gen 2 server from the database when a new value is written (SQLite supports trigger in theory).		Ulf ~~CCS-154~~ - Reach out Kuksa project for in-vehicle data server components DONE
5	OEM Cloud Vehicle client	• W3C Gen2 protocol (VISS Websocket Pub/Sub) • Options: ◦ Ulf wrote the client using Go-lang, stored in MEAE-GOT/W3C. ◦ Sanjeev involved in writing a client using Javascript. ◦ Curl script ◦ Custom code + HTTP library (e.g. written in python) custom code + libcurl binding ◦ No clear answer. Depends on use-case. What is the rate of data for example? • This program shall also store the data into the data lake program (or directly into the database used as data lake) CCS-118 - Develop OEM cloud vehicle client TO DO	Written in Go, some similar code as in W3C Gen 2 reference server ~~CCS-119~~ - Develop proof-of-concept version of OEM cloud vehicle client DONE ✓ Demo includes this function already.. In ccs-client repo. Client reads data from data server in vehicle via the VISSv2 protocol, and writes it into the OVDS database. ✓ Vehicle client can be set to either to HTTP Get or WebSocket subscription feature.	CCS-120 - Develop production-grade version of OEM cloud vehicle client TO DO	Ulf

6	OEM Cloud VSS2 database	• Selection of libraries and database ◦ Define database schema Start with Ulf's proposal for DB schema ~~CCS-121~~ - Develop OEM cloud VSS2 database DONE Postgres was discussed. Use SQLite first. ~~Currently the translation path to UUID. Translation is in a separate database, compared to the signal database. A future possibility is two tables in a single database, making it possible to use SQL JOIN statements.~~ now using Path as ID. This also simplifies supporting multiple VSS versions (where paths might differ) at the same time.	• ~~Custom control code (Python or C++)~~ implemented in Go. • + Sqlite binding, OK for PoC, or other database such as postgresql. ◦ Need to define database schema first proposal is here ~~CCS-122~~ - Develop proof-of-concept version of OEM cloud VSS2 database DONE ✓ Implemented in In ccs-client repo. This is the OVDS server. It exposes a REST protocol that is used by the client. ~~which may be easier since it is a single operation and not having to look into both databases. (See JOIN idea on the left for alternative).~~ REST protocol can also deliver full time series. ⚠ Should we split up the software into more logical repositories? Agreed, but not as urgent.	In production more likely to be an object store database instead of a RDBS. Sanjeev looking at Apache ecosystem and Hortonworks /Cloudera platform capabilities. ~~CCS-124~~ - Investigate Apache NiFi/Spark and other technologies for data architecture DONE	Ulf ~~CCS-125~~ - Reach out Geotab to learn the Geotab platform capabilities DONE
7	OEM Cloud Identity management	• Implementation of end-user login and authentication using **OpenID** ◦ Lots of candidates. Responsible implementer should propose a good alternative. CCS-126 - Develop OEM cloud identity management TO DO	❌ *TODO* *Later. (Programming language / technology preference could be influenced by the programmer)*		--
8	OEM Cloud Access management	• Implementation of authorization between end-user and 3rd party application or Neutral Server using **OAuth2** • Iyyaz points to the following code ◦ https://www.ory.sh/hydra/ taken from https://oauth.net/code/ **CCS-127 - Develop OEM cloud access management** IN PROGRESS	❌ *TODO* *Later. (Programming language / technology preference could be influenced by the programmer)*		--
9	OEM Cloud Resource Management = Data server API	• Implementation of basic API management • Resource Mgmt is the ExVe naming. • Implementation of a GraphQL API using the VSS2 schema • Interface to the client, is of course defined by GraphQL+schema. • Interface to database (GraphQLDatabaseIF) is described above CCS-128 - Develop OEM cloud resource management (= Data server API) TO DO	GraphQL Apache Apollo? ~~CCS-129~~ - Develop proof-of-concept version of OEM cloud data API DONE Located in vss-graphql repo • ✓ Has one schema. • ✓ Proposal (PR in vss-tools) for GraphQL schema generator now exists. • ✓ Has Apollo server in docker **TODO** ❌ Needs to implement the connection from GraphQL (Apollo) to database. It could use the REST protocol of the OVDS server, or directly SQLite database. • **^^ Important to fix!!! ^^**		(Alexander Domin) ~~CCS-130~~ - Reach out to BMW for the GraphQL example DONE **JIRA TODO:** Implement the connection between GraphQL and OVDS database.

10	Neutral Server Data Marketplace	• Separate instance that consumes the OEM Cloud OAuth2 and GraphQL APIs ◦ We could implement a very simplistic neutral server using the published API of High Mobility, but an open implementation. ◦ Security, consent and other complicates things. Leave those details out. CCS-131 - Develop neutral server market place TO DO	CCS-132 - Develop proof-of-concept version of neutral server market place TO DO In vss-graphql-client-swift repository ✓ Has programming framework/example (in SWIFT) to access data via graphql. ▪ *This could show a way for 3rd party applications to access the OEM /GraphQL interface directly but possibly a more limited REST-API is more realistic for 3rd part app access* ⭐ **TODO**: Expose a neutral-server API to third party applications. Nothing in particular to develop It would just be a proxy (for the GraphQL and/or REST) – same technologies as our current OEM connection would be used by the Neutral Server. The API would be quite transparent if VSS is exposed directly, ~~*but the Neutral Server API might expose different functions also.*~~ ? *But enriched functions / anonymized aggregated data might be provided by Neutral Server.*		Kevin
11	3rd Party Application	• Example applications exist on High Mobility's GitHub • One instance that consumes the Neutral Server/Data Marketplace APIs ◦ Use an example application from HM ◦ For example an app that just shows the data (written in Node.js) • One instance that consumes the OEM Cloud OAuth2 and GraphQL APIs ◦ Modify the application to use the OEM API directly CCS-133 - Develop 3rd party application IN PROGRESS	~~CCS-134~~ - Develop proof-of-concept version of 3rd party application DONE ✓ New sample app Connecting directly to OEM cloud for now, via GraphQL. For connecting to Neutral server instead: ▪ ~~Example/standard API work needed. The example apps are using High-Mobility's API.~~ See above ⚠ See above, it is first required to define Neutral Server API.		Kevin

previously captured notes on AWS. See 2021-May AMM presentation for more up to date presentation by Kevin.

Cloud Infrastructure Tools (for production-grade design proposals)

AWS tools – which of these might be useful?

- Lambda = "serverless" tasks
- Greengrass – edge/device platform
- Greengrass core = data processing software components
- Greengrass converters – are these useful for data conversion (VSS)
- Greengrass general = runtime platform, SOTA, etc.
  - package VISS Data Server and Statestorage into
- Timestream – time series database, is it an alternative to Influx?
  - Executes with serverless approach (with some kind of persistent storage behind of course)
- Kinesis Datastreams – data stream transfer, is it an alternative to Apache NiFi?
- Kinesis Firehose – capture and modify data, is it similar to Apache Spark?
- RDS – Relational DB service
- DynamoDB – key/value DB