Which feature gets the "Volume Up" button press?
Vehicle state dictates use cases

"Volume Up" pressed:

• **Are we in reverse?**
  Increase Parking Assist volume.

• **Is Navigation active**
  Increase Nav Volume.

• **Are we in an ongoing phone call?**
  Increase Phone Volume.

• **None of the above?**
  Increase Media Volume.
We need to move signal logic out of the code...

```cpp
if (signal.ID() == 0x4711 &&
    !drivetrain_obj.IsInReverse() &&
    GetActiveApplication() == "nav")
    media_obj.VolumeUp();
else if (...)
```
… into rule files
Benefits

• **Manage variance complexity**
  One system to handle use cases across vehicle lines, configuration, and regional legislation.

• **Provides testing framework**
  The rule specification driving the signal distribution can also be used to validate signal sequences and timing during test.

• **Updateable via OTA**
  Updated rule specifications can be pushed over the air, without the complexities of SOTA, to adjust fleet behavior.
Project Goals and Objectives

• **Explore signal transformation space**
  Can signal transformation be used to extract call flow logic from features?

• **Understand boundaries between state machines and features**
  What do we encode in signal manager and what stays in the feature?

• **Prepare for production-level implementation**
  Lessons learned fed into potential production variant of the manager.
What is a VSM rule?

• Monitors vehicle signals for specific conditions ...
  - condition: transmission.gear == 'reverse'

• ... and then emits signals or makes API-calls on its own
  emit:
    signal: lights.external.backup
    value: true

• All encoded as YAML
Conditions can be monitored in sequence ...

sequence:
- condition: \texttt{transmission.gear} == 'reverse'
- condition: \texttt{camera.backup.active} == true
emit:
- signal: \texttt{lights.external.backup}
  value: true

\begin{itemize}
\item \texttt{transmission.gear} = 'reverse'
\item \texttt{backup.camera.active} = true
\item \texttt{lights.external.backup} = true
\end{itemize}
Or in parallel

parallel:
- condition: transmission.gear == 'reverse'
  - emit:
    signal: lights.external.backup
    value: true
- condition: radar.forward.distance < 120
  - emit:
    signal: audio.distance_warn
    value: true

<< transmission.gear = 'reverse'
>> lights.external.backup = true
<< radar.forward.distance = 113
>> audio.distance_warn = true
Add timing to get a test specification

sequence:
- condition: transmission.gear == 'reverse'
- condition: camera.backup.active == true

start: 100
stop: 10000
emit:
- signal: lights.external.backup
  value: true

<< transmission.gear = 'reverse'
[115 msec pause]
<< backup.camera.active = true
[112 msec pause]
>> lights.external.backup = true
Conditions can be nested...

# Wait for gear to go into reverse
- condition: transmission.gear == 'reverse'
  - emit: # Turn on backup lights
    - signal: lights.external.backup
      value: true

# After lights turned on, wait for backup camera
- condition: camera.backup.active == true
  - emit: # Activate backup camera app
    - signal: ivi.activate_app
      value: 'backup_camera'

# Monitored in parallel with transmission.gear
- condition: ...

<< transmission.gear = 'reverse'
>> lights.external.backup = true
<< camera.backup.active = true
>> ivi.active_app = 'backup_camera'
... and cancelled if parent condition turns false

# Wait for gear to go into reverse
- condition: transmission.gear == 'reverse'
  - emit: # Turn on backup lights
    - signal: lights.external.backup
      value: true

# After lights turned on, wait for backup camera
- condition: camera.backup.active == true
  - emit: # Activate backup camera app
    - signal: ivi.activate_app
      value: 'backup_camera_app'

# Monitored in parallel with transmission.gear
- condition: ...

<< transmission.gear = 'reverse'
>> lights.external.backup = true
<< transmission.gear = 'neutral'
Conditions can be arbitrary expressions...

# Wait for gear to go into reverse and backup camera to be active before turning on backup lights and activate infotainment.
- condition: `transmission.gear == 'reverse' && camera.backup.active == true`
- emit: # Turn on backup camera and backup lights
  - signal: `ivi.activate_app`
    value: 'backup_camera'
  - signal: `lights.external.backup`
    value: true
... that can include signal values

# $-denoted signals are substituted for their values
- condition: transmission.is_reversing == true &&
camera.backup.active == $transmission.is_reversing
- emit: # Turn on backup camera and backup lights
  - signal: ivi.activate_app
    value: 'backup_camera'
  - signal: lights.external.backup
    value: $transmission.is_reversing

<< transmission.is_reversing = true
<< camera.backup.active = true
>> ivi.active_app = 'backup_camera'
>> lights.external.backup = true
You can run multiple signal managers in parallel
Future: API calls?

```python
- condition: transmission.gear == 'reverse'
call: Lights.Control('backup', 'on')

transmission.gear = 'reverse'
```

```python
>> Lights.Control('backup', 'on')
```

Signal Manager

DriveTrain ECU

Lights ECU
Conclusion

• **Separate out call flow logic to manage variance complexity**
  Separate rule YAML files for different models, markets, and configuration.

• **Rules are testable in the rig and in the field**
  Timing information can be left in production to detect issues in deployed fleet.

• **Simplified OTA**
  Rules can be pushed over the air without the full validation and installation process required by a software update.
Thank you!

Rudi Streif, Jaguar Land Rover - rstreif@jaguarlandrover.com
Magnus Feuer, Jaguar Land Rover - mfeuer1@jaguarlandrover.com

Project: https://github.com/genivi/vehicle_signal_manager
Visit GENIVI at http://www.genivi.org or http://projects.genivi.org
Contact us: help@genivi.org

This work is licensed under a Creative Commons Attribution-Share Alike 4.0 (CC BY-SA 4.0)
GENIVI is a registered trademark of the GENIVI Alliance in the USA and other countries.
Copyright © GENIVI Alliance 2017.