

# PARSING A CAN BUS MESSAGE WITH A SCRIPT

## 1. Introduction

A Controller Area Network (CAN bus) is a robust vehicle and industrial bus standard designed to allow devices to communicate with each other without a host computer. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on copper, but it can also be used in many other contexts. For each device, the data in a frame is transmitted sequentially but in such a way that if more than one device transmits at the same time, the highest priority device can continue while the others back off. Frames are received by all devices, including by the transmitting device.

CAN bus is found commonly in vehicles, with standards for the information that is transmitted on the CAN bus being defined for heavy vehicles (SAE J1939), trailers (ISO 11992), cars (OBD2). CANopen is a standard used widely in industrial automation for control applications.

The Senquip ORB-C1 has a CAN bus peripheral that can read information off vehicle and industrial CAN networks. Using the CAN peripheral, users can capture specified messages on a CAN bus. Parsing CAN bus messages allows individual pieces of information to be extracted so that they can be turned into measurements, mathematically manipulated, and used to create exceptions. This application note discusses the parsing of CAN bus

## 2. Capturing CAN Bus Message

The Senquip ORB-C1 has a CAN bus interface that can be configured to capture specified messages on a CAN bus. A typical automotive CAN network will contain hundreds of messages, all with their own identifiers. The ORB can filter only the required messages by filling in the *ID Capture List*. Multiple of the same message can also be captured or all messages on the bus can be read. The time for which the CAN peripheral should scan can be set. For further details on configuring the CAN bus peripheral, please refer to the ORB User Guide: <http://docs.senquip.com/orbug/>.

This application note will assume that the ORB is connected to a CAN bus operating on J1939, and that the filters have been set to capture the Electronic Engine Controller 1 – EEC1 message. A brief introduction to J1939 is given below.

### 2.1. Introduction to J1939

SAE J1939 is a standard developed by the Society of Automotive Engineers (SAE) that defines communication for vehicle networks in trucks, buses, agricultural equipment, mining equipment and other heavy machinery. J1939 data messages, or packets, contain the actual data and a header. The header contains an identifier which is made up of:

- The priority of the message
- A message type indicator (PGN)
- Source address

Document Number APN0012	Revision 1.0	Prepared By NGB	Approved By NB
Title Parsing a CAN bus message with a Script			Page 2 of 8

The PGN contained in the identifier describes the message function and the data that will be contained in that message. J1939 attempts to define standard PGNS to encompass a wide range vehicle purposes. Some example PGN's include:

PGN	Description
pgn61442	Electronic Transmission Controller 1 - ETC1
pgn61444	Electronic Engine Controller 1 - EEC1
pgn64978	ECU Performance - EP
pgn65031	Exhaust Temperature - ET
pgn65102	Tachograph - TCO1
pgn65203	Fuel Information (Liquid) - LFI

Each PGN may contains multiple parameters. For example, PGN 61444, Electronic Engine Controller 1, contains the following parameters: engine torque demand, drivers demand, actual engine torque, engine speed and more. Each parameter is known as an SPN (Suspect Parameter Number). A PGN identifies a message that contains multiple SPNs.

A description of each PGN and how the SPNs are arranged within that message is given in the J1939 specification. For example, PGN 61444, Electronic Engine Controller 1 has the following description and structure.

#### PGN 61444 Electronic Engine Controller 1

Transmission repetition rate	Engine speed dependent
Data length	8
Extended data page	0
Data page	0
PDU format	240
PDU specific	4
Default priority	3

Start position	Length	Parameter Name	SPN
1.1	4 bits	Engine torque mode	899
1.5	4 bits	Actual engine - percent torque high resolution	4154
2	1 byte	Driver's demand engine - percent torque	512
3	1 byte	Actual engine - percent torque	513
4-5	2 bytes	Engine speed	190
6	1 byte	Source address of controlling device for engine control	1483
7.1	4 bits	Engine starter mode	1675
8	1 byte	Engine demand - percent torque	2432

So, if for instance, one wanted to extract engine speed (SPN190), from the PGN 61444 message, it is located in bytes 4 and 5. For a full description of SPN190, we again refer to the specification.

The description of the SPN190 message – Engine Speed is: actual engine speed which is calculated over a minimum crankshaft angle of 720 degrees divided by the number of cylinders.

#### SPN 190 – Engine Speed

Document Number APN0012	Revision 1.0	Prepared By NGB	Approved By NB
Title Parsing a CAN bus message with a Script		Page 3 of 8	
Data length	2 bytes		
Resolution	0.125 rpm/bit, zero offset		
Data range	0 to 8,031.875 rpm		
Type	Measured		
Suspect Parameter Number	190		
Parameter Group Number	[61444]		

From the full description, we see that the engine speed contained in bytes 4 and 5 of the PGN needs to be multiplied by 0.125 to determine the RPM. The process of determining the RPM now becomes:

- Capture the Electronic Engine Controller 1 message (PGN 61444)
- Extract SPN 190 from bytes 4 and 5
- Multiply the SPN by 0.125

These operations are perfect for a script running on an ORB.

You may have noticed that we are looking to capture PGN 61444, however capturing is done based on CAN bus identifiers. Here is a quick introduction into converting a PGN into an identifier. There are many online tools that accomplish this easily.

The CAN identifier for J1939 is 29 bits and is made up of 3 bits of priority, 18 bits of PGN and 8 bits of source address. The priority is defined in the specification and is 3 for this message. The PGN we know is 61444. The source address is the address of the module sending the message and is assumed to be 0 in this case. The source address can be obtained by doing a scan of all messages on the bus and determining the source address from the last 8 bits.

Part	Bits	Value	Binary
Priority	3	3	011
PGN	18	61444	00 1111 0000 0000 0100
Source Address	8	0	0000 0000

By combining the 3 components of the identifier, we get 011 00 1111 0000 0000 0100 0000 0000 or in decimal, 217056256, or hex CF00400. This the number that can be used as a filter to specify the Electronic Engine Controller 1 message in the ORB CAN settings and is the number that will be used in a script to identify the message.

### 3. Parsing the ASCII String

We will assume that the Electronic Engine Controller 1 message is being correctly received by the CAN peripheral on the Senquip ORB. Figure 2 shows the Electronic Engine Controller 1 message arriving at the Senquip Portal. Note the id is shown as hex 0CF00400 as expected. By looking at the *Raw Data* window, or referring to the ORB documentation, we note in Figure 1 that the JSON key for the CAN peripheral is can1. The key will be used to retrieve the data from the ORB.

Document Number APN0012	Revision 1.0	Prepared By NGB	Approved By NB
Title Parsing a CAN bus message with a Script			Page 4 of 8

```
"can1": [
  {
    "data": "FFFFFF6813FFFFFF",
    "id": 217056256
  }
]
```

Figure 1 – CAN message in raw data

Let us assume in our example, that we want to extract engine speed from the Engine Controller 1 message. We learned before that engine speed is in position 4 and 5 of the data field, 0x68 and 0x13 in this case. Taking the decimal form of the HEX value 1368 (Intel byte order), we get 4968 in decimal. To arrive at the RPM, we conduct a scaling of this value using the offset 0 and the scale 0.125 RPM/bit. The physical engine speed is found to be 621 RPM.

Once parsed and converted to km/h, the engine speed will be dispatched to the Senquip Portal for display and storage.

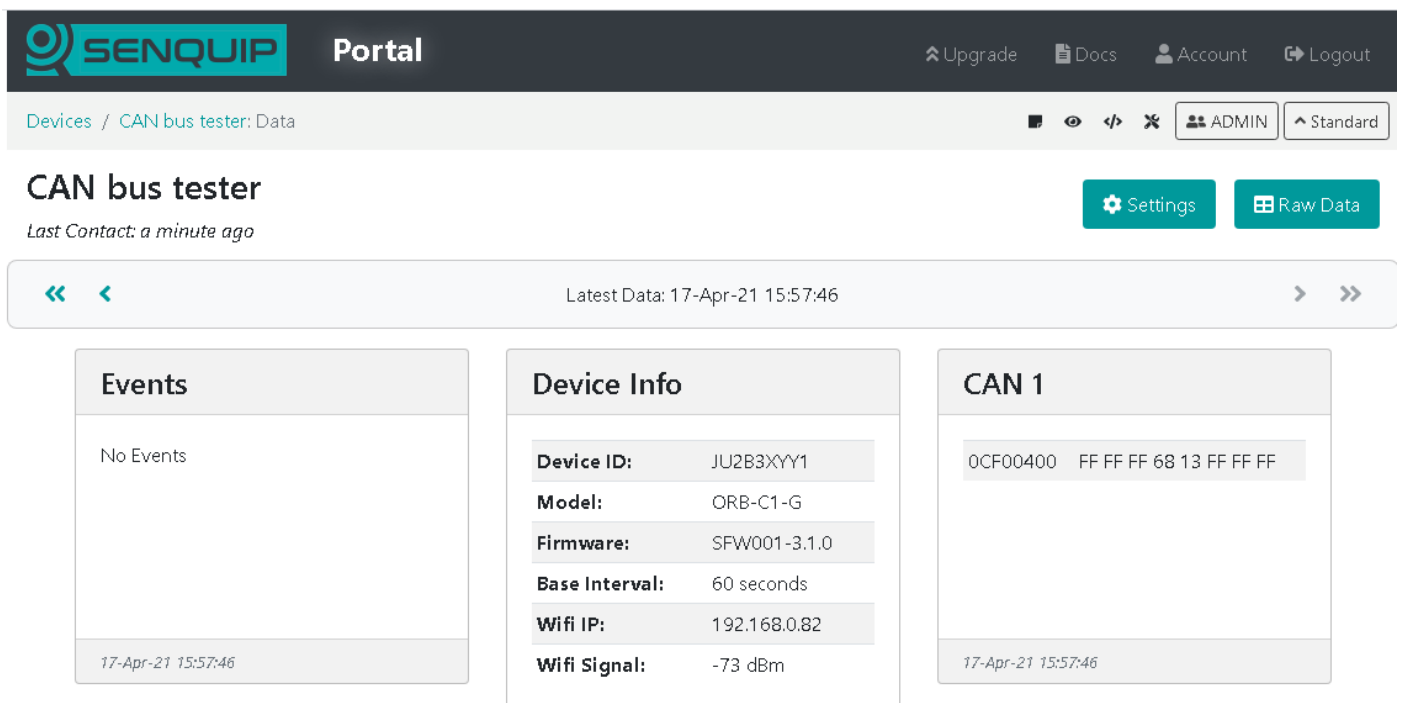


Figure 2 - CAN Electronic Engine Control message being received by an ORB

## 4. Writing the Script

In this application note, we will assume that you have access to the scripting feature, that the ORB is subscribed to a Premium plan, and that you are familiar with the scripting environment on the Senquip Portal. For further details on scripting on the Senquip ORB, please see the scripting guide: [https://docs.senquip.com/scripting\\_guide](https://docs.senquip.com/scripting_guide).

To be able to monitor the engine speed, a custom parameter, Engine Speed, will be generated and displayed on the Senquip Portal. Figure 3 shows the creation of the custom parameter cp1 (RPM).

Document Number  
APN0012

Revision  
1.0

Prepared By  
NGB

Approved By  
NB

Title  
Parsing a CAN bus message with a Script

Page  
5 of 8

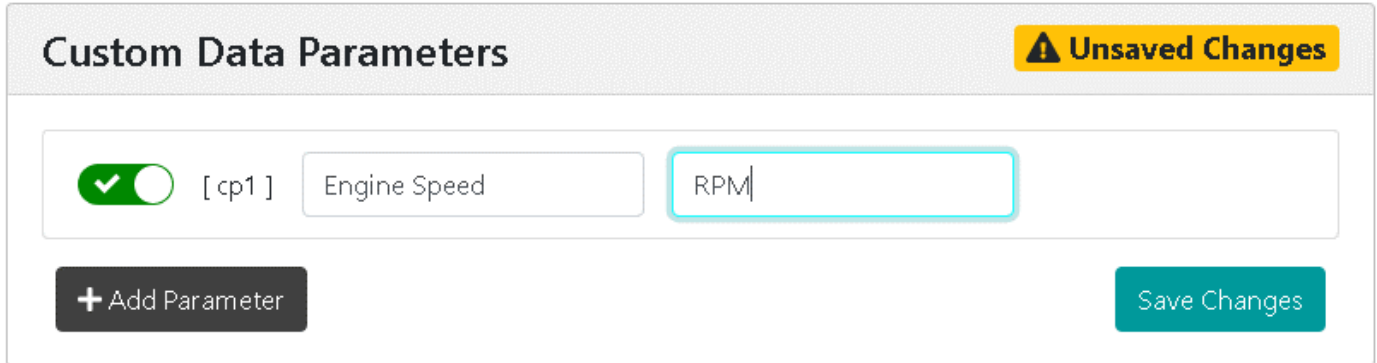
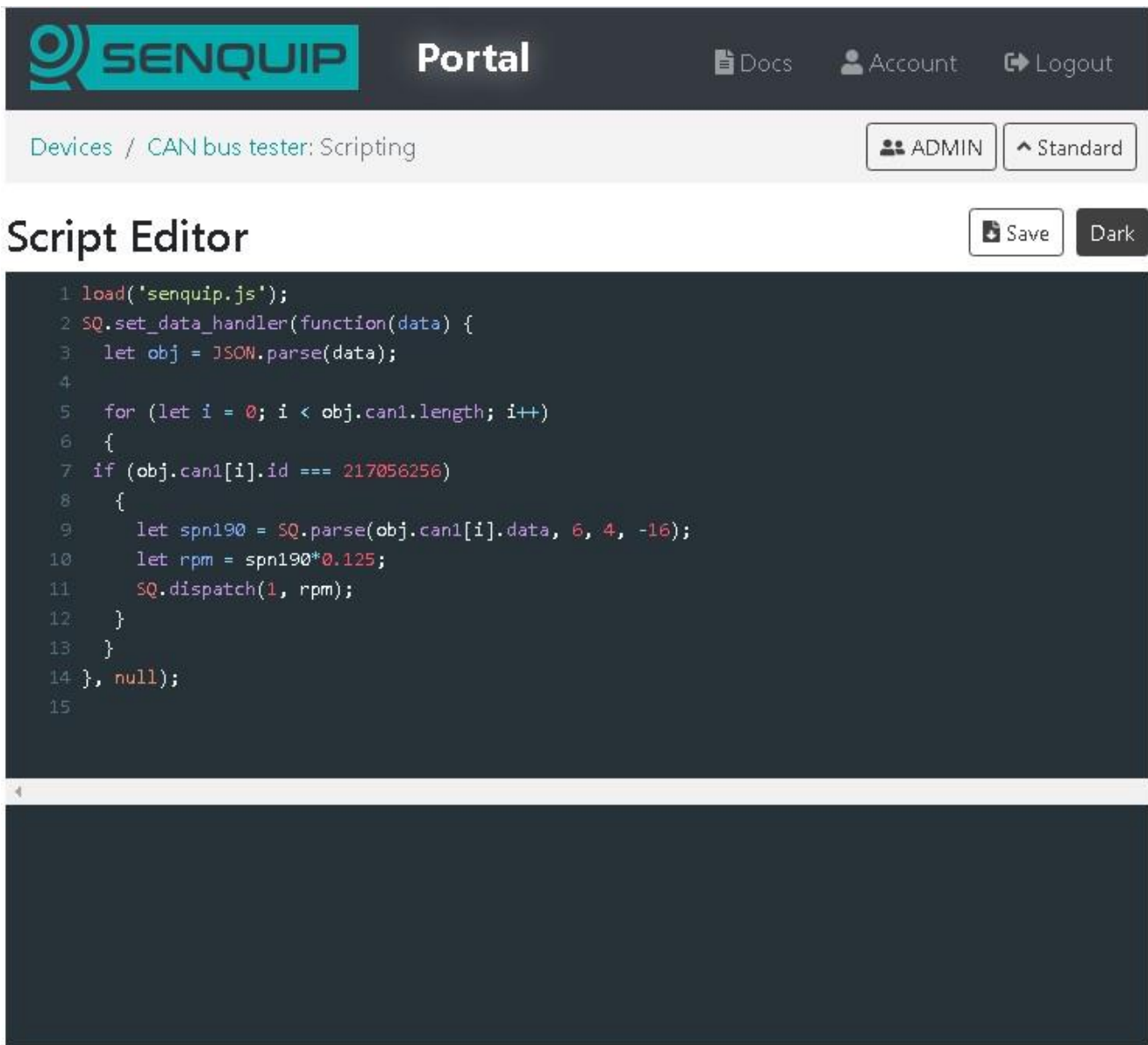


Figure 3 – A custom parameter is created to show Engine Speed

Figure 4 shows the script to parse the Electronic Engine Controller 1 message and convert the engine speed to RPM. A brief explanation for each line is given below.

- 1) Load the Senquip Library that includes all the functions prefixed with SQ
- 2) Setup the function that will be executed at each base interval, after the measurements have been taken
- 3) Parse the JSON data to create an object that can be indexed by the JSON key, can1 in this case
- 5) Cycle through all of the received CAN messages and for each one, parse and scale the required data values
- 7) Identify the Electronic Engine Controller 1 message by its identifier (217056256)
- 9) Extract bytes 4 and 5 in Intel order by starting at character 6 (6) in the message, taking 4 characters (6813), identifying them as hex and reversing their order (0x1368).
- 10) Use the scaling factor of 0.125 as specified for the SPN to convert to revs per minute
- 11) Dispatch the engine speed in rpm to the Senquip Portal to be shown as custom parameter 1.
- 14) Close the function



```
1 load('senquip.js');
2 SQ.set_data_handler(function(data) {
3   let obj = JSON.parse(data);
4
5   for (let i = 0; i < obj.can1.length; i++)
6   {
7     if (obj.can1[i].id === 217056256)
8     {
9       let spn190 = SQ.parse(obj.can1[i].data, 6, 4, -16);
10      let rpm = spn190*0.125;
11      SQ.dispatch(1, rpm);
12    }
13  }
14 }, null);
15
```

Size: 311 / 12188 bytes [Scripting Docs]

Reboot

Download

Figure 4 - Script to parse the electronic engine controller 1 message

Once the script is downloaded and the ORB is rebooted, the function runs, and the portal now shows engine speed in revs per minute.

Document Number  
APN0012

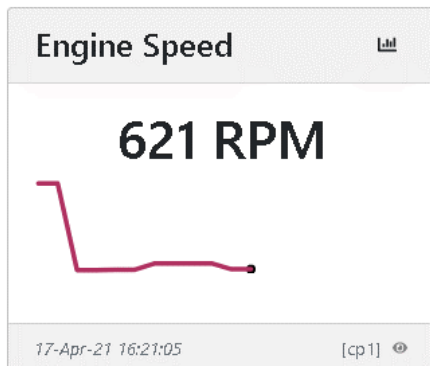
Revision  
1.0

Prepared By  
NGB

Approved By  
NB

Title  
Parsing a CAN bus message with a Script

Page  
7 of 8



The code as presented has been written to be simple. It is unnecessarily verbose and an effort should be made to compress the code to conserve memory on the ORB.

The code should be further developed to perform checks on the data to ensure integrity and to protect the JavaScript interpreter from invalid data.

## 5. Conclusions

Users can write their own JavaScript for the Senquip ORB. Scripting allows CAN messages to be parsed to extract components. Those components can then be converted to numbers, which allows them to be manipulated mathematically, and to create complex exceptions. The calculated values can be dispatched to the Senquip Portal or another server for display and storage.

Document Number  
APN0012

Revision  
1.0

Prepared By  
NGB

Approved By  
NB

Title  
Parsing a CAN bus message with a Script

Page  
8 of 8

## APPENDIX 1 – SOURCE CODE

```
load('senquip.js');
SQ.set_data_handler(function(data) {
  let obj = JSON.parse(data);

  for (let i = 0; i < obj.can1.length; i++)
  {
    if (obj.can1[i].id === 217056256)
    {
      let spn190 = SQ.parse(obj.can1[i].data, 6, 4, -16);
      let rpm = spn190*0.125;
      SQ.dispatch(1, rpm);
    }
  }
}, null);
```