Using CAN Services with AGL

AGL Training Class
October 20, 2022
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

• AGL CAN services pre-Marlin
• AGL CAN services today?
• Vehicle Signaling Schema (VSS), Vehicle Information Server (VIS), and KUKSA.val
• VSS/VIS Examples
• CAN to VSS Configuration
• VSS Client Example: agl-service-hvac
• Future development
CAN Services pre-Marlin

• In the legacy application framework vehicle signaling was enabled with:
  • agl-service-can-low-level
  • agl-service-signal-composer
• API bindings developed by IoT.bzh for AGL
• Used in the demo platform to implement dashboard, HVAC, and steering wheel demonstrations
OpenXC XML-based CAN signal definitions
  - OpenXC was a Ford open-source project
  - No obvious user community
Code generator to create signal definitions plugin for API service
  - Based on a couple of OpenXC libraries
  - API with signal subscription, write, etc.
Some degree of J1939 and ISOTP support
  - Not fully exercised in upstream AGL
  - ISOTP not enabled by default in build
agl-service-signal-composer

- Service to abstract signals from multiple sources
  - CAN, GPIO, etc.
- Configuration via JSON files
  - Some provision for runtime loading of configuration
- Plugin architecture and signal processing heavily tied to legacy application framework architecture
AGL CAN Services Today?

• During the discussions around replacing the legacy application framework there was no obvious candidate to replace agl-service-can-low-level
  • AGL members all have their own implementations
  • No FOSS project in the space with a significant userbase
• No desire to invest the effort to build a similar API without member engagement
  • Focus on technology demonstrations using Linux SocketCAN
AGL CAN Services Today? (continued)

• Investigation in 2021 found emerging Vehicle Signal Specification (VSS) and Vehicle Information Service (VIS) Server standards
• Initial investigation suggested that a VIS server would provide capabilities similar to agl-service-signal-composer
• Vehicle Signal Specification
• Open source project started under COVESA
  • https://github.com/COVESA/vehicle_signal_specification
• Developed by BMW, Volvo, Bosch, JLR, etc.
• Hierarchical signal schema in JSON
  • Other formats also possible
• Schema currently at version 3.0
VISS

- Vehicle Information Service (VIS) Server
- Open source project started under COVESA
- Developed by BMW, Volvo, Bosch, JLR, etc.
- Standardization process underway with W3C
  - [https://w3c.github.io/automotive/vehicle_data/vehicle_information_service.html](https://w3c.github.io/automotive/vehicle_data/vehicle_information_service.html)
- Websocket API to access VSS signals
- Reference implementation in Go
  - [https://github.com/w3c/automotive-viss2](https://github.com/w3c/automotive-viss2)
- Also an implementation in C++, KUKSA.val
KUKSA.val

- [https://github.com/eclipse/kuksa.val](https://github.com/eclipse/kuksa.val)
- Primarily developed by Bosch, with contributions from others
  - Under active development
- Extends VISS with a gRPC version of the API
- JSON web token (JWT) authorization mechanism
- Python and Go client libraries, with examples
- Example feeder clients to push signal data
  - Implemented in Python using client library
• Provides mechanism for adding new signals via overlay JSON files
  • Used in AGL demo for steering wheel switch and a few other signals
  • See:
    https://git.automotivelinux.org/AGL/meta-agl-demo/tree/recipes-connectivity/kuksa-val/kuksa-val-agl/00-agl_vs_s_overlay_2.2.json?h=needlefish
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
- **GPS feeder**
  - Pushes location data from gpsd
- **Replay feeder**
  - Can be used to replay a stream of VIS updates
VSS Example

• ABS error signal (Vehicle.ADAS.ABS.Error):

"Error": {
    "datatype": "boolean",
    "description": "Indicates if ABS incurred an error condition. True = Error. False = No Error."
    "type": "sensor",
    "uuid": "cd2b0e86aa1f5021a9bb7f6bda1cbe0f"
},
VSS Example Breakdown

• Data types
  • boolean, float, integer, unsigned integer, string
• Signal types
  • sensor = input
  • actuator = output
• In VIS specification sensors are read-only!
  • However, KUSKA.val allows writing to sensors to enable more flexible broker-like architectures
  • KUKSA.val also adds the concept of target values for actuator signals to separate current versus target values
VIS Get Example

• See more at https://w3c.github.io/automotive/vehicle_data/vehicle_information_service.html#message-structure

• Request

```json
{
    "action": "get",
    "path": "Signal.Drivetrain.InternalCombustionEngine.RPM",
    "requestId": "8756"
}
```
• Reply

```json
{
    "action": "get",
    "requestId": "8756",
    "value": 2372,
    "timestamp": 1489985044000
}
```
CAN to VSS Configuration
DBC Feeder Configuration

1. CAN signal configuration in DBC file
2. CAN signal to VSS signal configuration in mapping.yaml
3. DBC feeder configuration in .ini file
CAN Signal DBC configuration

- Create with Vector's tools
- Create by hand
- AGL using:
CAN to VSS configuration

- DBC feeder mapping of CAN signal from DBC file to VSS signal
- Some simple transforms possible: value mapping, math (e.g. for scaling), see:
- AGL using:
DBC Feeder .ini Configuration

- DBC feeder configuration file to specify:
  - DBC file
  - mapping yaml file
  - CAN device
  - KUKSA.val server location
  - KUKSA.val authorization token

- AGL using:
DBC Feeder config.ini

[general]
# use case:
# switch between databroker and kuksa
# default kuksa
usecase = kuksa
# VSS mapping file
mapping=/etc/kuksa-dbc-feeder/mapping.yml

[kuksa_val]
# kuksa VSS server address
server=wss://localhost:8090
# JWT security token file
token=/etc/kuksa-dbc-feeder/dbc_feeder.json.token

[can]
# CAN port
port=can0
# Enable SAE-J1939 Mode. False: ignore
J1939=False
# DBC file used to parse CAN messages
dbcfile=/etc/kuksa-dbc-feeder/agl-vcar.dbc
Using Your Own Configuration?

- For testing during development, perhaps start by editing files in /etc/kuksa-dbc-feeder on target.
- To apply new configuration for your own demos, either:
  - Potentially submit change against meta-agl-demo upstream.
  - Replace configuration files with a bbappend against kuksa-dbc-feeder if you have your own local layer.
VSS Client Example: agl-service-hvac
agl-service-hvac

• In legacy application framework provided a simple temperature and fan speed API
  • Used by the Qt demo HVAC application
  • Originally used SocketCAN directly to drive HVAC controller
  • Was converted to use agl-service-can-low-level API
• With the removal of the application framework, code leveraged to implement a new service backend for VSS HVAC signals
VSS HVAC Schema

- VSS includes a full set of HVAC signals
  - 4 rows
    - left and right sides
      - fan speed and direction
      - temperature
  - Example signals:
    - Vehicle.Cabin.HVAC.Station.Row1.Left.FanSpeed
    - Vehicle.Cabin.HVAC.Station.Row2.Right.Temperature
Implementation

- [https://git.automotivelinux.org/apps/agl-service-hvac/](https://git.automotivelinux.org/apps/agl-service-hvac/)
- Currently WebSocket client via Boost library
  - Plan is to migrate to KUKSA.val gRPC API
- Listens for Row1 Left and Right fan speed and temperature actuator changes
- Pushes fan speed updates out to HVAC controller via CAN
  - Switched back to doing direct SocketCAN writes
- Pushes temperature updates out to LEDs in demo unit via GPIO
agl-service-hvac Source Walkthrough
Future Development
CES 2023 Development

- Potentially convert agl-service-hvac, agl-service-audiomixer, and libqtappfw client to the KUKSA.val gRPC API
- The steering wheel demonstration has been converted from LIN to CAN, finish integration
Post-CES Plans

• Complete transition to KUKSA.val gRPC API
• Investigate options for authorization token handling
  • Currently installing with applications during build as stopgap
  • Aim is to demonstrate something more useful for production
    • systemd credentials management?
    • OAuth?