Web API for Vehicle Data RI

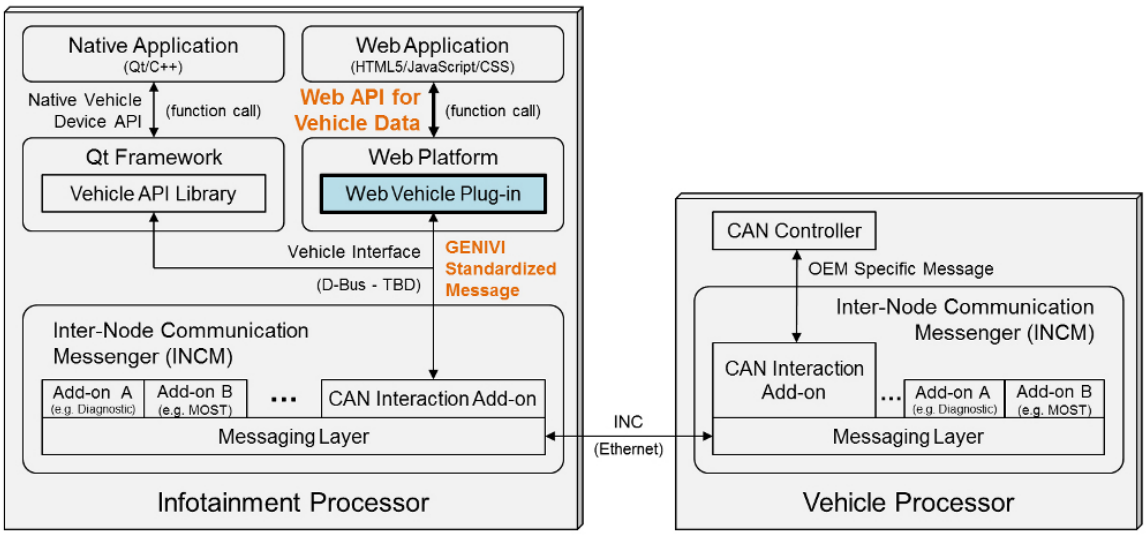
Reference implementation of Web API for Vehicle Data

1. Objective

Provide sample implementation of Web API for Vehicle Data to demonstrate its feasibility.
Provide sample code that shows how to implement Web API for Vehicle Data for a platform provider and how to use it for a web application developer.

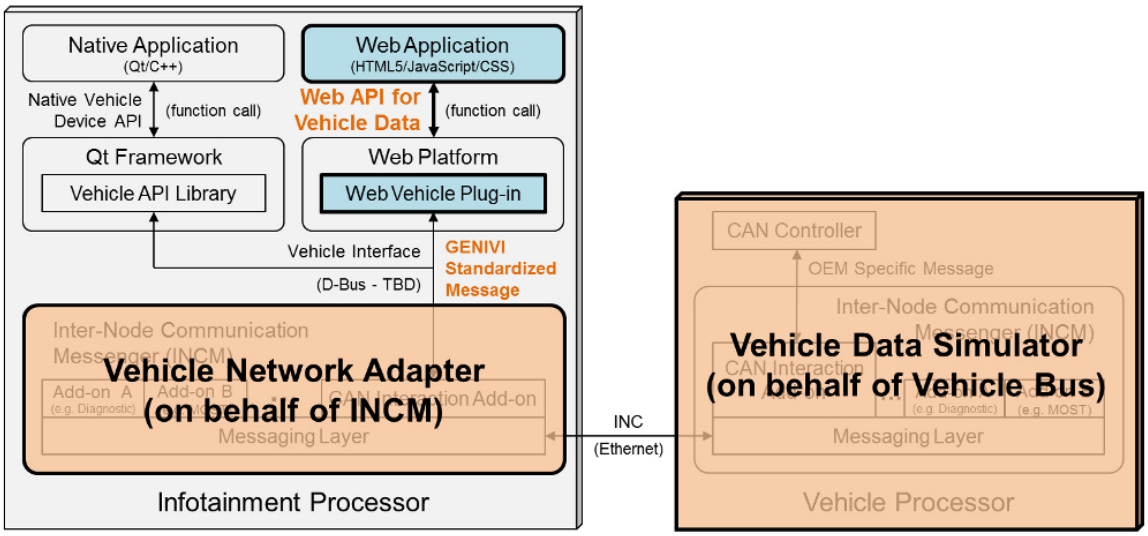
2. Architecture Overview

2.1. The relation of Web API with the other GENIVI components



The architecture above are represented several times in various GENIVI meetings.
It shows that there are dependencies with INCM, Vehicle Interface, and Web Platform (as a different type of Browser).
Although Web APIs for Vehicle Data can be defined regardless of these projects, all things are required to show it working in implementation.
Since it's not available now, an alternative way was needed.

2.2. The organization of this reference implementation



2.2.1 Vehicle Data Simulator

Vehicle Data Simulator needs to be implemented on behalf of real vehicle bus and bus messages.
It supports both sending a bus message as needed and showing received messages.

2.2.2 Vehicle Network Adapter

Vehicle Network Adapter plays a similar role of INCM.
As a counter part of Vehicle Data Simulator, it delivers data via D-Bus for the other GENIVI components.

2.2.3 Vehicle Interface

When Vehicle Network Adapter delivers data via D-Bus, D-Bus interfaces shall be defined. It's a scope of Vehicle Interface Project.
Since it has not been defined yet, we need to define it our own way.
As Vehicle Interface project progresses, these part has to be updated.

2.2.4 Web Platform

Web API can be implemented as a various ways.
This reference implementation is made as web plug-in using FireBreath framework.
FireBreath helps to greatly reduce the effort for implementing web plug-ins. It is also compatible with various web browsers.
This reference implementation works on a browser, so a browser which supports NPAPI needs to be installed.
Various browsers such as MeeGo Browser, Chrome, and Firefox are supported.
As Browser project progresses, these part may need to be updated.

2.2.5 Web Application

Simple Web Page using JavaScript is provided to show web application developers how to use Web API for Vehicle Data.

> This reference implementation is not intended to be included GENIVI baseline yet.
> To do that, the other related components (INCM, Vehicle Interface, and Browser) should be also included or completed.

3. User Manual

3.1. Directory Structure

- **./bin** - A directory for binary files (VehicleNetworkAdapter and VehicleDataSimulator) and its network config file (network.cfg)
- **./doc** - A directory for documents which explain Web API for Vehicle Data and its reference implementation
- **./html** - A direcroty for a sample web page for testing Web API for Vehicle Data
- **./script** - A direcroty for build scripts
- **./src** - A directory for source codes

3.2. Build and Install

To build the reference implementation, following packages are required:

- GCC compiler

- D-Bus library
- Qt SDK (Tested on Qt version 4.8.1)
- CMake version 2.8 (for FireBreath)
- libgtk2.0-dev (for FireBreath)
- git (for FireBreath)

To build and install all projects, run:

```
./script/build-all.sh
```

(Optional) If you want to build and install each project separately, run following scripts:

```
./script/build-plugin.sh (for Browser Plugin)
./script/build-vna.sh  (for Vehicle Network Adapter)
./script/build-vds.sh (for Vehicle Data Simulator)
```

To see usage of each build script, run the script with -h or --help option

3.3. Run and Test

To change network settings (D-Bus and socket), edit ./bin/network.cfg

To run the reference implementation, run:

```
./bin/VehicleDataSimulator &
./bin/VehicleNetworkAdapter
```

and open the following web page on your web browser (Google chrome or Firefox)

```
e.g. google-chrome ./html/index.html
```

3.4. How to use Web API for Vehicle Data

3.4.1 Web plug-in registration

Consider that the web vehicle plug-in is already installed to the browser.
In **html**, write the following object tag.

```
...
<body>
 <object id='plugin' type='application/x-webapiforvehicledata' width='0' height='0'></object>
...
```

In **JavaScript**, define a variable (*vehicle*) and get the object into *vehicle*.

```
window.onload = function() {
 vehicle = document.getElementById('plugin').vehicle;
...
}
```

Then, the plug-in can be accessible via *vehicle*.

3.4.2 Getting a single vehicle data

Let's get the tire pressure status for the front left tire and notice the status to the driver.
Call the get function with a callback function (*handleVehicleData*)

```
vehicle.get('maintenance_tirepressurestatus_frontleft', handleVehicleData, handleError);
function handleVehicleData(data) {
 if (data.tirePressureStatusFrontLeft == 0) {
  alert('Tire pressure status (front-left) is normal.');
 } else if (data.tirePressureStatusFrontLeft == 1) {
  alert('Tire pressure status (front-left) is low.');
 } else if (data.tirePressureStatusFrontLeft == 2) {
  alert('Tire pressure status (front-left) is high.');
 }
}
```

If you do not want to handle an error, just write *null*.

```
vehicle.get('maintenance_tirepressurestatus_frontleft', handleVehicleData, null);
```

3.4.3 Getting multiple vehicle data

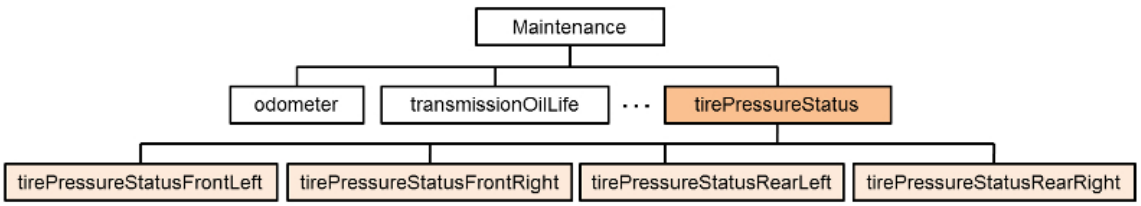
Let's get tire pressure status for all tires simultaneously and notice to the driver if there is a problem at any tire.
In the previous way, you have to get the status of each tire.

```
vehicle.get('maintenance_tirepressurestatus_frontleft', handleVehicleData, handleError);
vehicle.get('maintenance_tirepressurestatus_frontright', handleVehicleData, handleError);
vehicle.get('maintenance_tirepressurestatus_rearleft', handleVehicleData, handleError);
vehicle.get('maintenance_tirepressurestatus_rearright', handleVehicleData, handleError);
function handleVehicleData(data) {
 if ((data.tirePressureStatusFrontLeft != 0) || (data.tirePressureStatusFrontRight != 0) ||
     (data.tirePressureStatusRearLeft != 0) || (data.tirePressureStatusRearRight != 0)) {
  alert('Check tire pressure.');
 }
}
```

However, with the upper level type ('maintenance_tirepressurestatus'), the code becomes quite simple.

```
vehicle.get('maintenance_tirepressurestatus', handleVehicleData, handleError);
```

- Tree representation of MaintenanceEvent



3.4.4 Adding event listener(s)

Let's add an event listener to monitor the tire pressure status for the front left tire.

```
vehicle.addEventListener('maintenance_tirepressurestatus_frontleft', handleVehicleData, false);
```

Also, you can use the upper level type to add multiple listeners.

```
vehicle.addEventListener('maintenance_tirepressurestatus', handleVehicleData, false);
```

A callback function (*handleVehicleData*) is called whenever any of tire pressure status is changed.

3.4.5 Setting a single vehicle data

Assume that driver seat position can be set in this vehicle.
Let's set the driver seat position for recline seatback.
Create an object (*obj*) and add an attribute in the obj.

```
var obj = new Object();
obj.driverSeatPositionReclineSeatback = 0;
vehicle.set('personalization_driverseatposition_reclineseatback', obj, handleSuccess,
handleError);
```

3.4.6 Setting multiple vehicle data

Let's set all driver seat position.
Just add attributes to the *obj* and use the upper level type.

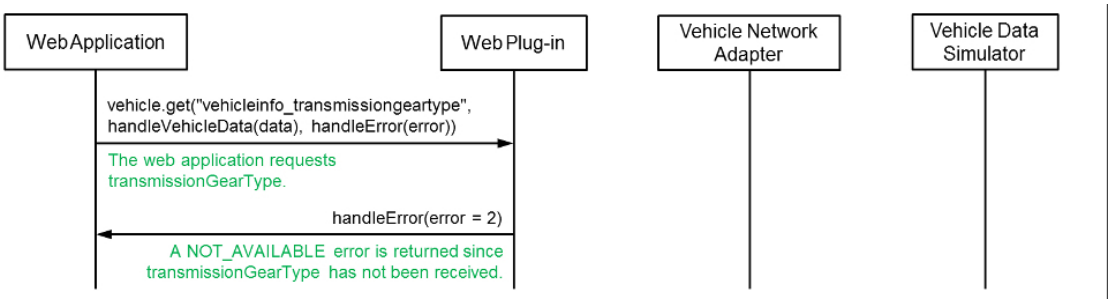
```
var obj = new Object();
obj.driverSeatPositionReclineSeatback = 0;
obj.driverSeatPositionSlide = 0;
obj.driverSeatPositionCushionHeight = 0;
obj.driverSeatPositionHeadrest = 0;
obj.driverSeatPositionBackCushion = 0;
obj.driverSeatPositionSideCushion = 0;
vehicle.set('personalization_driverseatposition', obj, handleSuccess, handleError);
```

4. Test Cases

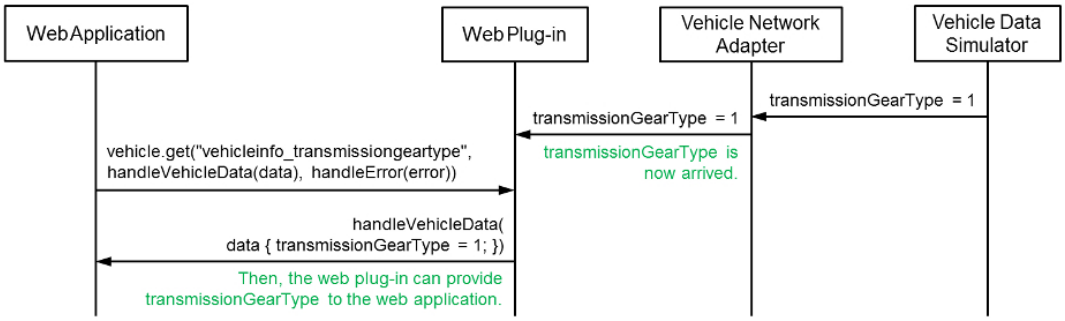
Name	Description	Input	Result Details
Getting Data	Checking whether get operation works or not. The data from vehicle bus shall be delivered to Web Application correctly.	Various attributes with test data	OK
Getting Multiple Data	Checking whether get operation works well with multiple access option. Group of multiple data shall be delivered correctly as indicated types.	Various multiple attributes	OK
Setting Data	Checking whether set operation works or not. The data from web application shall be delivered to vehicle bus correctly.	Various attributes with test data	OK
Setting Multiple Data	Checking whether set operation works well with multiple access option. Group of multiple data shall be delivered correctly as indicated types.	Various multiple attributes	OK
Getting Unsupported Data	Checking whether error handling works correctly for get operation.	Data with invalid data types	OK
Setting Unsupported Data	Checking whether error handling works correctly for set operation.	Data with invalid data types	OK
Checking Supported Types	Checking which data types are supported in the system by using getSupportedType operation	Various types	OK

4.1. Getting Data

If requested data is not received yet from INCM (Vehicle Network Adapter), an error callback with the code *NOT_AVAILABLE* is called.

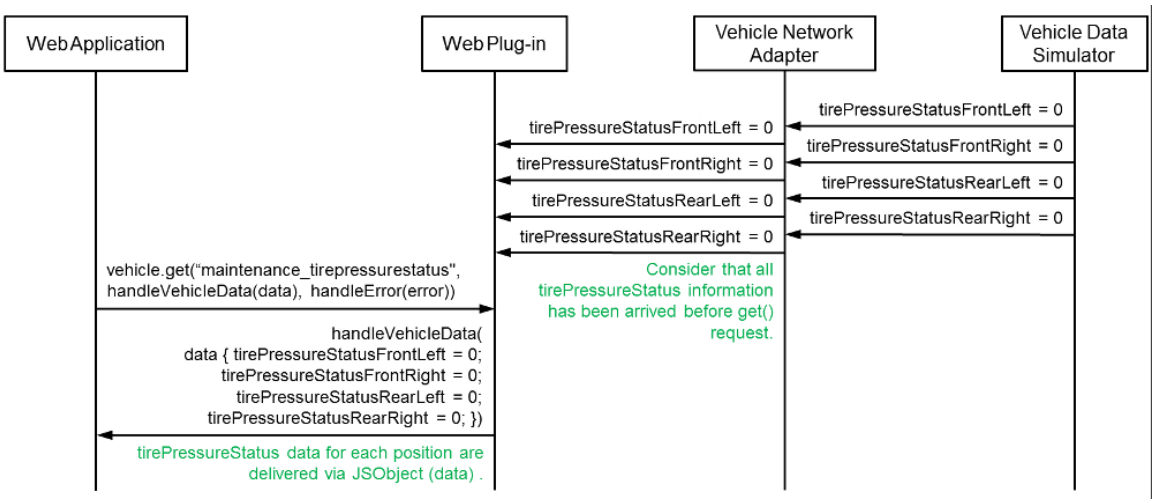


If requested data is available (since it is received already), a callback for handling vehicle data (*handleVehicleData*) is called.
A JSObject (*data*) is used to deliver the requested data to web application.



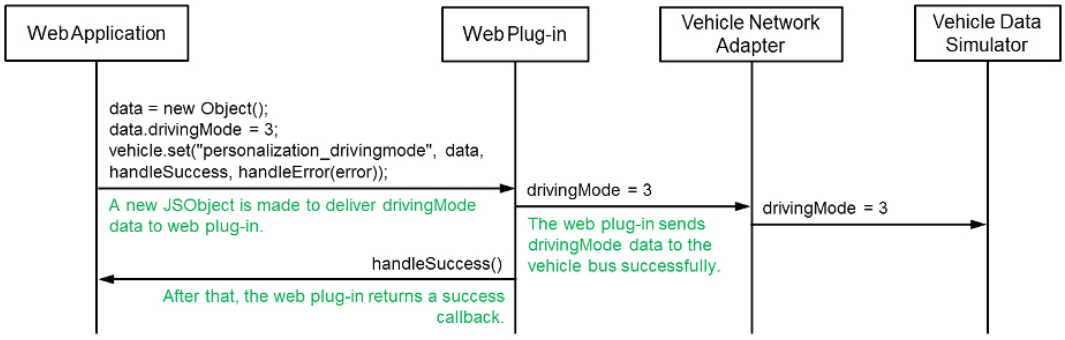
4.2. Getting Multiple Data

Multiple vehicle data are also delivered via JSObject.



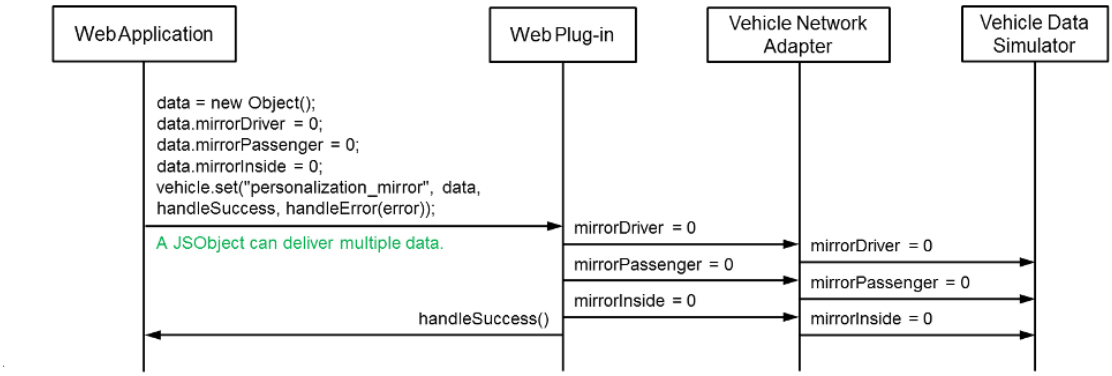
4.3. Setting Data

Only allowed vehicle data (by OEM) can be set.
The success callback means only that the set command is transferred to vehicle bus successfully, not being accepted by the target ECU and the real value is changed.



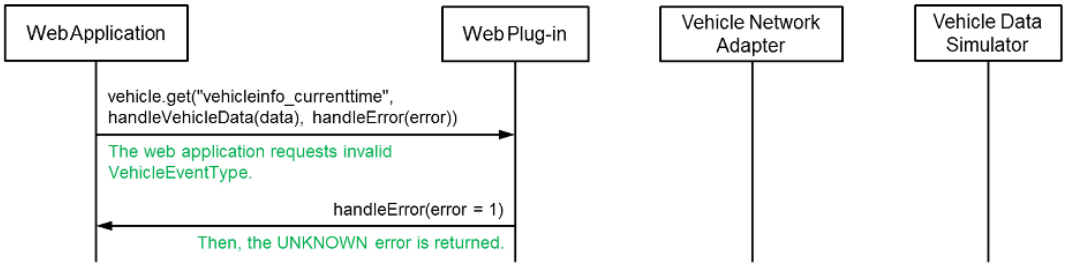
4.4. Setting Multiple Data

Setting a group of data is also allowed.



4.5. Getting/Setting unsupported data

If the requested VehicleEventType is invalid, an error callback with the code *UNKNOWN* is called.



4.6. Checking Supported Types

If the method is called with a VehicleEventType parameter, an array of all VehicleEventType objects which belong to the type is returned. 2nd parameter indicated whether the types are for writable (true) or readable (false).