Vehicle Data Interfaces

October 20, 2016 | Enabling the Connected Car

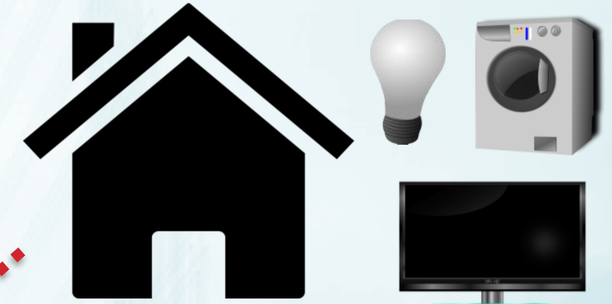
Rudolf J Streif

Networking Expert Group Lead, GENIVI Alliance

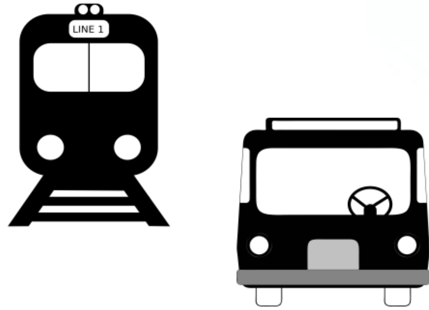
This work is licensed under a Creative Commons Attribution-Share Alike 4.0 (CC BY-SA 4.0)
GENIVI is a registered trademark of the GENIVI Alliance in the USA and other countries.
Copyright © GENIVI Alliance 2016.

The Problem

Smart City



Smart Home



Intermodal Transportation

Connected Devices

V2I

V2V

GENIVI®

The Challenge

- Providing access to vehicle status information and data to cloud services, web applications, mobile devices and more.
- There is no standard convention for a vehicle data API.
- OEMs wish to be able to easily extend a standard API with signals and controls for their purposes.
- Security mechanisms are required that provide authentication and authorization to access vehicle signals and control.
- Design that decouples signal interface from the electrical architecture of the vehicle.

GENIVI®

Conventional Approach – “Fat API”

- An API for every signal or control:

```
var vehicle = navigator.vehicle;
vehicle.vehicleSpeed.get().then(function (vehicleSpeed) {
    console.log("Vehicle speed: " + vehicleSpeed.speed);
}, function (error) {
    console.log("There was an error"); });
var vehicleSpeedSub = vehicle.vehicleSpeed.subscribe(function (vehicleSpeed) {
    console.log("Vehicle speed changed to: " + vehicleSpeed.speed);
    vehicle.vehicleSpeed.unsubscribe(vehicleSpeedSub);
});
```

- Issues with this approach:
  - Addition of new signals and controls requires change of the specification.
  - Challenges maintaining backwards compatibility.
  - Complexity in providing per-API authorization and access control.
  - Single end-point addressing.

GENIVI®

New Approach – Services with Signal Tree

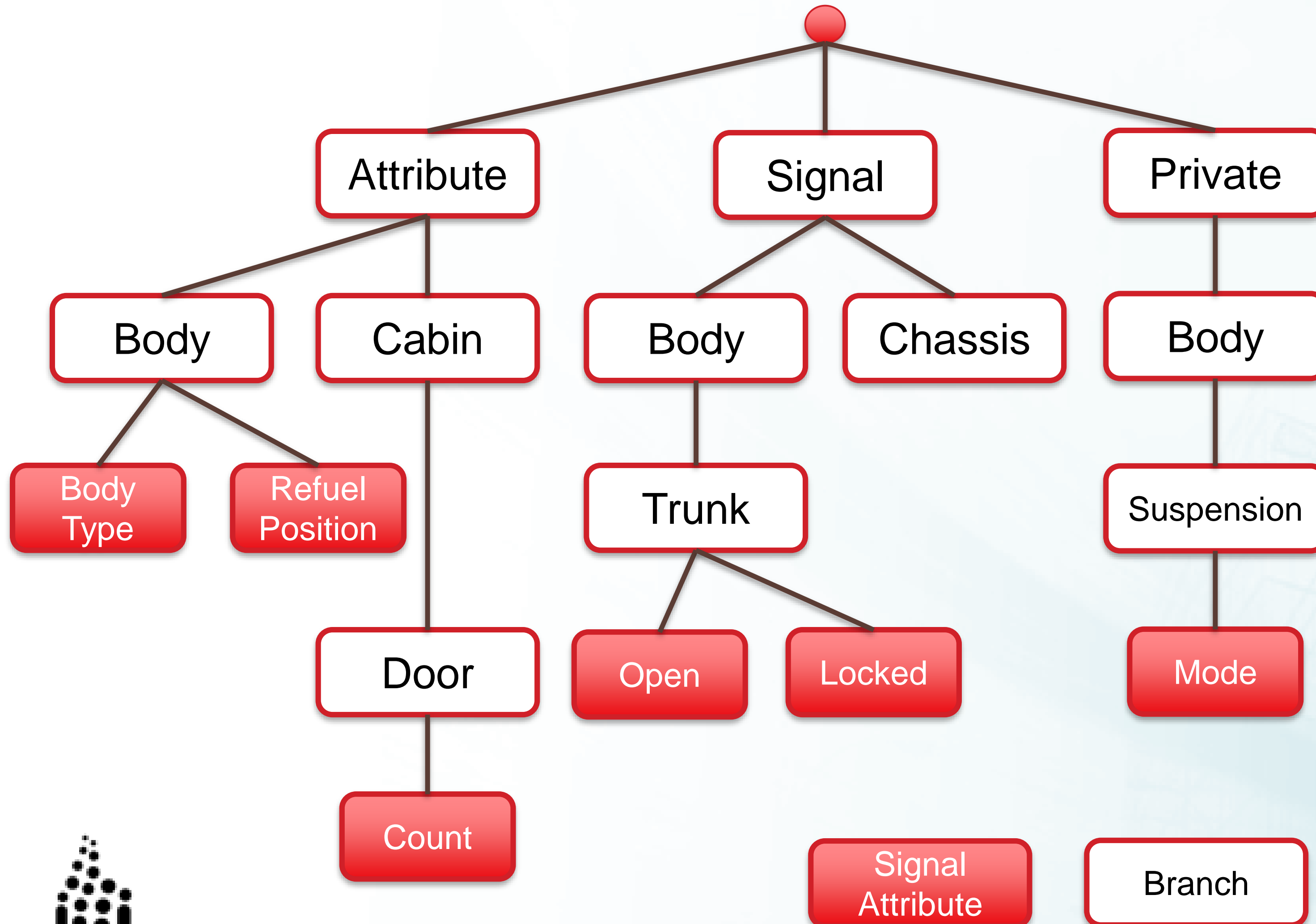
- The core services *get*, *set*, *subscribe*, *unsubscribe*, *getVSS* and *authorize* are provided by a network server.
  - The services *get*, *set*, *subscribe* and *unsubscribe* provide access to vehicle signals and controls.
  - The service *getVSS* allows clients to query the server for available signals.
  - Using the *authorize* service, the client presents a security token to the server for authentication and authorization.
- Vehicle Signals and Controls are identified as nodes of a vehicle signal tree.
  - A fully qualified signal name addresses a single signal node.
  - Wildcards for branches and node names provide for addressing of signal groups.

GENIVI®

Vehicle Signal Tree

Vehicle Signal Specification

Vehicle Signal Tree



- Tree structure provides for hierarchical access to signals and attributes.
- Branches group signals and attributes into entities that logically belong together.
- Wildcards allow access to entire sets of signals.

GENIVI®

Copyright © GENIVI Alliance 2016

Addressing

Signal.Chassis.Brake.FluidLevel
Signal.Drivetrain.FuelSystem.Level
Attribute.Cabin.Door.Count
Attribute.Engine.Displacement

- Dot-notation for name path.
- Last path component, called node, represents the signal or attribute.
- Leading path components represent the branches.
- Wildcards can be used to address multiple signals and/or branches.

GENIVI®

Specification Format

```
- Signal.Drivetrain.Transmission:
  type: branch
  description: Transmission-specific data
- Signal.Drivetrain.Transmission.Speed:
  type: Int32
  min: -250
  max: 250
  unit: m/s
  description: Current vehicle speed, sensed by gearbox
```

- Formatted as YAML lists
- Simple conversion into other formats such as JSON, France IDL, CSV, and more
- # denotes a comment or a directive

GENIVI®

Specification Format – Branch Description

```
- Signal.Drivetrain.Transmission:
  type: branch
  description: Transmission-specific data
```

- Fields
  - `type` – always set to branch for a branch
  - `description` – informative text describing the branch

GENIVI®

Specification Format – Signal Description

```
- Signal.Drivetrain.Transmission.Speed:
  type: Int32
  min: -250
  max: 250
  unit: m/s
  description: Current vehicle speed, sensed by gearbox
```

- Fields
  - `type` – data type expressed as France IDL data type
  - `unit` – SI unit unless the type is Boolean
  - `min`, `max` – unless the type is Boolean or enumeration
  - `enum` – enumeration values for enumeration
  - `description` – informative text describing the signal

GENIVI®

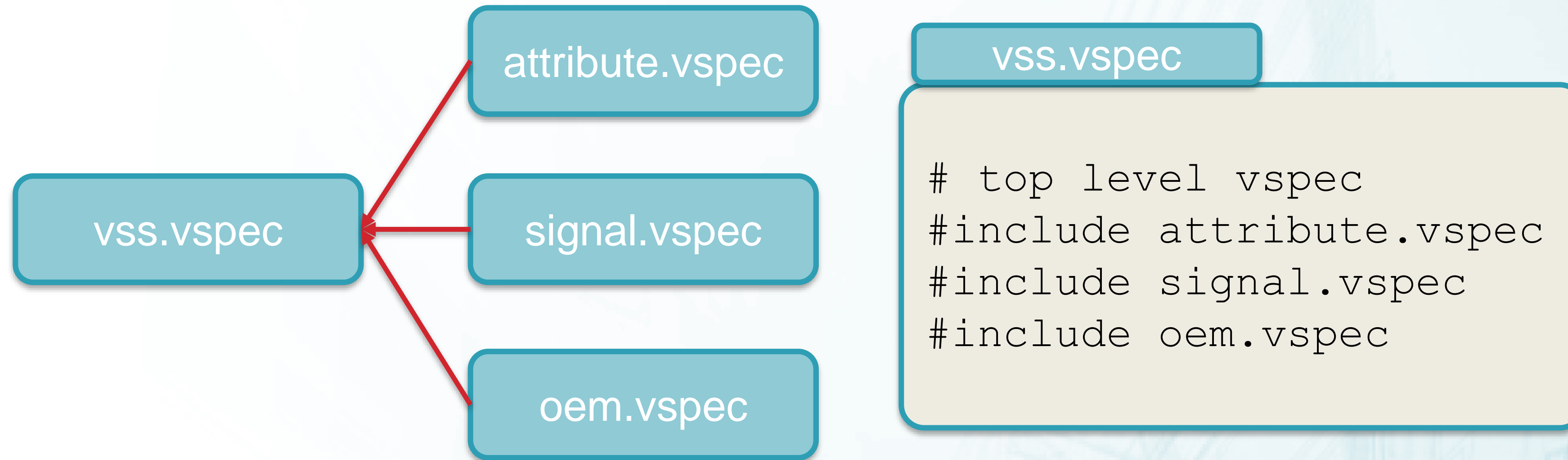
Specification Format – Attribute Description

```
- Attribute.Cabin.Door.Count:
  type: Uint8
  value: 4
  description: Current vehicle speed, sensed by gearbox
```

- Fields
  - Same as signal
  - `value` – attribute setting
- Attributes are used to describe configuration data.

GENIVI®

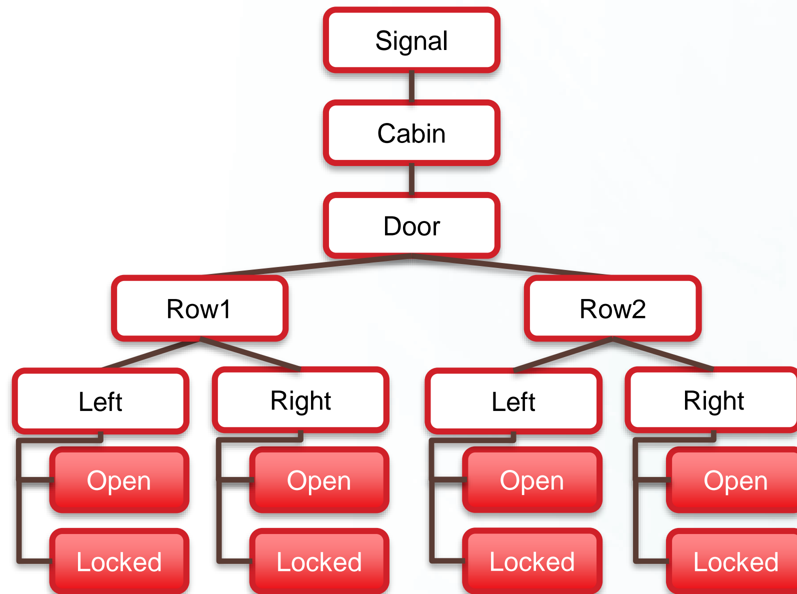
Aggregate File Inclusion



- Vehicle signal specification files (vspec) can include other vspec file using the `#include` directive.
- Content of the included file is inserted into the including file at the position of the `#include` directive.
- Facilitates collaboration and minimizes editorial conflicts.

GENIVI®

Reuse File Inclusion



door.vspec

```
# door signals
-  Open:
   type: Boolean
   description: Door is open
-  Locked:
   type: Boolean
   description: Door is locked
```

cabin.vspec

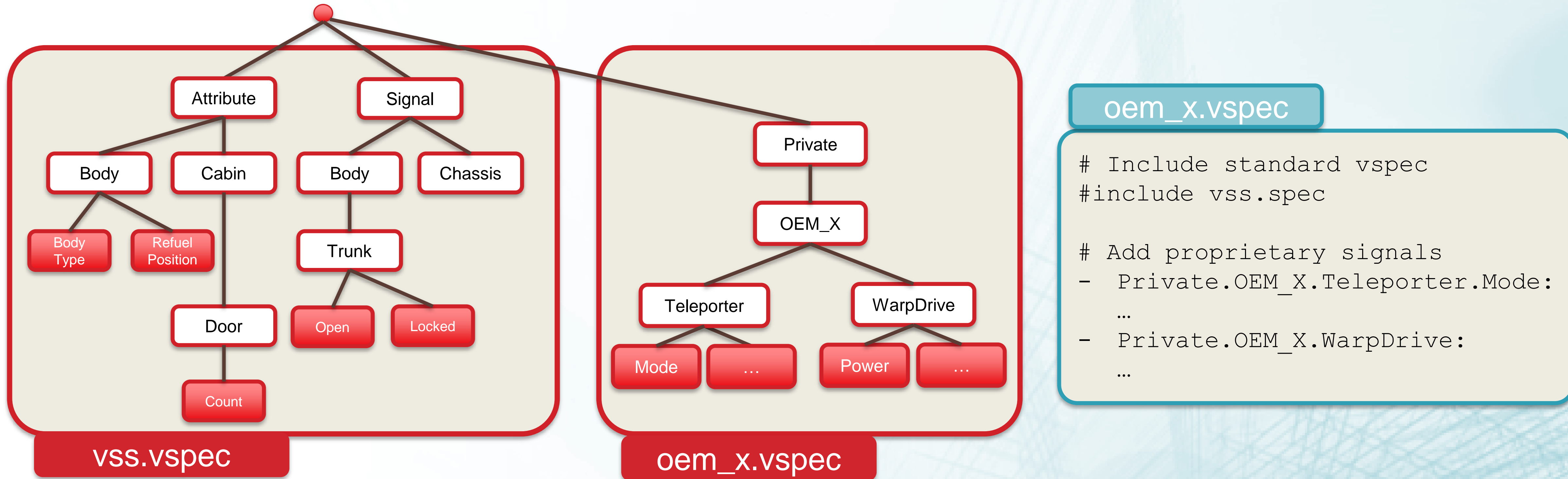
```
# doors
#include door.vspec Signal.Cabin.Door.Row1.Left
#include door.vspec Signal.Cabin.Door.Row1.Right
#include door.vspec Signal.Cabin.Door.Row2.Left
#include door.vspec Signal.Cabin.Door.Row2.Right
```

- Specification fragments are included at a specific position of the signal tree.
- Specification fragments can be reused and an update is automatically reflected everywhere where the fragment is used.

GENIVI®

Copyright © GENIVI Alliance 2016

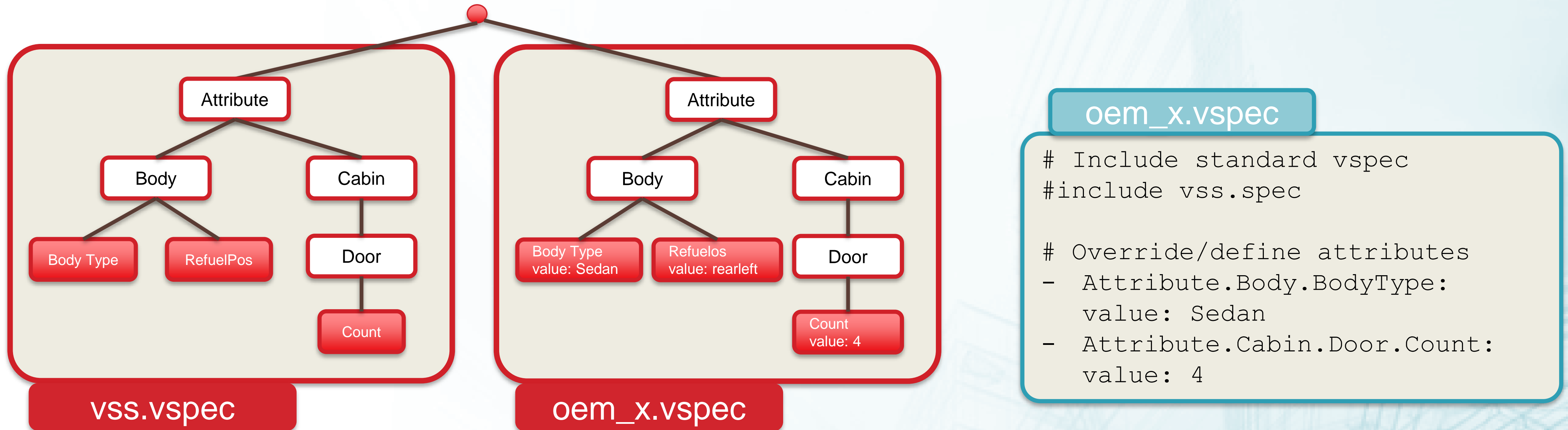
Private OEM Extensions



- OEMs can use GENIVI vspec as a starting point and add proprietary signals.
- Use cases for
  - Reserved use by OEM and chosen vendors;
  - Public use by 3rd party application developers.
- Mature private extensions intended for public use can be submitted for VSS inclusion.

GENIVI®

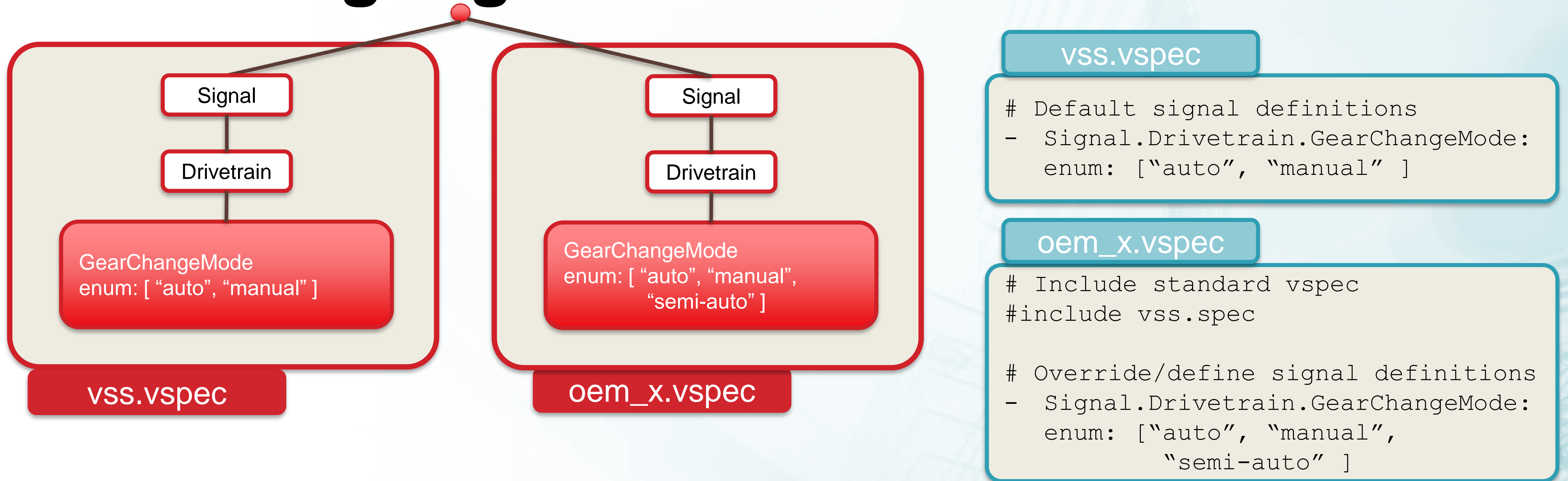
Attribute Declaration and Definition



- Standard VSS either
  - Only declares an attribute or
  - Declares and attribute and assigns a default value.
- Declaration is overridden by definition in an OEM- or model-specific VSS file with the correct value.

GENIVI®

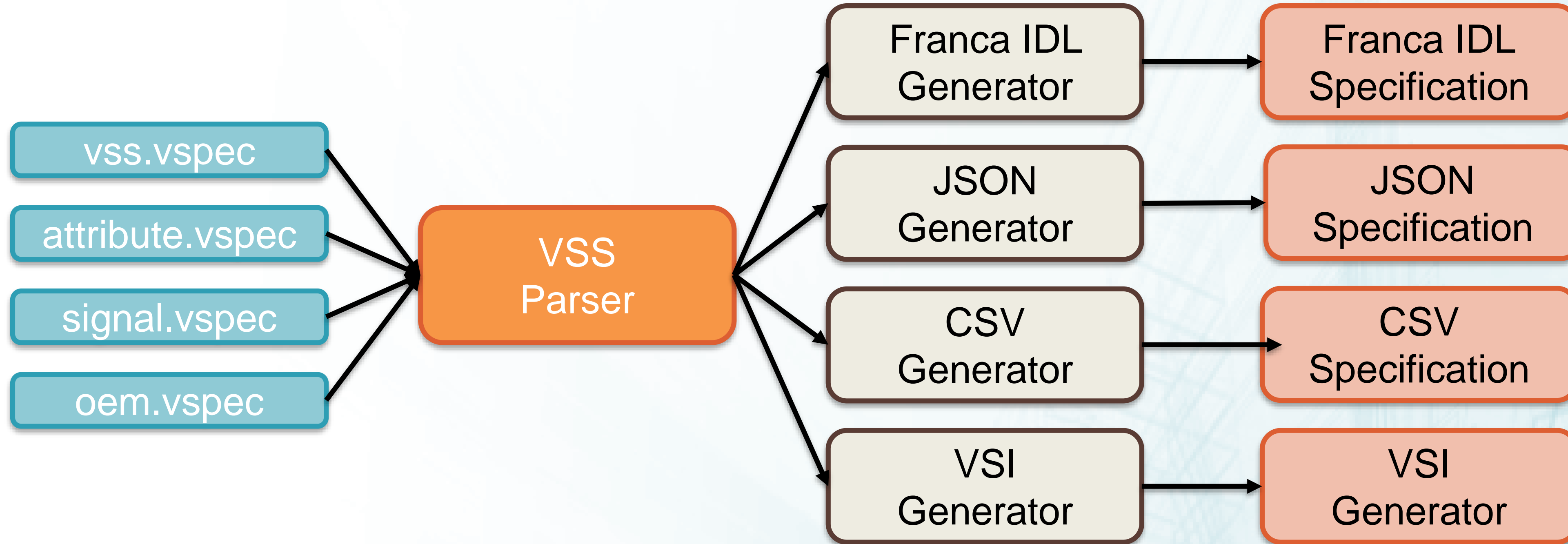
Overriding Signal Definitions



- Standard vspec lacks setting or has incorrect setting for a OEM/model etc.
- OEM/model-specific vspec can override the setting.

GENIVI®

Format Transformation

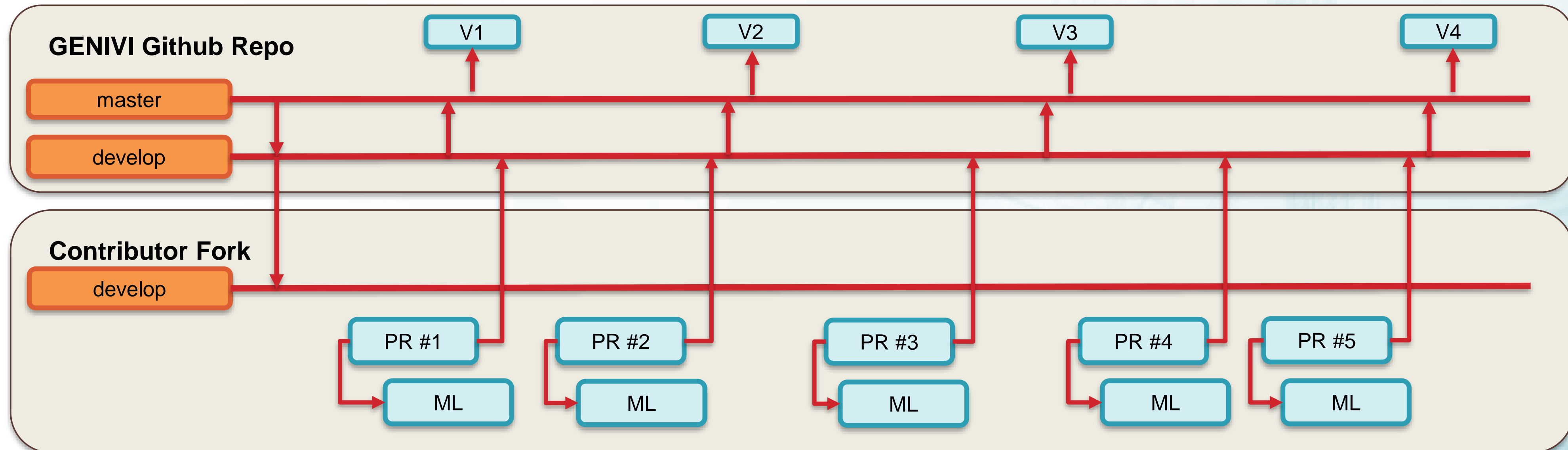


- Tools written in Python transform VSS YAML (vspec) format into other formats.
- Standard Python library parses VSS YAML into a data structure.
- Output generators use the data structure to write their specific format.
- Output generators for Franca IDL, JSON, CSV and VSI are currently available. Other generators can easily be added.
- The VSI generator creates an alphabetically sorted list of the fully qualified signal and attribute names and assigns an index value to them.

GENIVI®

Contribution and Releases

- Repository on Github under the GENIVI organization:
  https://github.com/GENIVI/vehicle_signal_specification

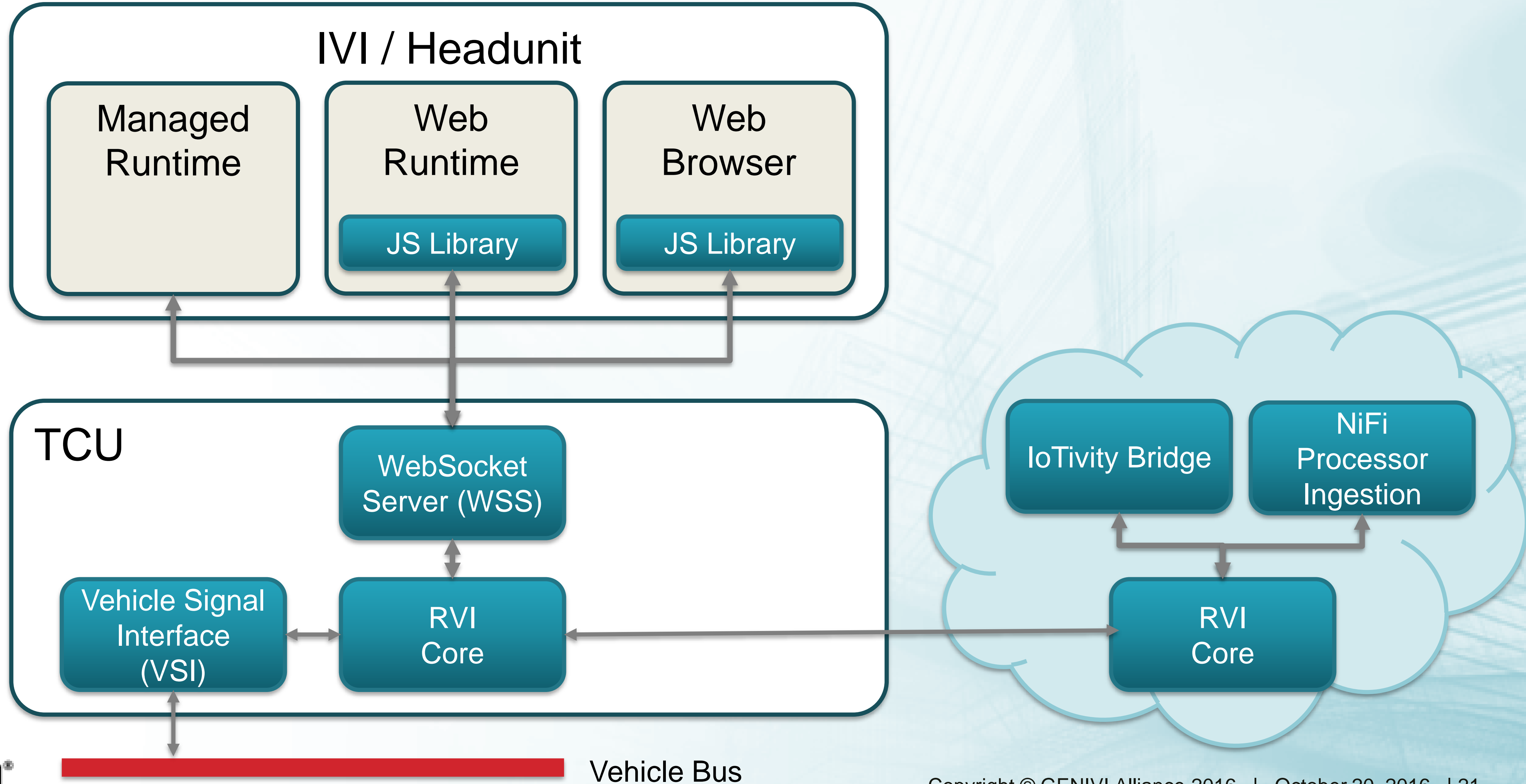


- Contributor forks GENIVI VSS repo.
- Contributor makes changes and submits pull-request against develop branch.
- Contributor e-mail genivi-projects mailing list pull-request info (hypertext link).
- Maintainer and contributors discuss and approve. Maintainer merges pull request.
- Releases are created by merging the develop branch into the master branch and tagging the master branch.

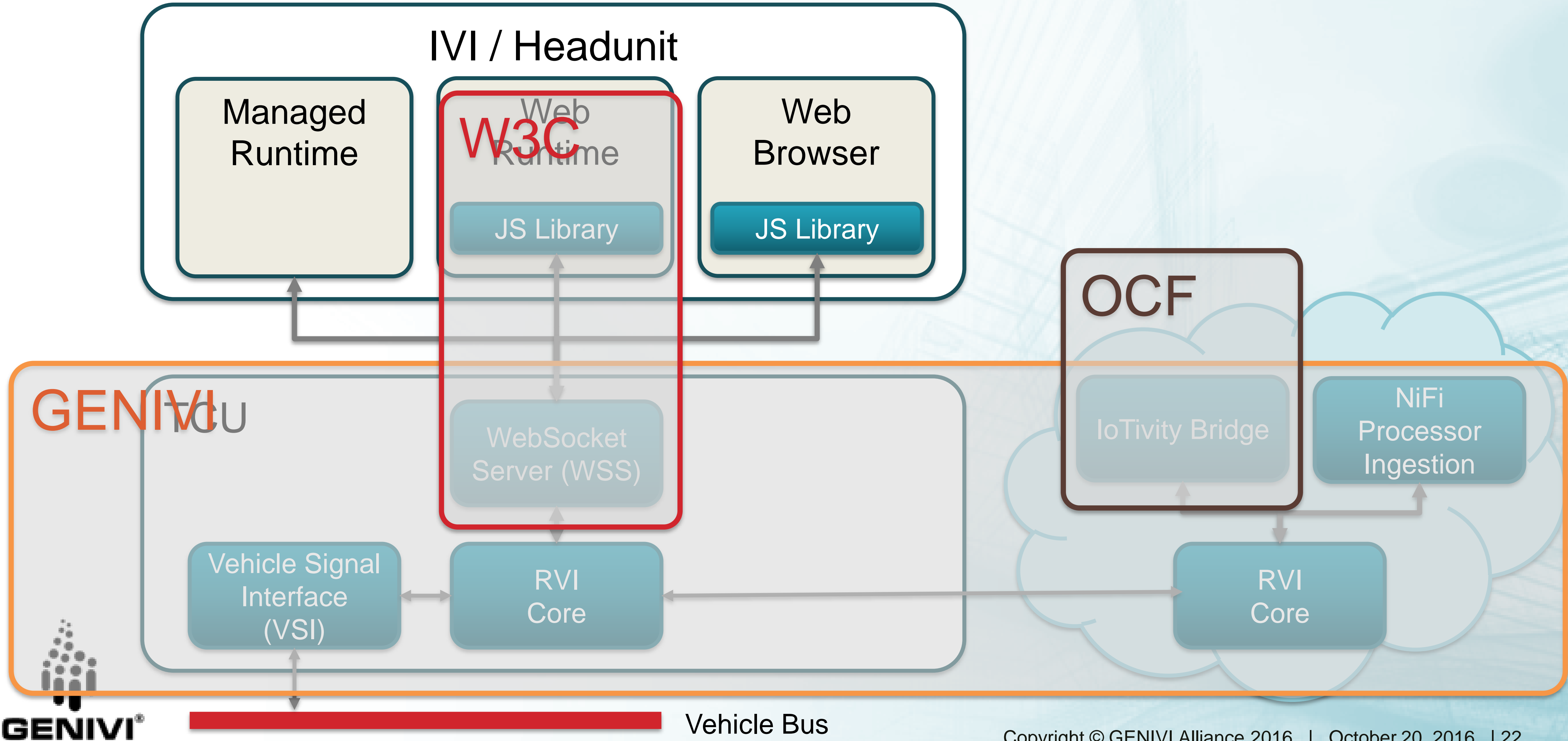
Architecture

GENIVI®

Vehicle Data Interfaces Architecture



Vehicle Data Interfaces Architecture



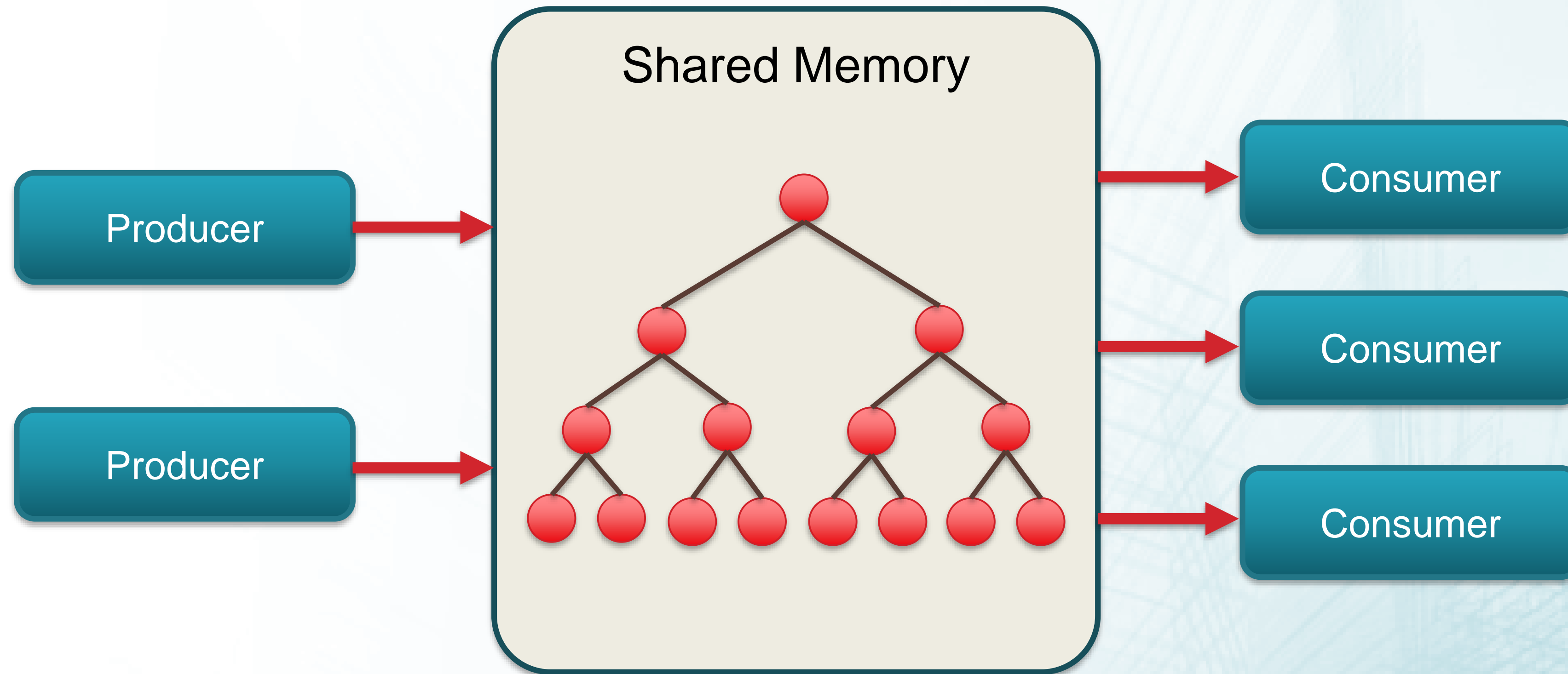
Vehicle Signal Interface (VSI)

Vehicle Signal Interface (VSI) - Overview

- High-speed switchboard:
  - Up to 10 million transactions per second
  - Implemented in C
- Core library with API to implement VSI sources and sinks:
  - Interfaces to vehicle buses such as CAN.
  - Interfaces to RVI and/or other applications.
- Signals are identified by either name or ID. Two sets of APIs e.g.:
  - `int vsi_fire_signal ( vsi_handle handle, vsi_result* result );`
  - `int vsi_fire_signal_by_name ( vsi_handle handle, vsi_result* result );`
- Lookup functions to convert signal names to ID and vice versa:
  - Signal map can be imported from VSI file created by the `vss2vsi` transformation tool.
- Signals can be grouped and an application can listen to individual signals in the group or all signals.
- Signal switchboard is implemented as B-tree database in shared memory.

GENIVI®

Vehicle Signal Interface (VSI) - Design



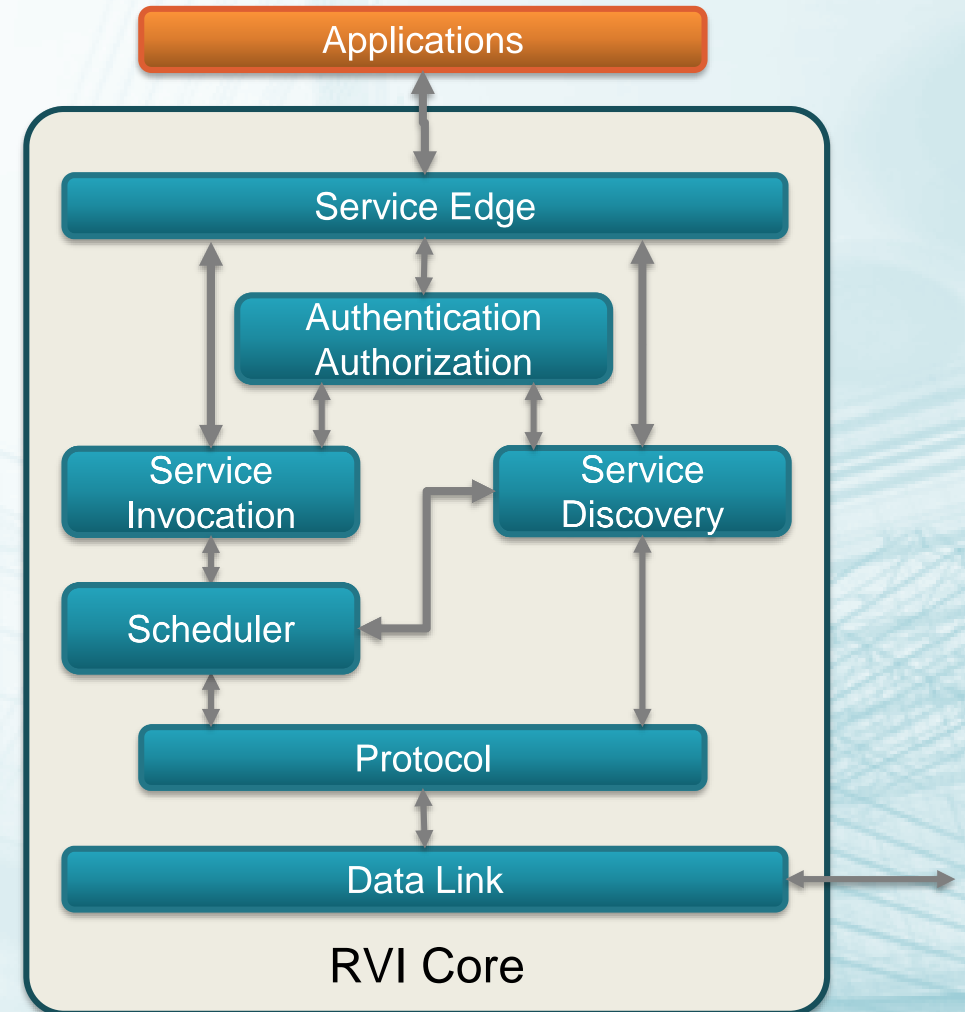
- Producers post signals into VSI shared memory where they are stored in a b-tree ordered by signal ID.
- Consumers read individual signals or signal groups from shared memory. Read functions return immediately if a signal has been posted or block until a signal arrives.
- Callback functions are not supported.

GENIVI®

Remote Vehicle Interaction (RVI)

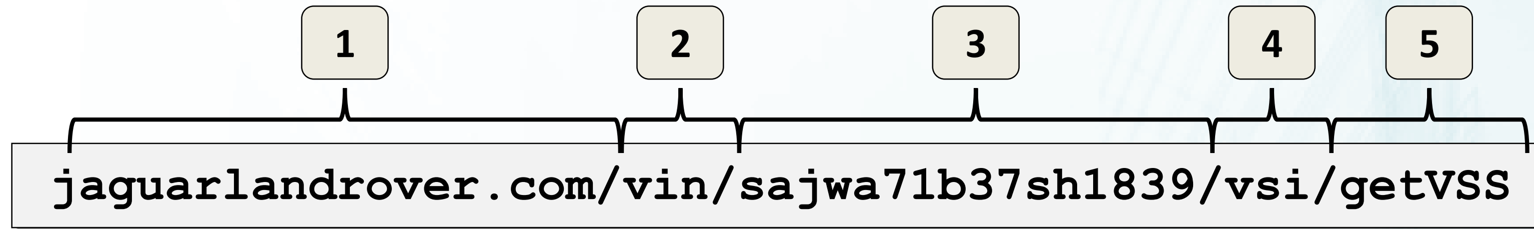
RVI Core

- **Service Edge**
  - Manages traffic from and to application.
- **Authentication / Authorization**
  - Manages certificates that allow applications to discover and invoke services.
- **Service Discovery**
  - Identifies and locates local and remote services.
- **Service Invocation**
  - Receives and dispatches local and remote service calls.
- **Scheduler**
  - Stores and forwards messages for unavailable destinations.
- **Protocol**
  - Encodes and decodes messages.
- **Data Link**
  - Controls data transmission to other RVI nodes.



GENIVI

RVI Service Addressing



#	Name	Description
1	Organization	Specifies a sub-section hosted by a specific entity
2	VIN sub-tree	Specifies sub section for all vehicles
3	VIN	Vehicle Identification Number
4	Service Domain	Domain of service
5	Service Command	Service command within the service domain

GENIVI®

RVI Security

- TLS-protected Internode Communication
  - Prevent replay attacks.
  - Prevent man-in-the-middle attacks.
- Certificate-based Node Authentication and Service Authorization
  - Certificates, signed by a trusted provisioning system, attest application identity and grant access to services.
- Self-carried application authentication and service authorization
  - A Node presents its certificates to another node to authenticate itself and provide its service authorization. No connection to a server is required.
  - Each certificate carries the node’s public key. Nodes sign all messages with their private key.

GENIVI®

Copyright © GENIVI Alliance 2016

Web Vehicle Signal Service (WVVS)

Copyright © GENIVI Alliance 2016

Overview

- A web socket server (NodeJS etc.) provides access to vehicle signals.
- Web clients such as applications running inside a web runtime or a web browser communicate with the web socket server using a JavaScript library which implements the web socket server protocol and exposes an object API.
- Native clients can directly use the web socket server protocol.
- Clients can be agents with no UI or applications with UI.

Internet
Web Server
IVI / Headunit
Internal
Web Server
Managed
Runtime
Web
Runtime
JS Library
Web
Browser
JS Library
TCU
WebSocket
Server (WSS)
Vehicle

Copyright © GENIVI Alliance 2016

Service Messages

Service Messages	
authorize	Enables client to pass security tokens for Security Principals to the server to support access-control.
getVSS	Allows the client to request metadata describing signals and data attributes that are potentially accessible.
get	Enables the client to get a value once.
set	Enables the client to set a value once.
subscribe	Enables the client to receive a notification containing a JSON data structure with values for one or more vehicle signals and/or data attributes. The client requests that it is notified when the signal changes on the server.
unsubscribe	Allows the client to notify the server that it should no longer receive notifications based on that subscription.
unsubsribeAll	Allows the client to notify the server that it should no longer receive notifications for any active subscription.

GENIVI W3C

Authorization – Security Principals

Security Principal	Token Name	Description
User	Authorization	The user that the client is making requests on behalf of. This may be a person e.g. driver or passenger, it may be an organisation e.g. Emergency Services or may be any other legal entity.
Device	www-vehicle-device	The originating device that is making the request to the server. This may be an ECU in the vehicle that is hosting the WebSocket Server or may be a device that is connected to the vehicle via a WiFi hotspot or may be any other device.

Authorization – Example

```
if(userTokenOnly){
  // Pass user token only
  vehicle.send('{ "action": "authorize",
    "tokens": { "authorization": "<user_token_value>" },
    "requestId": "<some_unique_value>" }');

} else if (deviceTokenOnly) {
  // Pass vehicle/device token only
  vehicle.send('{ "action": "authorize",
    "tokens": { "www-vehicle-device": "<device_token_value>" },
    "requestId": "<some_unique_value>" }');

} else if (userAndDeviceToken) {
  // Pass tokens for user and device
  vehicle.send('{ "action": "authorize",
    "tokens": { "authorization": "<user_token_value>",
      "www-vehicle-device": "<device_token_value>" },
    "requestId": "<some_unique_value>" }');
}
```

GENIVI® W3C®

Authorization – Security Token

Element	Description
Path	The signal path the token authorizes. The path may be a branch name or contain wildcards to authorize entire branches.
Actions	List of actions that the token authorizes for the path. The list contains at least one of the actions getVSS, get, set, subscribe and unsubscribe.
Valid From	Timestamp in UTC indicating the date and time from which on the token is valid.
Valid Until	Tmestamp in UTC indicating the date and time until which the token is valid.

GENIVI® W3C®

Introspection – getVSS

WebIDL

```
interface vssRequest {
    attribute Action  action;
    attribute string? path;
};

interface vssSuccessResponse {
    attribute Action action;
    attribute string path;
    attribute object vss;
};

interface vssErrorResponse {
    attribute Action action;
    attribute string path;
    attribute Error  error;
};
```

Message

```
client -> {
   "action": "getVSS",
   "path": "Signal.Body"
 }
 receive <- {
   "action": "getVSS",
   "path": "Signal.Body",
   "vss": { }
 }
```

GENIVI W3C

Get Signal Value – get

WebIDL

```
interface getRequest {
     attribute Action  action;
     attribute DOMString path;
};

interface getSuccessResponse {
     attribute Action action;
     attribute DOMString path;
     attribute any value;
     attribute DOMTimeStamp timestamp;
};

interface getErrorResponse {
     attribute Action action;
     attribute DOMString path;
     attribute Error  error;
 attribute DOMTimeStamp timestamp;
};
```

Message

```
client -> {
   "action": "get",
   "path": "Signal.Drivetrain.Speed",
 }
 receive <- {
   "action": "get",
   "path": "Signal.Drivetrain.Speed",
   "value": 55,
   "timestamp": <DOMTimeStamp>
 }
```

GENIVI W3C

Set Signal Value – set

WebIDL

```
interface getRequest {
    attribute Action  action;
    attribute DOMString path;
    attribute any value;
};

interface setSuccessResponse {
    attribute Action action;
    attribute DOMString path;
    attribute any value;
    attribute DOMTimeStamp timestamp;
};

interface setErrorResponse {
    attribute Action action;
    attribute DOMString path;
    attribute Error  error;
    attribute DOMTimeStamp timestamp;
};
```

Message

```
client -> {
    "action": "set",
    "path": "Signal.Cabin.Door.*.IsLocked",
    "value":{ [ {"Row1.Right.IsLocked" : true },
                {"Row1.Left.IsLocked" : true },
                {"Row2.Right.IsLocked" : true },
                {"Row2.Left.IsLocked" : true } ] }
    }

  receive <- {
    "action": "set",
    "path": "Signal.Cabin.Door.*.IsLocked",
    "value":{ [
{"Signal.Cabin.Door.Row1.Right.IsLocked" : true },
{"Signal.Cabin.Door.Row1.Left.IsLocked" : true },
{"Signal.Cabin.Door.Row2.Right.IsLocked" : true },
{"Signal.Cabin.Door.Row2.Left.IsLocked" : true } ]
},
    "timestamp": <DOMTimeStamp>
  }
```

Subscription Request – subscribe

WebIDL

```
interface subscribeRequest {
    attribute Action  action;
    attribute DOMString path;
    attribute object? filters;
    attribute string requested;
};

interface subscribeSuccessResponse {
    attribute Action action;
    attribute string requestId;
    attribute string subscriptionId;
    attribute DOMTimeStamp timestamp;
};

interface subscribeErrorResponse {
    attribute DOMString path;
    attribute string requestId;
    attribute Error  error;
    attribute DOMTimeStamp timestamp;
};
```

```
interface subscriptionNotification {
    attribute string  subscriptionId;
    attribute DOMString path;
    attribute any value;
    attribute DOMTimeStamp timestamp;
};

interface subscriptionNotificationError {
    attribute string  subscriptionId;
    attribute DOMString path;
    attribute object filters;
    attribute Error error;
    attribute DOMTimeStamp timestamp;
};
```

GENIVI® W3C

Copyright © GENIVI Alliance 2016

Subscription Request – subscribe

Message

```
client -> {
    "action": "subscribe",
    "path": "Signal.Drivetrain.Transmission.TripMeter",
    "requestId": 1004 }
}

  receive <- {
    "action": "subscribe",
    "reqestId": 1004,
    "subscriptionId": 35472,
    "timestamp": <DOMTimeStamp>
  }
```

GENIVI W3C®

Unsubscription Request – unsubscribe

WebIDL

```
interface unsubscribeRequest {
    attribute Action  action;
    attribute string subscriptionId;
    attribute string requestId;
};

interface unsubscribeSuccessResponse {
    attribute Action action;
    attribute string? subscriptionId;
    attribute string requestId;
    attribute DOMTimeStamp timestamp;
};

interface unsubscribeErrorResponse {
    attribute Action action;
    attribute string subscriptionId;
    attribute Error  error;
    attribute string requestId;
    attribute DOMTimeStamp timestamp;
};
```

Message

```
client -> {
    "action": "unsubscribe",
    "subscriptionId": 102,
    "requestId": 5273
    }

  receive <- {
    "action": "unsubscribe",
    "subscriptionId": 102,
    "requestId": 5273
    "timestamp": <DOMTimeStamp>
  }
```

GENIVI W3C

Serverside Filtering

- For signal subscriptions filters can be provided to throttle messages on the server side.
- Filters only apply to nodes of the VSS tree and not to branches.
- Filter tags include:
  - Interval
  - Range
  - Minimum Change

Message

```
// client receives data every 100ms
 { "action": "subscribe",
   "path": "<any_path>",
   "filters": { "interval": 100 },
   "requestId": "<some_unique_value>" }

 // client receives data when the value is
 // between 100 and 200 (inclusive)
 { "action": "subscribe",
   "path": "<any_path>",
   "filters": {
       "range":{ "above": 100, "below": 200 }
   },
   "requestId": "<some_unique_value>" }

 // client receives data when the value is below
 // 100 (inclusive)
 { "action": "subscribe",
   "path": "<any_path>",
   "filters": { "range": { "below": 100 } },
   "requestId": "<some_unique_value>" }
```

Repository and Specification

https://github.com/w3c/automotive

https://w3c.github.io/w3c/automotive

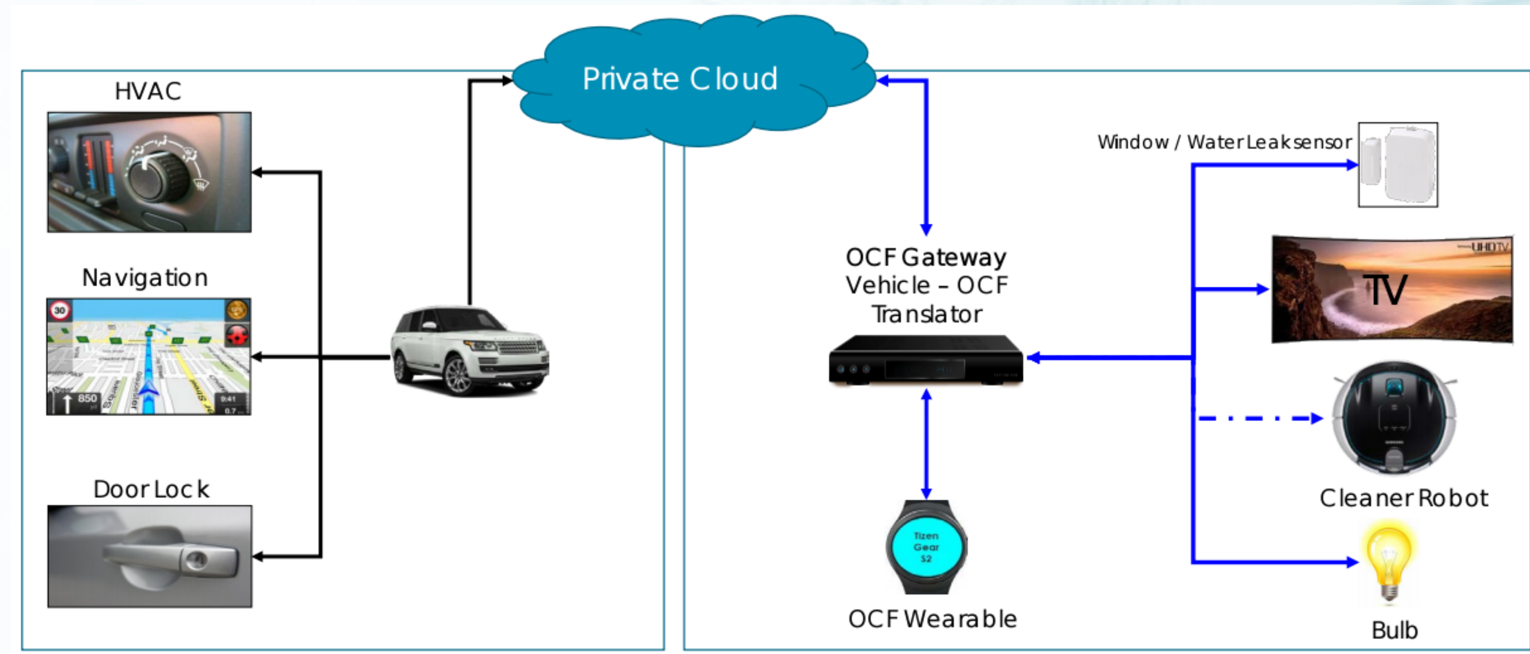
GENIVI W3C®

Open Connectivity Foundation (OCF)

OPEN CONNECTIVITY FOUNDATION™

GENIVI and OCF

- IoTivity RVI Demonstrator, GENIVI AMM, Paris April 2016
- Automotive in OCF, OCF AMM, Portland June 2016
- OCF Announces Automotive Project, August 2016



FOUNDATION™

Questions?

Thank you!

Weekly Networking Expert Group Call

Mondays 0800 PT / 1700 CET

https://genivi.webex.com/genivi/j.php?MTID=mdb9482b92015e5cb7386c1a65e32a887

Meeting number: 579 975 193

Mailing List

https://mail.genivi.org/sympa/info/eg-nw