INTEGRATION WITH ENOVATION MPC-20

1. Overview

The Enovation PowerCore MPC-20 is a compact, intelligent engine controller designed for diesel and spark-ignited engines in both mobile and stationary applications. Commonly used in irrigation, pumping, and generator control systems, the MPC20 provides precise engine management, comprehensive protection, and flexible communication capabilities.

Integrating a Senquip device with an Enovation MPC-20 allows operators to monitor and control their engines remotely, viewing key operating parameters and fault conditions in real time via the Senquip Portal or a third-party SCADA system. This integration improves asset utilisation, reduces downtime, and enables proactive maintenance, especially valuable for equipment deployed in remote or unattended locations.

Figure 1 – Enovation PowerCore MPC-20 Controller

This application note describes how to interface a Senquip telemetry device with an Enovation MPC-20 via Modbus RTU over RS485. Through this connection, the Senquip device can read engine and system data, log it to the cloud, and send remote start/stop or auto/manual commands to the controller. The concepts demonstrated are applicable to other Enovation engine controllers that use a similar Modbus register map and communications architecture.

The following sections provide details on wiring, configuration, and the Senquip device script required to achieve full monitoring and control.

***Disclaimer**: The information provided in this application note is intended for informational purposes only. Users of the remote machine control system described herein should exercise caution and adhere to all relevant safety guidelines and regulations. By utilising the information provided in this application note, users acknowledge their understanding and acceptance of the associated risks. The authors and contributors disclaim any warranties, expressed or implied, regarding the accuracy or completeness of the information presented.*

2. Wiring the Senquip Device to the MPC-20

In this application note, the RS485 serial port on a Senquip ORB is used to interface with the FCI port on the rear of the Enovation MPC-20 controller. This connection enables Modbus RTU communication between the controller and the Senquip device.

Pin #	Pin Assignments	Pin #	Pin Assignments	Pin #	Pin Assignments
1	SW_BATT	31	BATT	61	BATT
2	GND	32	GND	62	GND
3	AN1_IN	33	DIG_IN1	63	DIG_IN2
4	AN2_IN	34	DIG_IN3	64	DIG_IN4
5	AN3_IN	35	DIG_IN5	65	DIG_IN6
6	AN4_IN	36	Reserved	66	HS_2A_OUT_1
7	AN5_IN	37	Reserved	67	HS_2A_OUT_2
8	AN6_IN	38	Reserved	68	Reserved
9	AN7_IN	39	Reserved	69	FREQ_IN
10	AN8_IN	40	AN_OUT	70	GND
11	Reserved	41	Reserved	71	Reserved
12	RS485_L	42	Reserved	72	CAN1L
13	RS485_H	43	Reserved	73	CAN1H
14	ETH_TD_P	44	ETH_TD_N	74	CAN2L
15	ETH_RD_P	45	ETH_RD_N	75	CAN2H
16	USB_DP_OUT	46	USB_DM_OUT	76	USB1_VBUS
17	USB_GND	47	USB_SHLD	77	USB_ID_OUT
18	RLY4_NC	48	Reserved	78	RLY1_NC
19	RLY4_COM	49	Reserved	79	RLY1_COM
20	RLY4_NO	50	Reserved	80	RLY1_NO
21	Reserved	51	Reserved	81	Reserved
22	RLY5_NC	52	Reserved	82	RLY2_NC
23	RLY5_COM	53	Reserved	83	RLY2_COM
24	RLY5_NO	54	Reserved	84	RLY2_NO
25	Reserved	55	Reserved	85	Reserved
26	RLY6_NC	56	Reserved	86	RLY3_NC
27	RLY6_COM	57	Reserved	87	RLY3_COM
28	RLY6_NO	58	Reserved	88	RLY3_NO
29	LS_1A_OUT_1	59	Reserved	89	HS_5V_OUT_1
30	LS_1A_OUT_2	60	Reserved	90	HS_5V_OUT_2

Figure 1 – FCI Pin specification

The following connections are required:

Connection	Senquip ORB	MPC-20 FCI Pin
RS485 A	Pin 7, A / TX	13, RS485_H
RS485 B	Pin 6, B / RX	12, RS485_L
GND	Pin 2, GND	2, GND
Switched PWR	Pin 1, PWR +	1, SW_BATT

If a screened wire is available, the screen should be connected to either the Senquip or the MPC-20 controller ground but not both. Connecting to both can create a ground loop which will be susceptible to magnetic fields.

Note: The MPC-20 documentation does not specify an internal RS485 termination resistor. If the cable length between the controller and the Senquip device exceeds a few metres or communication is unreliable, fit a 120 Ω

termination resistor across the RS485 A and B lines. The Senquip ORB and QUAD devices allow internal termination to be enabled via configuration.

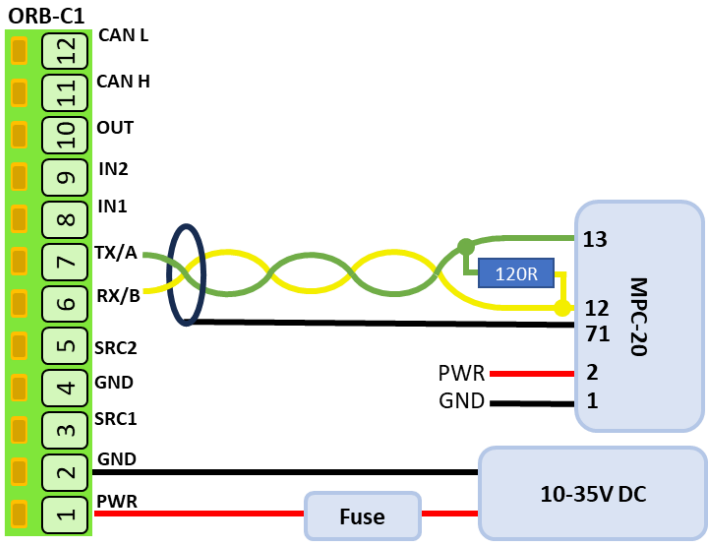

Figure 2 - Senquip ORB to MPC-20 Wiring

3. Senquip Device Configuration

We get the communications specification for the MPC-20 controller from the PowerCore® MPC-20-R2 Engine Controller Operations Manual. The manual contains panel setup information, and a Modbus register map.

The default serial port settings are:

Parameter	Value
Interface	RS485
Bit rate	19200 bps
Data bits	8
Stop bits	1
Parity	None
Protocol	Modbus RTU
Address	1

Note that if multiple controllers are on the same RS485 network, they will each need a different Modbus Address.

The serial port on the Senquip device is configured to mirror the MPC-20 default settings:

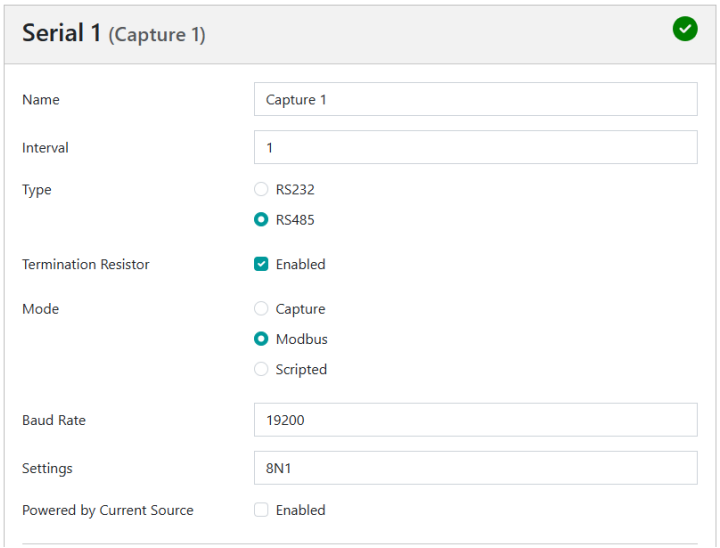

Figure 3 - Senquip Device Serial Port Settings

4. Reading Values from the Controller

The MPC-20 controller communicates using Modbus RTU. Each parameter is available as a holding register that can be read by the Senquip device via the RS485 connection. Depending on the data type, some registers are 16-bit words while others are combined to form 32-bit values.

The table below lists the registers configured for this application.

MODBUS Register	Name	Unit	Register Address	Register Size
40001	Engine Hours	Hours	0	32-bit (unsigned)
40003	Engine Speed	RPM	2	
40005	Oil Pressure	kPa	4	16-bit
40006	Engine Temperature	°C	5	16-bit
40007	Controller State		6	16-bit
40008	Shutdown State 1		7	16-bit
40009	Shutdown State 2		8	16-bit
40010	Shutdown State 3		9	16-bit
40013	Speed Setpoint	RPM	12	16-bit
40014	Ambient Temperature	°C	13	16-bit
40015	Mode (Manual = 0, Auto = 1)		14	16-bit
40239	Fuel Level	%	238	16-bit
40242	Warnings 1		241	16-bit
40243	Warnings 2		242	16-bit
40244	Warnings 3		243	16-bit

In Modbus terminology, holding registers are typically referenced using Modicon addressing, where register 40001 corresponds to an absolute address of 0. The MPC-20 follows this convention; therefore, to read register 40001, the Senquip device requests address 0.

Values returned from the MPC-20 may require conversion to engineering units. Calibration can be applied directly within the Senquip Modbus configuration for each register.

The Senquip device Modbus map is configured accordingly:

ID		Name	Slave Address	Function	Register Address	Calibration	Units	Warning	Alarm	Raw
1	X	Engine Hours	1	Read Unsigned Holding (32-bits, Little Endian register c	0	,,,Saat	Hours	None	None	
2	X	Oil Pressure	1	Read Signed Holding (16-bits)	4	0,1000,0,1000	kPa	None	None	
3	X	Engine Temperature	1	3: Read Unsigned Holding (16-bits)	5	0,10,0,1,C	°C	None	None	
4	X	Controller State	1	3: Read Unsigned Holding (16-bits)	6	None		None	None	
5	X	Shutdown State 1	1	3: Read Unsigned Holding (16-bits)	7	None		None	None	
6	X	Shutdown State 2	1	3: Read Unsigned Holding (16-bits)	8	None		None	None	
7	X	Shutdown State 3	1	3: Read Unsigned Holding (16-bits)	9	None		None	None	
8	X	Speed Setpoint	1	3: Read Unsigned Holding (16-bits)	12	,,,RPM	RPM	None	None	
9	X	Ambient Temp	1	Read Signed Holding (16-bits)	13	0,100,0,1,C	°C	None	None	
10	X	Mode	1	Read Signed Holding (16-bits)	14	None		None	None	
11	X	Engine Speed	1	3: Read Unsigned Holding (16-bits)	2	0,1000,0,1000	RPM	None	None	
13	X	Fuel Level	1	3: Read Unsigned Holding (16-bits)	238	0,1000,0,100,%	%	None	None	
14	X	Warnings 1	1	3: Read Unsigned Holding (16-bits)	241	None		None	None	
15	X	Warnings 2	1	3: Read Unsigned Holding (16-bits)	242	None		None	None	
16	X	Warnings 3	1	3: Read Unsigned Holding (16-bits)	243	None		None	None	

The engine hours are read as a single 32 bit number using little Endian format where 40001, the lower word will arrive first and 40002, the higher word will arrive last.

Decoded Modbus

Device Address	Function Code	Data				Checksum	
		Register Address		Number of Registers			
1 byte	1 byte	2 bytes		2 bytes		2 bytes	
01	03	00	00	00	02	C4	0B

Modbus Message in Hex Format: \x01\x03\x00\x00\x00\x02\xC4\x0B

Figure 4 - Engine Hours Read as Two 16 Bit Words

Instead of displaying the controller status code, shutdown status code, and warning codes, we will display the text description of each as provided in Appendix A – MPC-20 Status, Shutdown, Warning Codes.

5. Remote Control

The MPC-20 can be controlled remotely over Modbus RTU. Two writable holding registers are used for basic engine operation:

Function	Register Address	Write Values	Description
Auto/Manual Select	14	1 = Auto 0 = Manual	Sets the operating mode of the controller. The controller must be in Auto before a remote start or stop command will be accepted.

Start/Stop Command	11	1 = Start 0 = Stop	Issues a remote start or stop when in Auto mode. Command is momentary—written once per action.

In response to a user input, the Senquip device will write to these registers over RS485 using standard Modbus function 06 (write single register). The start button will have an associated confirmation message that must be acknowledged "Safety Check: Area must be clear before remote start."

6. The Device Script

First, we load the required libraries and create a global variable to store the engine mode.

```
load('senquip.js');
load('api_serial.js');

let mode = 0; // stores Auto or Manual mode status
```

Displaying the engine mode and status in a meaningful way requires that we interpret the Modbus status codes in accordance with the text descriptions described in Appendix A. Functions are created, which when called with the Modbus register value, return the text description of that code. To do this, we use an object with the Modbus code as key, and the descriptor as data.

```
function ControllerMode(i) {
  let enum_lookup = {
    '0': 'Manual',
    '1': 'Auto',
  };
  return enum_lookup[i] || 'No Read';
}

function ContollerState(i) {
  let enum_lookup = {
    '0': 'ECU Stabilize Delay',
    '1': 'Engine Stopped',
    '2': 'Controller in Standby',
    '3': 'Prestart 1 Delay',
    '4': 'Check Safe to Start',
    '5': 'Prestart 2 Delay',
    '6': 'Crank On',
    '7': 'Crank Rest',
    '8': 'False Start Check',
    '9': 'Engine Warmup Delay',
    '10': 'Line Fill 1 Delay',
    '11': 'Line Fill 2 Delay',
    '12': 'Running Loaded',
    '13': 'Running Cooldown Delay',
    '14': 'Energize to Stop Delay',
    '15': 'Engine Spindown Dela',
    '16': 'Wait to Start Delay'
  };
  return enum_lookup[i] || 'No Read';
}
```

Functions are also created, that given shutdown and warning bitmasks, return the text descriptions of the shutdown warning status described in Appendix A. The functions cycle through the provided bitmask, appending each valid message, and a carriage return to improve readability. One of these functions is shown below.

```
function ActiveShutdown1(i) {
  let enum_lookup = {
    '0': 'Overspeed SD',
    '1': 'Underspeed SD',
    '2': 'Overcrank SD',
    '3': 'Low Oil Pressure SD',
    '4': 'High Engine Temp SD',
    '5': 'Low Fuel SD',
    '6': 'Low Discharge Pressure SD',
    '7': 'High Discharge Pressure SD',
    '8': 'No Speed Signal During Crank SD',
    '9': 'Low Lube Level SD',
    '10': 'Fuel Leak SD',
    '11': 'Fuel Filter Restriction SD',
    '12': 'Low Gear Box Pressure SD',
    '13': 'High Gear Box Pressure SD',
    '14': 'Battery Charger Fail SD',
    '15': 'Remote Stop SD'
  };
  let bitmask = 0x01; // to scan through all the bits
  let error_string = '';
  for(let j = 0; j<15; j++){
    if((i & bitmask) !==0){error_string = error_string + enum_lookup[j] + '\x0A\x0D';}
    bitmask = bitmask << 1;
  }
  return error_string;
}
```

The data handler runs after every measurement cycle and call the enum functions to interpret the mode, status, shutdown, and warning codes before dispatching them to the Senquip Portal. Events are dispatched if shutdown or warning messages are present.

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

  mode = 'No Data'; // auto or manual
  let status = 'No Data'; // status message
  let shutdown = '-'; // shutdown message
  let warning = '-'; // warning message
  let battery = obj.mod3; // battery voltage

  if (typeof obj.mod13 === 'number') {mode = ControllerMode(obj.mod13);} // 40015 - mode, auto or manual
  if (typeof obj.mod6 === 'number') {status = ContollerState(obj.mod6);} // 40007 - controller state

  if (typeof obj.mod7 === 'number') {shutdown = ActiveShutdown1(obj.mod7);} // 40008 - shutdown state #1
  if (typeof obj.mod8 === 'number') {shutdown = shutdown + ActiveShutdown2(obj.mod8);} // #2
  if (typeof obj.mod9 === 'number') {shutdown = shutdown + ActiveShutdown3(obj.mod9);} // #3

  if (typeof obj.mod16 === 'number') {warning = ActiveWarningStatus1(obj.mod16);} // 40242 - warning #1
  if (typeof obj.mod17 === 'number') {warning = warning + ActiveWarningStatus2(obj.mod17);} // #2
  if (typeof obj.mod18 === 'number') {warning = warning + ActiveWarningStatus3(obj.mod18);} // #3

  // update the display with latest values
  SQ.dispatch(1, status);
  SQ.dispatch(2, shutdown);
  SQ.dispatch(3, warning);
  SQ.dispatch(4, mode);

  // setup a custom alert messages - SMS and email alerts must be configured in the alerts section
  if(shutdown !== '-'){SQ.dispatch_event(2,SQ.WARNING,shutdown);} // send SMS/email if shutdown message
  if(warning !== '-'){SQ.dispatch_event(3,SQ.WARNING,warning);} // send SMS/email if warning message

}, null);
```

To perform engine starting and stopping, we need to perform Modbus writes. A function, sendVal, is created to write an unsigned 16-bit value to a Modbus holding register. The function accepts an object containing the slave address (sadr), register address (radr), and value (val) to be written.

It constructs a Modbus write message (function code 6) by encoding the slave address as a 1-byte hexadecimal string and both the register and value as 2-byte hexadecimal strings using the SQ.encode() function. These components are concatenated to form the base Modbus message.

A CRC checksum is then computed using the SQ.crc() function and encoded with byte order reversed (using -SQ.U16) to meet Modbus MSB-first requirements. This completes the full Modbus packet, which is transmitted via SERIAL.write() on serial port 1.

```
function sendVal(sendObj){
 let s = SQ.encode(sendObj.sadr,SQ.U8); // encode dec address into hex
 let r = SQ.encode(sendObj.radr,SQ.U16); // encode dec register number into hex
 let v = SQ.encode(sendObj.val,SQ.U16); // encode dec data into hex
 let a = s+'\x06'+r+v; // 6 is the MODBUS write unsigned 16 function code
 let c = SQ.crc(a); // use the Senquip CRC function to calculate the Modbus CRC
 c = SQ.encode(c, -SQ.U16); // encode the CRC function in hex + flip byte order
 let t = a+c; // create the final Modbus write message
 SERIAL.write(1,t,t.length); // send the message to serial port 1
}
```

Adding Control Buttons

We will add trigger buttons for auto and manual control and for start and stop. The start button will have an associated confirmation message that must be acknowledged “Safety Check: Area must be clear before remote start.” The buttons are added to a group of buttons called *Engine Control*.

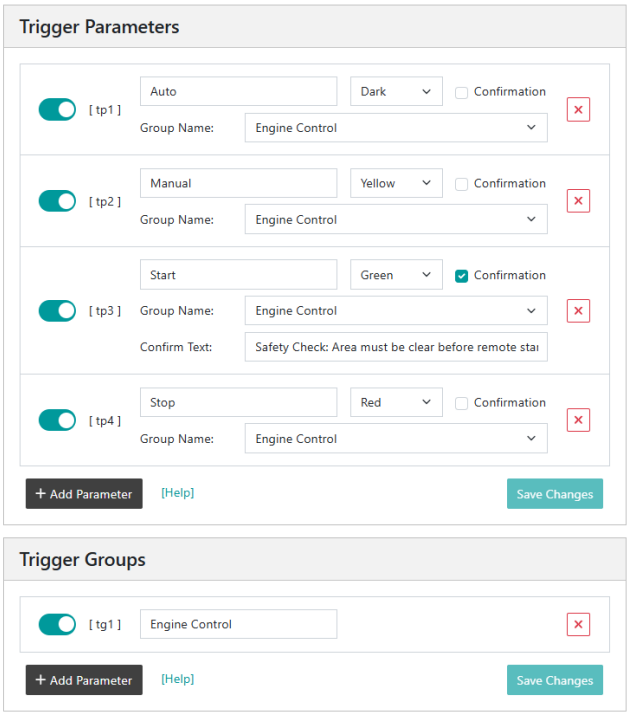

Figure 5 - Configuring Trigger Buttons

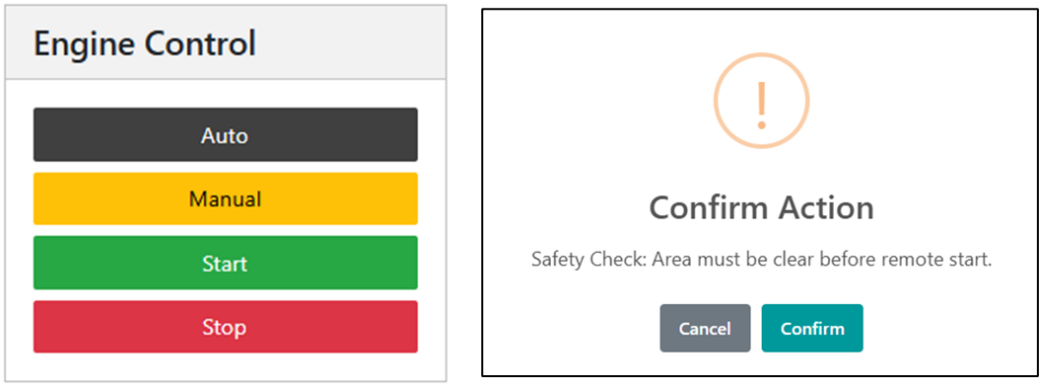

Figure 6 - Start and Stop Buttons on the Device Page and Confirm Message when Start is Pressed

When a trigger button is pressed, the next time the Senquip device contacts the Senquip Portal, the trigger will be retrieved, and the *trigger handler* will run. In the *trigger handler*, the trigger number is checked, and if active, the associated script will run on the Senquip device.

In this case, the trigger buttons use the sendVal function to write to the required Modbus registers. Alert events are dispatched when the engine starts or stops , and Info events inform the user if they try to start or stop the engine in manual mode.

```
SQ.set_trigger_handler(function(tp) {
  if (tp === 1) { sendVal({sadr:1, radr:14, val:1}); }  // Set to Auto
  if (tp === 2) { sendVal({sadr:1, radr:14, val:0}); }  // Set to Manual
  if (tp === 3) {
    if (mode === "Auto") {
      sendVal({sadr:1, radr:11, val:1}); // Start
      SQ.dispatch_event(4, SQ.ALERT, "Engine Starting");
    }
    else {
      SQ.dispatch_event(4 ,SQ.INFO, "Must be in Auto to remote start");
    }
  }
  if (tp === 4) {
    if (mode === "Auto") {
      sendVal({sadr:1, radr:11, val:0}); // Stop
      SQ.dispatch_event(4 ,SQ.ALERT, "Engine Stopping");
    }
    else {
      SQ.dispatch_event(4 ,SQ.INFO, "Must be in Auto to remote stop");
    }
  }
}, null);
```

7. Conclusions

The Senquip scripting language makes it simple to interface to a PowerCore MPC-20. Most Enovation controllers use RS485 and a similar Modbus register map, and so the application note is applicable to most other models of controller.

In addition to data received from the BBA controller, additional parameters such as location, battery voltage, pitch, roll, and vibration can be added using sensors integrated into the Senquip device. Other sensors can be added to measure oil quality, tamper and more.

Remote control is easily achieved with the implementation of a short script.

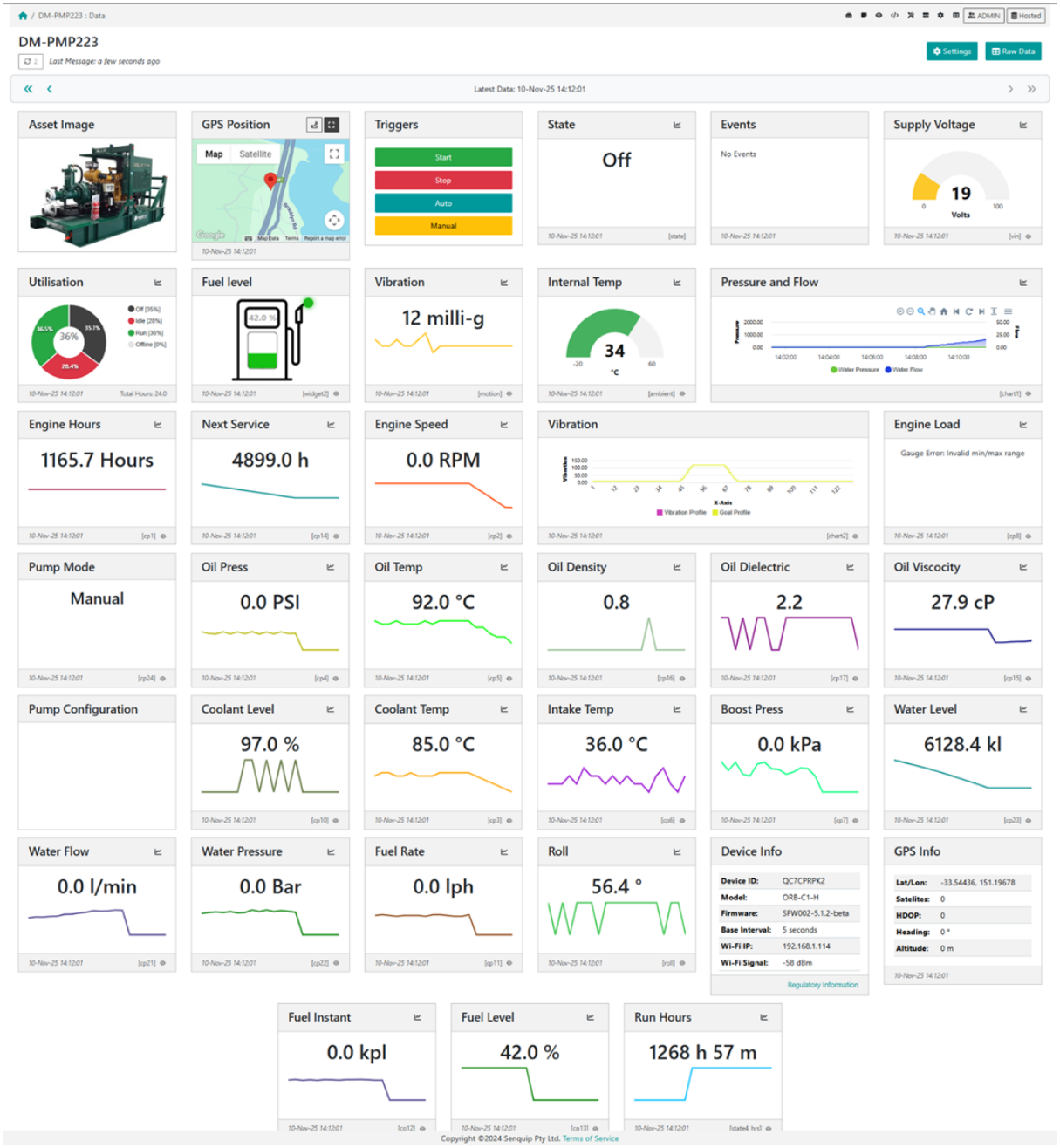

Figure 7 - Typical Pump Display on the Senquip Portal

8. Appendix A – MPC-20 Status, Shutdown, Warning Codes

MPC-20 Controller States:

Code	Controller State (4007)
0	ECU Stabilize Delay
1	Engine Stopped
2	Controller in Standby
3	Prestart 1 Delay
4	Check Safe to Start
5	Prestart 2 Delay
6	Crank On
7	Crank Rest
8	False Start Check
9	Engine Warmup Delay
10	Line Fill 1 Delay
11	Line Fill 2 Delay
12	Running Loaded
13	Running Cooldown Delay
14	Energize to Stop Delay
15	Engine Spindown Delay
16	Wait to Start Delay

MPC-20 Shutdown States:

Bit	Shutdown Register 1 (40008)	Shutdown Register 2 (40009)	Shutdown Register 3 (40010)
0	Overspeed SD	Coolant Level SD	Pivot Alignment SD
1	Underspeed SD	High Level SD	INDEX Coolant Level SD
2	Overcrank SD	Low Level SD	Speed Signal Lost While Running SD
3	Low Oil Pressure SD	High Flow SD	Reserved
4	High Engine Temp SD	Low Flow SD	Reserved
5	Low Fuel SD	Reserved	Reserved
6	Low Discharge Pressure SD	Reserved	Reserved
7	High Discharge Pressure SD	Water in Fuel SD	Reserved
8	No Speed Signal During Crank SD	Low Suction SD	Reserved
9	Low Lube Level SD	High Suction SD	Reserved
10	Fuel Leak SD	Reserved	Reserved
11	Fuel Filter Restriction SD	High Engine Oil Temp SD	Reserved
12	Low Gear Box Pressure SD	Low Gear Box Pressure SD	Reserved
13	High Gear Box Pressure SD	High Gear Box Pressure SD	Reserved
14	Battery Charger Fail SD	Reserved	Reserved
15	Remote Stop SD	Red Lamp Status	Reserved

MPC-20 Warning States:

Bit	Warning Register 1 (40242)	Warning Register 2 (40243)	Warning Register 3 (40244)
0	Low Fuel Warn	High Flow Warn	Pivot Alignment Warn
1	Fuel Leak Warn	Low Flow Warn	Reserved
2	Fuel Filter Restriction Warn	High Pump Oil Temp Warn	Reserved

3	Low Lube Level Warn	High Pump Housing Temp Warn	Reserved
4	Coolant Level Warn	Low Gear Box Pressure Warn	Reserved
5	Water in Fuel Warn	High Gear Box Pressure Warn	Reserved
6	No Flow Warn	Air Damper Closed Warn	Reserved
7	High Engine Oil Temp Warn	Air Filter Restriction Warn	Reserved
8	Low Oil Pressure Warn	Oil Filter Restriction Warn	Reserved
9	High Engine Temp Warn	Low Engine Temp Warn	Reserved
10	High Discharge Pressure Warn	High Engine Oil Pressure Warn	Reserved
11	Low Discharge Pressure Warn	Battery Charger Fail Warn	Reserved
12	High Suction Warn	Run To Destruct Warn	Reserved
13	Low Suction Warn	Battery High Warn	Reserved
14	High Level Warn	Battery Low Warn	Reserved
15	Low Level Warn	Amber Lamp Status	Reserved

9. Appendix B – Full Application Script

```
load('senquip.js');
load('api_serial.js');

let mode = ""; // stores Auto or Manual mode

function ControllerMode(i) {
  let enum_lookup = {
    '0': 'Manual',
    '1': 'Auto',
  };
  return enum_lookup[i] || 'No Read';
}

function ContollerState(i) {
  let enum_lookup = {
    '0': 'ECU Stabilize Delay',
    '1': 'Engine Stopped',
    '2': 'Controller in Standby',
    '3': 'Prestart 1 Delay',
    '4': 'Check Safe to Start',
    '5': 'Prestart 2 Delay',
    '6': 'Crank On',
    '7': 'Crank Rest',
    '8': 'False Start Check',
    '9': 'Engine Warmup Delay',
    '10': 'Line Fill 1 Delay',
    '11': 'Line Fill 2 Delay',
    '12': 'Running Loaded',
    '13': 'Running Cooldown Delay',
    '14': 'Energize to Stop Delay',
    '15': 'Engine Spindown Delay',
    '16': 'Wait to Start Delay'
  };
  return enum_lookup[i] || 'No Read';
}

function ActiveShutdown1(i) {
  let enum_lookup = {
    '0': 'Overspeed SD',
    '1': 'Underspeed SD',
    '2': 'Overcrank SD',
    '3': 'Low Oil Pressure SD',
```

```
    '4': 'High Engine Temp SD',
    '5': 'Low Fuel SD',
    '6': 'Low Discharge Pressure SD',
    '7': 'High Discharge Pressure SD',
    '8': 'No Speed Signal During Crank SD',
    '9': 'Low Lube Level SD',
    '10': 'Fuel Leak SD',
    '11': 'Fuel Filter Restriction SD',
    '12': 'Low Gear Box Pressure SD',
    '13': 'High Gear Box Pressure SD',
    '14': 'Battery Charger Fail SD',
    '15': 'Remote Stop SD'
  };
  let bitmask = 0x01; // to scan through all the bits
  let error_string = '';
  for(let j = 0; j<15; j++){
    if((i & bitmask) !==0){error_string = error_string + enum_lookup[j] + '\x0A\x0D';}
    bitmask = bitmask << 1;
  }
  return error_string;
}

function ActiveShutdown2(i) {
  let enum_lookup = {
    '0': 'Coolant Level SD',
    '1': 'High Level SD',
    '2': 'Low Level SD',
    '3': 'High Flow SD',
    '4': 'Low Flow SD',
    '5': 'Reserved Reserved',
    '6': 'Reserved Reserved',
    '7': 'Water in Fuel SD',
    '8': 'Low Suction SD',
    '9': 'High Suction SD',
    '10': 'Reserved Reserved',
    '11': 'High Engine Oil Temp SD',
    '12': 'Low Gear Box Pressure SD',
    '13': 'High Gear Box Pressure SD',
    '14': 'Reserved Reserved',
    '15': 'Red Lamp Status'
  };
  let bitmask = 0x01; // to scan through all the bits
  let error_string = '';
  for(let j = 0; j<15; j++){
    if((i & bitmask) !==0){error_string = error_string + enum_lookup[j] + '\x0A\x0D';}
    bitmask = bitmask << 1;
  }
  return error_string;
}

function ActiveShutdown3(i) {
  let enum_lookup = {
    '0': 'Pivot Alignment SD',
    '1': 'INDEX Coolant Level SD',
    '2': 'Speed Signal Lost While Running SD',
    '3': 'Reserved',
    '4': 'Reserved',
    '5': 'Reserved',
    '6': 'Reserved',
    '7': 'Reserved',
    '8': 'Reserved',
    '9': 'Reserved',
    '10': 'Reserved',
    '11': 'Reserved',
    '12': 'Reserved',
    '13': 'Reserved',
```

```
    '14': 'Reserved',
    '15': 'Reserved'
  };
  let bitmask = 0x01; // to scan through all the bits
  let error_string = '';
  for(let j = 0; j<15; j++){
    if((i & bitmask) !==0){error_string = error_string + enum_lookup[j] + '\x0A\x0D';}
    bitmask = bitmask << 1;
  }
  return error_string;
}

function ActiveWarningStatus1(i) {
  let enum_lookup = {
    '0': 'Low Fuel Warn',
    '1': 'Fuel Leak Warn',
    '2': 'Fuel Filter Restriction Warn',
    '3': 'Low Lube Level Warn',
    '4': 'Coolant Level Warn',
    '5': 'Water in Fuel Warn',
    '6': 'No Flow Warn',
    '7': 'High Engine Oil Temp Warn',
    '8': 'Low Oil Pressure Warn',
    '9': 'High Engine Temp Warn',
    '10': 'High Discharge Pressure Warn',
    '11': 'Low Discharge Pressure Warn',
    '12': 'High Suction Warn',
    '13': 'Low Suction Warn',
    '14': 'High Level Warn',
    '15': 'Low Level Warn'
  };
  let bitmask = 0x01; // to scan through all the bits
  let error_string = '';
  for(let j = 0; j<15; j++){
    if((i & bitmask) !==0){error_string = error_string + enum_lookup[j] + '\x0A\x0D';}
    bitmask = bitmask << 1;
  }
  return error_string;
}

function ActiveWarningStatus2(i) {
  let enum_lookup = {
    '0': 'High Flow Warn',
    '1': 'Low Flow Warn',
    '2': 'High Pump Oil Temp Warn',
    '3': 'High Pump Housing Temp Warn',
    '4': 'Low Gear Box Pressure Warn',
    '5': 'High Gear Box Pressure Warn',
    '6': 'Air Damper Closed Warn',
    '7': 'Air Filter Restriction Warn',
    '8': 'Oil Filter Restriction Warn',
    '9': 'Low Engine Temp Warn',
    '10': 'High Engine Oil Pressure Warn',
    '11': 'Battery Charger Fail Warn',
    '12': 'Run To Destruct Warn',
    '13': 'Battery High Warn',
    '14': 'Battery Low Warn',
    '15': 'Amber Lamp Status'
  };
  let bitmask = 0x01; // to scan through all the bits
  let error_string = '';
  for(let j = 0; j<15; j++){
    if((i & bitmask) !==0){error_string = error_string + enum_lookup[j] + '\x0A\x0D';}
    bitmask = bitmask << 1;
  }
  return error_string;
```

```
}

function ActiveWarningStatus3(i) {
  let enum_lookup = {
    '0': 'Pivot Alignment Warn',
    '1': 'Reserved',
    '2': 'Reserved',
    '3': 'Reserved',
    '4': 'Reserved',
    '5': 'Reserved',
    '6': 'Reserved',
    '7': 'Reserved',
    '8': 'Reserved',
    '9': 'Reserved',
    '10': 'Reserved',
    '11': 'Reserved',
    '12': 'Reserved',
    '13': 'Reserved',
    '14': 'Reserved'
  };
  let bitmask = 0x01; // to scan through all the bits
  let error_string = '';
  for(let j = 0; j<15; j++){
    if((i & bitmask) !==0){error_string = error_string + enum_lookup[j] + '\x0A\x0D';}
    bitmask = bitmask << 1;
  }
  return error_string;
}


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

  mode = 'No Data'; // auto or manual
  let status = 'No Data'; // status message
  let shutdown = '-'; // shutdown message
  let warning = '-'; // warning message
  let battery = obj.mod3; // battery voltage

  if (typeof obj.mod13 === 'number') {mode = ControllerMode(obj.mod13);} // register 40015 - controller mode,
auto or manual
  if (typeof obj.mod6 === 'number') {status = ContollerState(obj.mod6);} // register 40007 - controller state
  if (typeof obj.mod7 === 'number') {shutdown = ActiveShutdown1(obj.mod7);} // register 40008 - active
shutdown state
  if (typeof obj.mod8 === 'number') {shutdown = shutdown + ActiveShutdown2(obj.mod8);} // register 40009 -
active shutdown state
  if (typeof obj.mod9 === 'number') {shutdown = shutdown + ActiveShutdown3(obj.mod9);} // register 40100 -
active shutdown state
  if (typeof obj.mod16 === 'number') {warning = ActiveWarningStatus1(obj.mod16);} // register 40242 - active
warning state
  if (typeof obj.mod17 === 'number') {warning = warning + ActiveWarningStatus2(obj.mod17);} // register 40243
- active warning state
  if (typeof obj.mod18 === 'number') {warning = warning + ActiveWarningStatus3(obj.mod18);} // register 40244
- active warning state

  // update the display with engine data
  SQ.dispatch(1, status);
  SQ.dispatch(2, shutdown);
  SQ.dispatch(3, warning);
  SQ.dispatch(4, mode);

  // here we setup the customer alert messages - SMS and email alerts must be configured in the alerts
section
  if(shutdown !== '-'){SQ.dispatch_event(2,SQ.WARNING,shutdown);} // send SMS/email if shutdown message
present
  if(warning !== '-'){SQ.dispatch_event(3,SQ.WARNING,warning);} // send SMS/email if warning message present
```

```
  if(battery < 11 || battery > 15){SQ.dispatch_event(3,SQ.WARNING,"Battery voltage error: " +
SQ.to_fixed(battery, 1));} // send SMS/email if battery low or high

}, null);

function sendVal(sendObj){
let s = SQ.encode(sendObj.sadr,SQ.U8); // encode dec address into hex
 let r = SQ.encode(sendObj.radr,SQ.U16); // encode dec register number into hex
 let v = SQ.encode(sendObj.val,SQ.U16); // encode dec data into hex
let a = s+'\x06'+r+v; // 6 is the MODBUS write unsigned 16 function code
let c = SQ.crc(a); // use the Senquip CRC function to calculate the Modbus CRC
c = SQ.encode(c, -SQ.U16); // encode the CRC function in hex + flip byte order
 let t = a+c; // create the final Modbus write message
 SERIAL.write(1,t,t.length); // send the message to serial port 1
}

SQ.set_trigger_handler(function(tp) {
  if (tp === 1) { sendVal({sadr:1, radr:14, val:1}); }  // Set to Auto
  if (tp === 2) { sendVal({sadr:1, radr:14, val:0}); }  // Set to Manual
  if (tp === 3) {
    if (mode === "Auto") {
      sendVal({sadr:1, radr:11, val:1}); // Start
      SQ.dispatch_event(4, SQ.ALERT, "Engine Starting");
    }
    else {
      SQ.dispatch_event(4 ,SQ.INFO, "Must be in Auto to remote start");
    }
  }
  if (tp === 4) {
    if (mode === "Auto") {
      sendVal({sadr:1, radr:11, val:0}); // Stop
      SQ.dispatch_event(4 ,SQ.ALERT, "Engine Stopping");
    }
    else {
      SQ.dispatch_event(4 ,SQ.INFO, "Must be in Auto to remote stop");
    }
  }
}, null);
```