# CONNECTING TO A SPARTANLYNC TPMS

## 1. Introduction

The SpartanLync tire pressure and temperature monitoring system, Spartan Guard TPMS, has been developed specifically for industrial and commercial use. By increasing safety, reducing tire maintenance costs as well as fuel expenses by monitoring vehicle tires, Spartan Guard TPMS meets the challenges of fleet management.

SpartanLync sensors operate at ISM frequencies, ensuring excellent range and battery life. Signals transmitted by each in-tire sensor are received by antennas on the truck and trailer and are transmitted to the in-cab display over J1939 CAN.

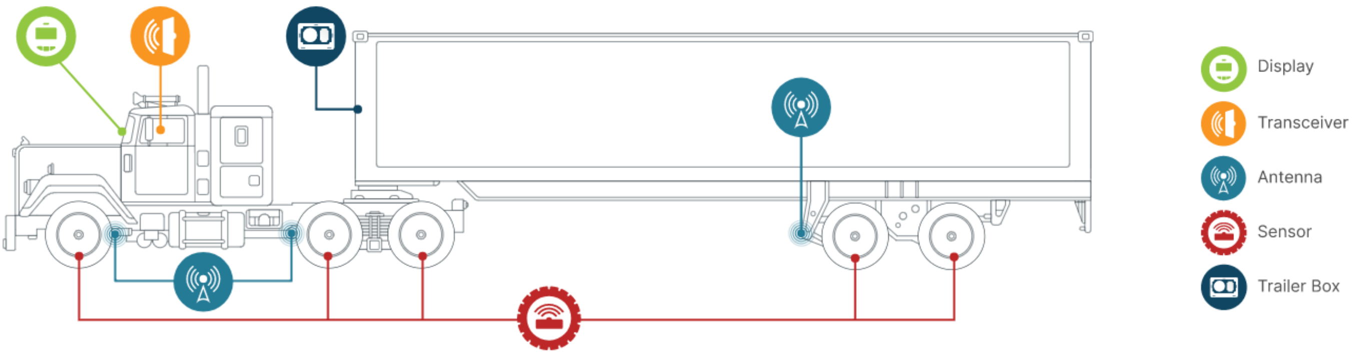



*Figure 1 - typical Setup on Larger Vehicles Such as Transport Trucks.*

Senquip devices can connect to the SpartanLync CAN bus and read individual tire data, making it available online. This application note describes how to connect to the SpartanLync CAN, and details the script required to run on the Senquip device.


*Figure 2 - SpartanLync Display shows Tire Pressure and Temperature Data in Real-Time.*

It is assumed that the user has Admin privileges and scripting rights for the device being worked on. To request scripting rights, contact [support@senquip.com](mailto:support@senquip.com).

## 2. References

The following documents were used in compiling this Application Note.

| Reference | Document | Document Number |
|---|---|---|
| A | Valor TPMS Gauge (1702) C-COM 2G Specifications | Rev 1702D |
| B | CAN ID Calculator | SOF002 Rev 1.0 |
| C | SAE International Surface vehicle Recommended Practice | SAE J1939-71 Dec 2003 |
| D | Parsing a CAN bus message with a Script. | APN0012 |

## 3. Install

The Senquip device needs to connect to the SpartanLync J1939 CAN. The SpartanLync display has the pinout shown in Figure 3. In this application note, a Senquip ORB with a single CAN interface will be used. If a second CAN interface to the vehicle CAN is required, a Senquip QUAD with 2 CAN interfaces could be used.

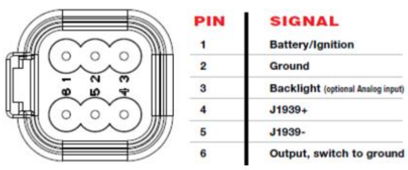



*Figure 3 - Electrical Connections*

Figure 4 shows how to connect a Senquip ORB to a SpartanLync TPMS. A 120R termination resistor is placed at either end of the CAN network.

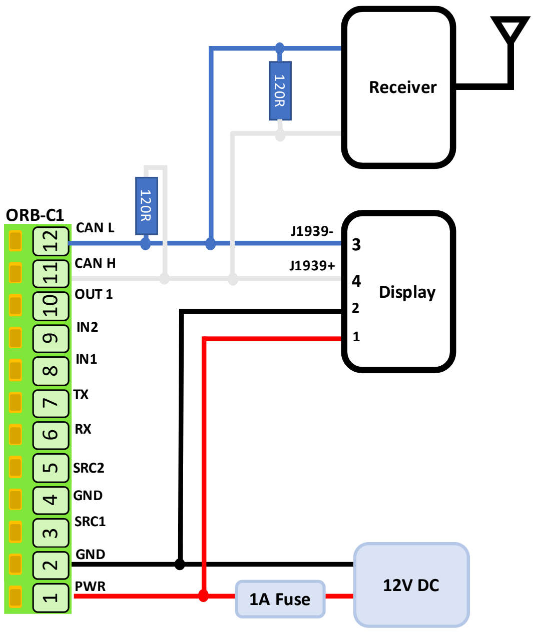



*Figure 4 - Connecting a Senquip ORB to a SpartanLync TPMS*

## 4. Senquip ORB Setup

The Senquip ORB was configured for 10 second updates and the CAN peripheral was enabled with a scan time of 10 seconds. Baud rate was specified in the display manual as 500kbps however a sticker on the receiver stated 250kbps and this was the correct speed.

A filter was placed to allow 16 tire condition messages to be received, allowing a maximum of 16 tires to be connected. Raw data was transmitted to the Senquip Portal for diagnostic purposes. It is recommended to turn this off later to save on data costs.

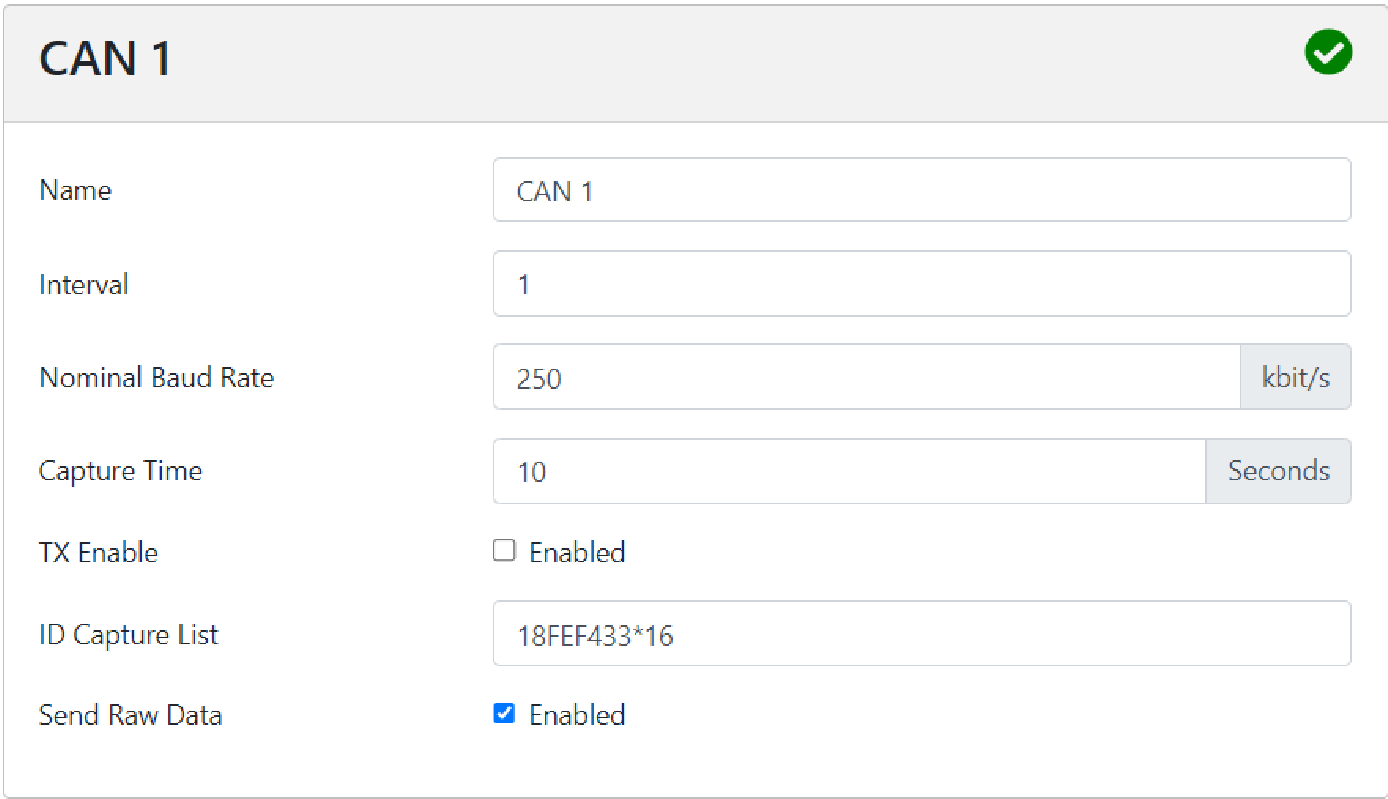



*Figure 5 - Senquip ORB CAN Setup*

From the display manual, the tire condition message is given as PGN 65268. More detail on this message is available in the J1939 specification.

| | | |
|---|---|---|
| **Tire Condition** (TIRE)<br>**65268** ($00FEF4_{16}$)<br><br>SA= 51<br>DA=N/A | **(929)** Tire Location<br>**(241)** Tire Pressure<br>**(242)** Tire Temperature<br>**(1699)** CTI Wheel Sensor Status<br>**(1698)** CTI Tire Status<br>**(2587)** Tire Pressure Threshold Detection | Used for Main Screen display.<br>*Tire Pressure resolution used is 5.5 kPa/bit. |

*Figure 6 - Display Manual Tire Condition Message Specification*

## *pgn65268 - Tire Condition - TIRE -*

| | | | |
| --- | --- | --- | --- |
| Transmission Repetition Rate: | | 10 s | |
| Data Length: | | 8 bytes | |
| Data Page: | | 0 | |
| PDU Format: | | 254 | |
| PDU Specific: | | 244 | |
| Default Priority: | | 6 | |
| Parameter Group Number: | | 65268 ( $00FEF4_{16}$ ) | |
| Bit Start Position /Bytes | Length | SPN Description | SPN |
| 1 | 8 bits | Tire Location | 929 |
| 2 | 1 byte | Tire Pressure | 241 |
| 3-4 | 2 bytes | Tire Temperature | 242 |
| 5.1 | 2 bits | CTI Wheel Sensor Status | 1699 |
| 5.3 | 2 bits | CTI Tire Status | 1698 |
| 5.5 | 2 bits | CTI Wheel End Electrical Fault | 1697 |
| 6-7 | 2 bytes | Tire Air Leakage Rate | 2586 |
| 8.6 | 3 bits | Tire Pressure Threshold Detection | 2587 |

Tire Condition Message NOTE: Message has to repeated as necessary to transmit all available information. This method of location identification requires individual SPNs to be assigned to report failures specific to each individual component (I.e. each tire, each axle, etc.).

*Figure 7 - J1939 Tire Condition Message Specification*

To calculate the filter value for the message in the Senquip CAN setup, the following values were used.

- PGN: 65268 – from the display datasheet and J1939 specification.
- Source Address: 51 – from the display datasheet.
- Priority: 6 – from the J1939 PGN specification

Putting these values into the [Senquip CAN ID Calculator]() gives a message ID of 18FEF433 in hex.

In reality, a simpler way was to scan the CAN bus and look for messages and then use the Senquip CAN ID Calculator to identify the PGNs in each message.

**CAN ID Calculator Rev 1.0**

| Message ID to PGN | | | |
|---|---|---|---|
| | | HEX | DEC |
| INPUT: | Message ID (HEX) | 18FEF433 | 419361843 |
| | | HEX | DEC |
| OUPUT: | PGN (DEC) | FEF4 | 65268 |
| | Source Address (DEC) | 33 | 51 |
| | Priority (DEC) | 6 | 6 |

| PGN to Message ID | | | |
|---|---|---|---|
| | | DEC | HEX |
| INPUT: | PGN (DEC) | 65268 | FEF4 |
| | Source Address (DEC) | 51 | 33 |
| | Priority (DEC) | 6 | 18 |
| | | DEC | HEX |
| OUTPUT: | Message ID (HEX) | 419361843 | 18FEF433 |

*Figure 8 - Senquip CAN ID Calculator*

Further detail on the message is available from the J1939 specification for each SPN. Interestingly the pressure resolution is specified as 4 kPa/bit in the J1939 specification but is noted as 5.5 kPa/bit in the display specification. We have used 5.5 kPa as our pressure scale.

***spn241 - Tire Pressure -*** Pressure at which air is contained in cavity formed by tire and rim.

| | |
|---|---|
| Data Length: | 1 byte |
| Resolution: | 4 kPa/bit , 0 offset |
| Data Range: | 0 to 1000 kPa |
| Type: | Measured |
| Suspect Parameter Number: | 241 |
| Parameter Group Number: | [65268] |

*Figure 9 - Tire Pressure SPN Specification*

***spn242 - Tire Temperature -*** Temperature at the surface of the tire sidewall.

| | |
|---|---|
| Data Length: | 2 bytes |
| Resolution: | 0.03125 deg C/bit , -273 deg C offset |
| Data Range: | -273 to 1735 deg C |
| Type: | Measured |
| Suspect Parameter Number: | 242 |
| Parameter Group Number: | [65268] |

*Figure 10 - Tire Temperature SPN Specification*

***spn929 - Tire Location -*** To identify to which of several similar devices (such as tires or fuel tanks) the information applies.
The low order 4 bits represent a position number, counting left to right when facing in the direction of normal vehicle travel (forward).
The high order 4 bits represent a position number, counting front to back on the vehicle.
The value 0xFF indicates not available.
It is recommended that output devices add 1 to the position number (range 1 to 15, not 0 to 14) for use by drivers and service technicians.
Examples: Tire pressure for location 0x00 would be left front tire.
Tire pressure for location 0x23 would be right outside rear rear on a 3-axle tractor with dual axle per side (3rd axle, 4th tire).

| | |
|---|---|
| Bit Length: | 8 bits |
| Type: | Measured |
| Suspect Parameter Number: | 929 |
| Parameter Group Number: | [ 65268 ] |

*Figure 11 - Tire Location SPN Specification*

## 5. Writing the Script to Interpret the CAN Data

A script has been written to parse the CAN messages to extract individual tyre data. Various libraries that are used by the script are loaded and an object, *tires*, is created. Tires will be used to store the tyre location (id), pressure, and temperature of each tire.

```
load('senquip.js');
load('api_sys.js');
load('api_math.js');

let tires=[]; // object containing tire data
```

In the main handler, a loop is generated to run through all the CAN messages received by the Senquip ORB in the last measurement cycle. Each CAN message is broken up into individual SPNs representing tire location, pressure, and temperature. Other SPNs are available but that are not used in this application note. For further information on parsing CAN messages, see Senquip application note [APN0012 - Parsing a CAN bus message with a Script](APN0012 - Parsing a CAN bus message with a Script).

Parsed data is passed into an object *sensor* and is passed to a function *loadData* to be loaded into the tires object.

```
SQ.set_data_handler(function(data) {
  let obj = JSON.parse(data);

  if (typeof obj.can1 !== "undefined") {
    for (let i = 0; i < obj.can1.length; i++) {  // Run through all the received CAN messages

      // Look for the specific PGN:
      if (obj.can1[i].id === 0x18FEF433) { // PGN: 65268  - Tire Condition
        let spn929 = SQ.parse(obj.can1[i].data, 0, 2, -16); // tire location
        let spn241 = 5.5*SQ.parse(obj.can1[i].data, 2, 2, -16); // tire pressure, 5.5 kPa per bit
        let spn242 = 0.03125*SQ.parse(obj.can1[i].data, 4, 4, -16)-273; // tire temperature, 0.03125 deg C/bit , -273 deg C offset

        let sensor = {id: spn929,
                      pres:spn241,
                      temp:spn242};

        loadData(sensor);  // This function loads the data into the tires object
      }
    }
  }
```

The *LoadData* function searches through the tires object to see if the received tire location has been found before. If it has, the latest pressure and temperature data is loaded. If not, a new tire entry is added to the *tires* object.

```
let tires=[]; // object containing tire data

function loadData(sensor){ // Validates that the tire already exists and loads latest data
  for (let j = 0; j < tires.length; j++) {
    if (tires[j].id === sensor.id){ // If the id if found, load the latest data
    tires[j].pres = sensor.pres;
    tires[j].temp = sensor.temp;
    return sensor.id;
    }
  }
  tires.push(sensor); // If not found, create a new entry
}
```

After the latest data is loaded into the tires object, we return to the main data handler and proceed to dispatch the data to the Senquip Portal in summary form and to gauge widgets. The pressure and temperature in the summary message are rounded before sending.

```
      loadData(sensor);  // This function loads the data into the tires object
    }
  }
}

for (let j = 0; j < tires.length; j++) {
  if (typeof tires[j].pres === "number" && typeof tires[j].temp === "number"){ //invalid data will be converted to FAULT text
    let msg = "Pressure:    "+JSON.stringify(Math.round(tires[j].pres))+" kPa"+"\nTemperature: "+JSON.stringify(Math.round(tires[j].temp))+" C  ";
    SQ.dispatch(3*j+1,msg); // Dispatch each tire detail to a seperate widget on the Senquip Portal
    SQ.dispatch(3*j+2,tires[j].pres); // for a numerical pressure chart
    SQ.dispatch(3*j+3,tires[j].temp); // for a numerical temp chart
  }
  else{
    SQ.dispatch(3*j+1,"Not Connected"); // Error message
  }

}

}, null);
```

Data dispatched to the portal is shown as shown in Figure 12. Raw CAN data is left on for diagnostic purposes. GPS and pitch and roll from inbuilt sensors on the Senquip ORB are also shown.

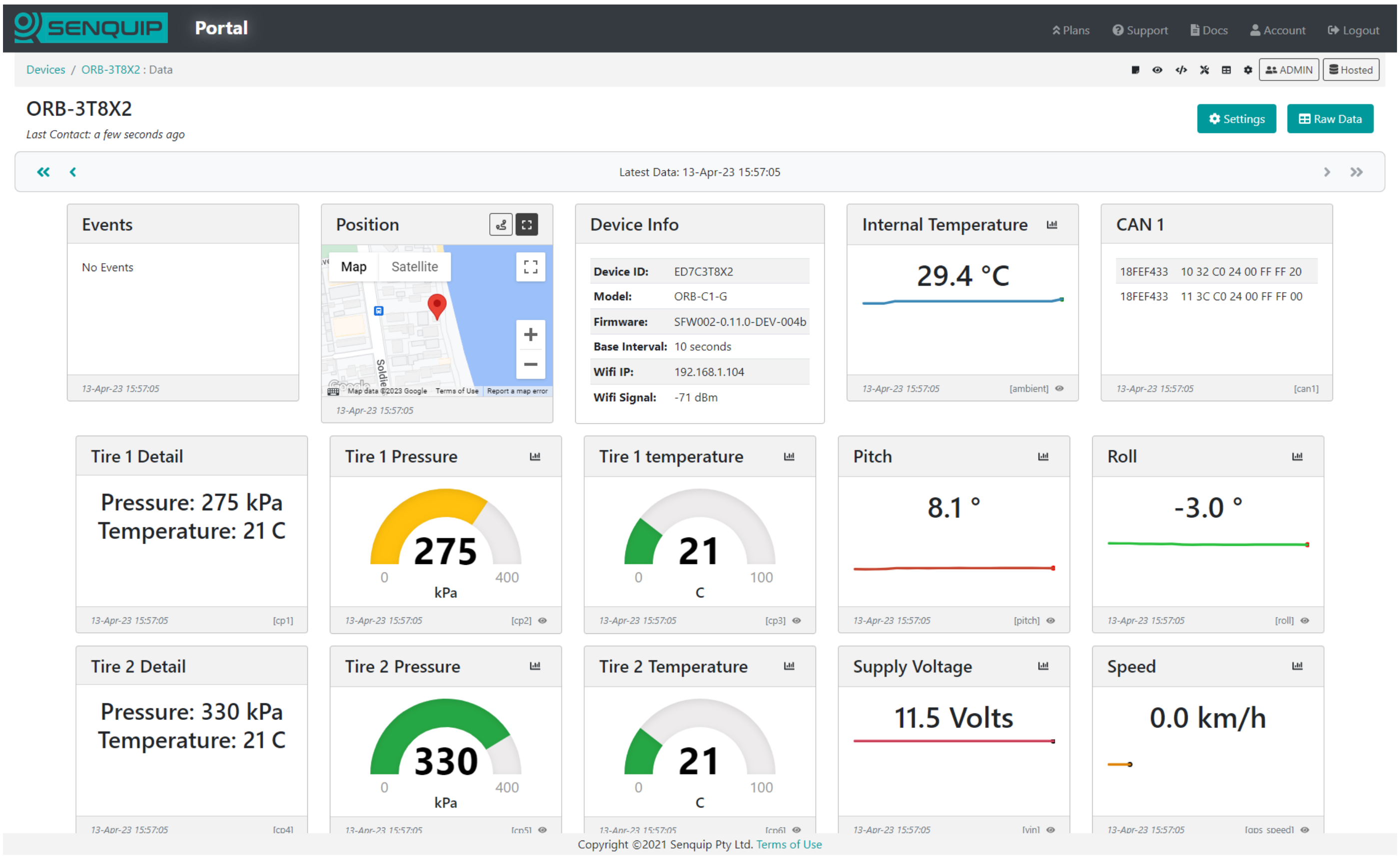
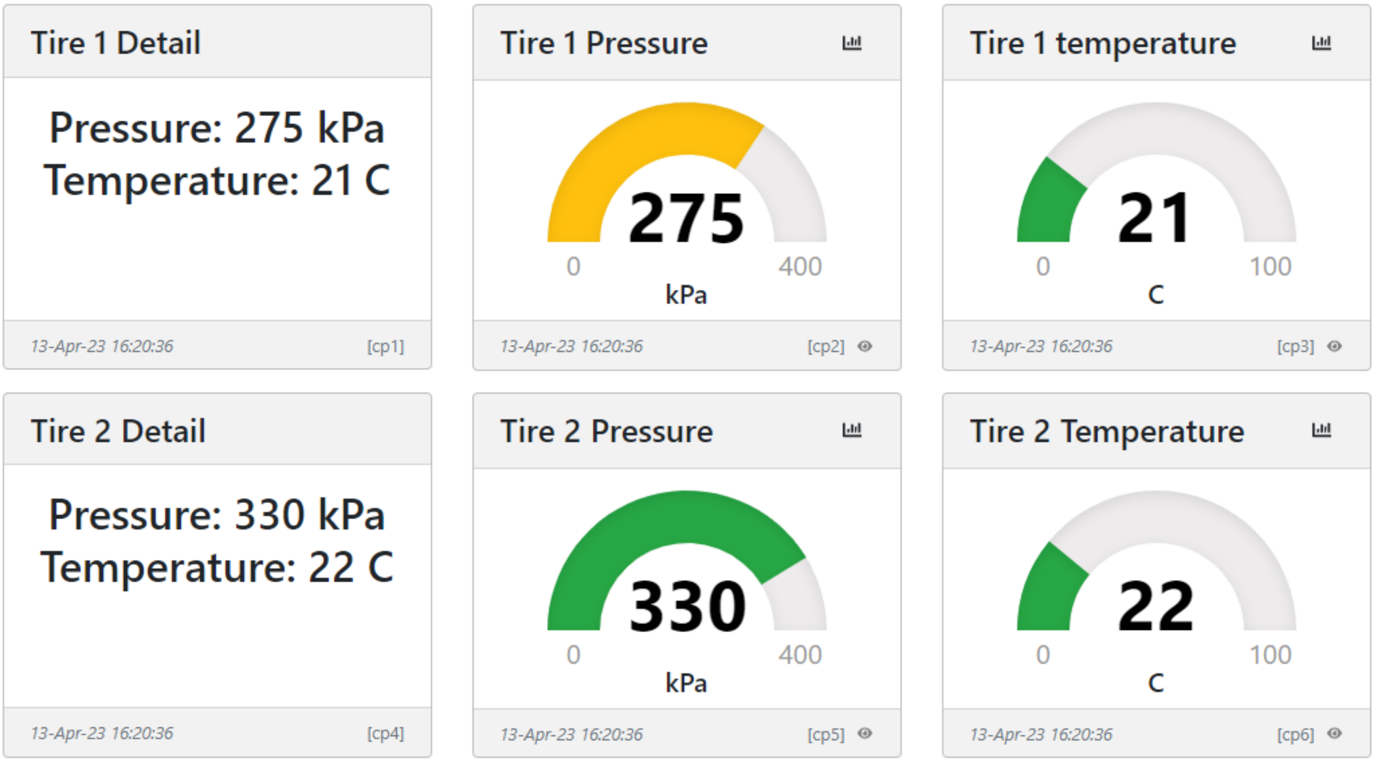



*Figure 12 - TPMS Display on the Senquip Portal.*

## 6. Conclusion

A Senquip ORB has been connected to the J1939 CAN bus available on a SpartanLync and is able to read pressure and temperate data and dispatch that data to the Senquip Portal.



## Appendix 1: Source Code

```
load('senquip.js');
load('api_sys.js');
load('api_math.js');

let tires=[]; // object containing tire data

function loadData(sensor){ // Validates that the tire already exists and loads latest data
  for (let j = 0; j < tires.length; j++) {
    if (tires[j].id === sensor.id){ // If the id if found, load the latest data
    tires[j].pres = sensor.pres;
    tires[j].temp = sensor.temp;
    return sensor.id;
    }
  }
  tires.push(sensor); // If not found, create a new entry
}

SQ.set_data_handler(function(data) {
  let obj = JSON.parse(data);

  if (typeof obj.can1 !== "undefined") {
    for (let i = 0; i < obj.can1.length; i++) {  // Run through all the received CAN messages

      // Look for the specific PGN:
      if (obj.can1[i].id === 0x18FEF433) { // PGN: 65268  - Tire Condition
        let spn929 = SQ.parse(obj.can1[i].data, 0, 2, -16); // tire location
        let spn241 = 5.5*SQ.parse(obj.can1[i].data, 2, 2, -16); // tire pressure, 5.5 kPa per bit
        let spn242 = 0.03125*SQ.parse(obj.can1[i].data, 4, 4, -16)-273; // tire temperature, 0.03125 deg C/bit , -273
deg C offset

        let sensor = {id: spn929,
```

```
                        pres:spn241,
                        temp:spn242};

          loadData(sensor);  // This function loads the data into the tires object
        }
      }
    }

  for (let j = 0; j < tires.length; j++) {
    if (typeof tires[j].pres === "number" && typeof tires[j].temp === "number"){ //invalid data will be converted to 
FAULT text
      let msg = "Pressure:    "+JSON.stringify(Math.round(tires[j].pres))+" kPa"+"\nTemperature: 
"+JSON.stringify(Math.round(tires[j].temp))+" C  ";
      SQ.dispatch(3*j+1,msg); // Dispatch each tire detail to a seperate widget on the Senquip Portal
      SQ.dispatch(3*j+2,tires[j].pres); // for a numerical pressure chart
      SQ.dispatch(3*j+3,tires[j].temp); // for a numerical temp chart
    }
    else{
      SQ.dispatch(3*j+1,"Not Connected"); // Error message
    }

  }

}, null);
```