Evolving Vehicle Signal Specification Usage in AGL

AGL All Member Meeting

July 12, 2023

Scott Murray (scott.murray@konsulko.com)

About me

- Linux user/developer since 1994
- Embedded Linux developer since 2000
- Principal Software Engineer at Konsulko Group since 2014
- Working on AGL on contract since 2016
  - Yocto Project maintenance
  - Demo development, integration, and maintenance

Agenda

- Vehicle Signal Specification (VSS)
- Vehicle Information Server (VIS) Service
- KUKSA.val
- KUKSA.val & VSS usage in AGL
- Future Development

Caveat

- Some content duplicated from Berlin AMM and EOSS presentations…
- … but updates with respect to future planning

Vehicle Signal Specification

- Vehicle Signal Specification (VSS)
- Open source project started under COVESA
  - https://github.com/COVESA/vehicle_signal_specification
- Major component of COVESA/W3C Common Vehicle Interface Initiative Project (CVII)
- Developed by BMW, Volvo, Bosch, JLR, etc.
- Associated standardizations underway at W3C

Vehicle Signal Specification (continued)

- Hierarchical signal schema
  - e.g. Vehicle.Powertrain.CombustionEngine.Speed
  - Typically stored in JSON format, other formats also possible
  - Generated from higher level metadata (vspec files)
  - Metadata tools (vss-tools) available
- Schema currently at version 4.0
  - Major recent addition is support for structured datatypes for signal values

VSS Signal vspec Example

```
Speed:
  datatype: float
  type: sensor
  unit: km/h
  description: Vehicle speed.
```

From:
https://github.com/COVESA/vehicle_signal_specification/blob/master/spec/Vehicle/Vehicle.vspec

VSS Signal JSON Example

```
"Speed": {
    "datatype": "float",
    "description": "Vehicle speed.",
    "type": "sensor",
    "unit": "km/h",
    "uuid": "efe50798638d55fab18ab7d43cc490e9"
},
```

From:

https://github.com/eclipse/kuksa.val/blob/master/data/vss-core/vss_release_3.1.1.json

Vehicle Information Service

- Vehicle Information Service (VIS) Specification
- Open source project started under COVESA
- Developed by BMW, Volvo, Bosch, JLR, etc.
- Standardization process underway with W3C
- HTTPS, Websocket, and MQTT APIs to access VSS signals
  - Get, Set, Subscribe, etc.
- Reference implementation in Go
  - https://github.com/w3c/automotive-viss2
- Also a partial implementation in C++ in KUKSA.val

KUKSA.val

- https://github.com/eclipse/kuksa.val
- Server primarily developed by Bosch, with contributions from others
- Implements most of VIS v1 and some of VIS v2
- Also extends VIS with a gRPC version of the API
  - KUKSA.val "databroker"
- JSON web token (JWT) authorization mechanism
- Python and Go client libraries, with examples
- Python feeder clients to push signal data

KUKSA.val (continued)

- C++ server provides mechanism for modifying or adding new signals via overlay JSON files
- Databroker does not have its own overlay scheme
  - Overlays done at vspec level and full VSS tree generated with vss-tools
- Currently used in AGL demo for steering wheel switches and a few other signals
  - meta-agl-demo/recipes-connectivity/vss/vss-agl/agl_vss_overlay.vspec

VSS vspec Overlay Example

```
Vehicle.Speed:
  datatype: float
  type: sensor
  dbc:
    signal: PT_VehicleAvgSpeed
    interval_ms: 100
```

From:
https://git.automotivelinux.org/AGL/meta-agl-demo/tree/recipes-connectivity/vss/vss-agl/agl_vss_overlay.vspec

KUKSA.val Feeders

- DBC feeder
  - Pushes selected CAN data to configured VSS signals
  - Uses DBC (CAN database) file for CAN signal definitions
    - DBC format comes from Vector, but is documented
  - YAML configuration file for CAN to VSS signal mapping in 0.2.x
  - Since 0.3.0 uses signal annotations applied using vspec overlays
- GPS feeder
  - Pushes location data from gpsd
- Replay feeder
  - Can be used to replay a stream of VIS updates

KUKSA.val & VSS Usage in AGL

KUKSA.val

- KUKSA.val server was initially added in the Marlin (13.0) release
  - Replacement for agl-service-can-low-level and agl-service-signal-composer
- A recipe to build the server is carried in the meta-agl-demo layer
  - Custom AGL VSS generated by applying overlay vspec file on top of base VSS
- The DBC feeder is also built and packaged with a recipe in meta-agl-demo
  - Uses DBC file with minimal "agl-vcar" CAN signal definitions

KUKSA.val & VSS Releases in AGL

- Magic Marlin (13.0) - Spring 2022
  - KUKSA.val 0.2.1 integrated
  - VSS 2.2
  - Using C++ server with VIS WebSocket API
  - kuksa-dbc-feeder CAN feeder for demos
- Nifty Needlefish (14.0) - Summer 2022
  - Upgraded to KUKSA.val 0.2.5 and VSS 3.0
- Optimistic Octopus (15.0) - Spring 2023
  - Upgraded to KUKSA.val 0.3.1 and VSS 3.1.1
  - Switch to using vspec overlay with vss-tools

Prickly Pike (16.0.0)

- Still using KUKSA.val 0.3.1
  - But newer post 0.3.1 commit to pick up some databroker improvements
- Databroker included in images for evaluation and testing
  - Tests Rust 1.68 Yocto mixin layer required for building databroker
- Application conversion to use KUKSA.val "VAL" gRPC API against databroker still in progress
  - Will not make 16.0.0…

VSS/VIS Using Applications

- Two categories
  - Pure VSS signal observers
    - e.g. dashboard applications
    - Read "sensors" in VSS terminology
  - VSS signal actors
    - e.g. services like agl-service-hvac
    - Implement "actuators" in VSS terminology
- KUKSA.val optionally extends VIS
  - Set actuator target value -> actuator sets sensor value
  - Not a hard requirement, but model somewhat assumed when using databroker

agl-service-hvac

- Original application framework based code leveraged to implement a service backend for VSS HVAC signals
- Currently WebSocket client via Boost library
- Listens for fan speed and temperature actuator changes
- Pushes fan speed updates out to HVAC controller via CAN
- Pushes temperature updates out to LEDs in demo unit via GPIO

agl-service-audiomixer

- In legacy application framework provided main and per-role volume controls
  - Sits on top of WirePlumber API
- With the removal of the application framework, code leveraged to implement a new service backend for VSS volume signal
  - Vehicle.Cabin.Infotainment.Media.Volume
- VSS does not currently have finer grained volume control signals
  - Plan is to perhaps expose a gRPC API for those

Qt Demo Applications

- VSS signal using applications:
  - Homescreen
  - Dashboard
  - Cluster dashboard
  - HVAC
  - Navigation
- VIS WebSocket client code is abstracted in libqtappfw-vehicle-signals Qt library to reduce code duplication

Flutter Demo Applications

- VSS signal using applications:
  - Homescreen
  - Dashboard
  - Cluster dashboard
  - HVAC
- WebSocket client code is currently duplicated in each application, but is not large

Future Development

VSS

- VSS 4.0 released in late May
- Has some known impact with signal name changes
  - Left/Right changes to Driver/Passenger
- Structure support does not have an impact (yet)
  - Current upstream plan is existing VSS signals will not be changed to use structs
- We have VSS version flexibility with KUKSA.val…
  - VSS schema is a configuration option
  - Currently using 3.1.1 for Pike 16.0.0
  - Will switch to 4.0 for Quillback

VSS (continued)

- There has been some discussion upstream on whether standardized sets of VSS signals are required
  - Idea currently does not seem to have much traction
  - But does seem possible down the road
- Ongoing discussion to expand scope
  - Expanding capabilities to ease integration with cloud data services
  - Potential impact currently unclear
    - So far seems to be a strong desire to keep VSS simple

KUKSA.val

- Upstream has deprecated original C++ server for Rust based databroker
- Databroker
  - Does not implement VIS WebSocket API support
  - "VAL" gRPC API is similar to VIS v1
  - Currently built into images, but not directly used
- Complete switchover to using databroker in Quirky Quillback (17.0)
  - Perhaps backport changes to Pike 16.x for CES 2024
- May start moving VSS & KUKSA.val recipes to meta-agl in Quillback
  - Would simplify use by downstream users

KUKSA.val (continued)

- Python "kuksa-viss" VIS proxy for databroker recently created
  - Some potential to ease migration to databroker
  - Plan is to at least add recipe in Quillback to have it available
- Mock service recently added to kuksa-services example repository
  - https://github.com/eclipse/kuksa.val.services
  - Python API for mocking actuators and sensors for application testing
  - Some investigation required to see if an example integration is worthwhile

Applications

- Plan is still to convert all existing VSS/VIS clients to the databroker gRPC API
- Qt demo conversion is prototyped
  - Slight blocker in getting TLS support debugged
- Flutter demo conversion not yet started
  - Possibly will use kuksa-viss proxy in the short-term
  - Possible code contribution from Harman
  - Longer term may investigate using native Rust gRPC
- Some possible efficiency gains to be had by refactoring apps to better take advantage of API

Applications (continued)

- Still need to investigate API authorization schemes
  - Currently just pointing at tokens in filesystem
  - systemd credentials mechanism, OAuth?
- May need to consider TLS certificate generation
  - KUKSA.val upstream plans to move to requiring TLS by default and also to stop shipping default certificates
  - Unclear whether our shipping default demo certificates is going to be problematic or not
- Documentation!
  - Aiming to get integration documentation into Pike 16.0.x point release

More Information

More information

- Vehicle Abstraction with Eclipse Kuksa and Eclipse Velocitas - Sven Erik Jeroschewski, Bosch Digital

  https://static.sched.com/hosted_files/aglammspring2023/5c/VehicleAbstractionwithEclipseKuksaandEclipseVelocitas.pdf

  https://www.youtube.com/watch?v=LHJnBKb1Ta8

- Vehicle Signaling Specification and KUKSA.val in AGL

  https://static.sched.com/hosted_files/aglammspring2023/8f/VSS%20and%20KUKSA.val%20in%20AGL.pdf

  https://www.youtube.com/watch?v=RhSocQDu_DY

Questions?