

CAN INTEGRATION WITH A BBA PUMP CONTROLLER

1. Overview

[BBA Pumps](#) is a leading global manufacturer of mobile pump systems designed for construction, mining, and flood control applications.

By integrating a Senquip device with the BBA control panel, users gain real-time visibility of pump operation, enabling proactive maintenance, faster response to faults, and improved operational efficiency.



Figure 1 - BBA Pump in Action

2. Introduction

BBA Pumps equips many of their mobile pump systems with LC40 control panels, providing advanced monitoring, control, and protection features via a CAN (Controller Area Network) interface. The LC40 panel broadcasts operational data such as engine status, pump parameters, fault codes, and system alarms across the CAN bus, while also accepting remote control commands such as start and stop.

This application note describes how to interface a Senquip telemetry device with a BBA Pump fitted with an LC40 control panel. By connecting to the CAN network, the Senquip device enables both remote monitoring of critical pump parameters and remote start/stop control of the pump system. This integration allows operators to manage pumps in real time, improving asset utilization, enabling proactive maintenance, and reducing the need for site visits, particularly in remote or unmanned locations.

Document Number
APN0041Revision
APrepared By
JGApproved By
NBTitle
CAN Integration with BBA Pump ControllerPage
2 of 21

Figure 2 – BBA LC45 Control Panel

The following sections provide guidance on wiring, configuration, and scripts required for the Senquip device.

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.*

3. Wiring the Senquip Device to the BBA LC40 Controller

In this application note, we will use CAN port 1 on a QUAD-C2 wired to the external device connector, J20, on the BBA LC40 controller.

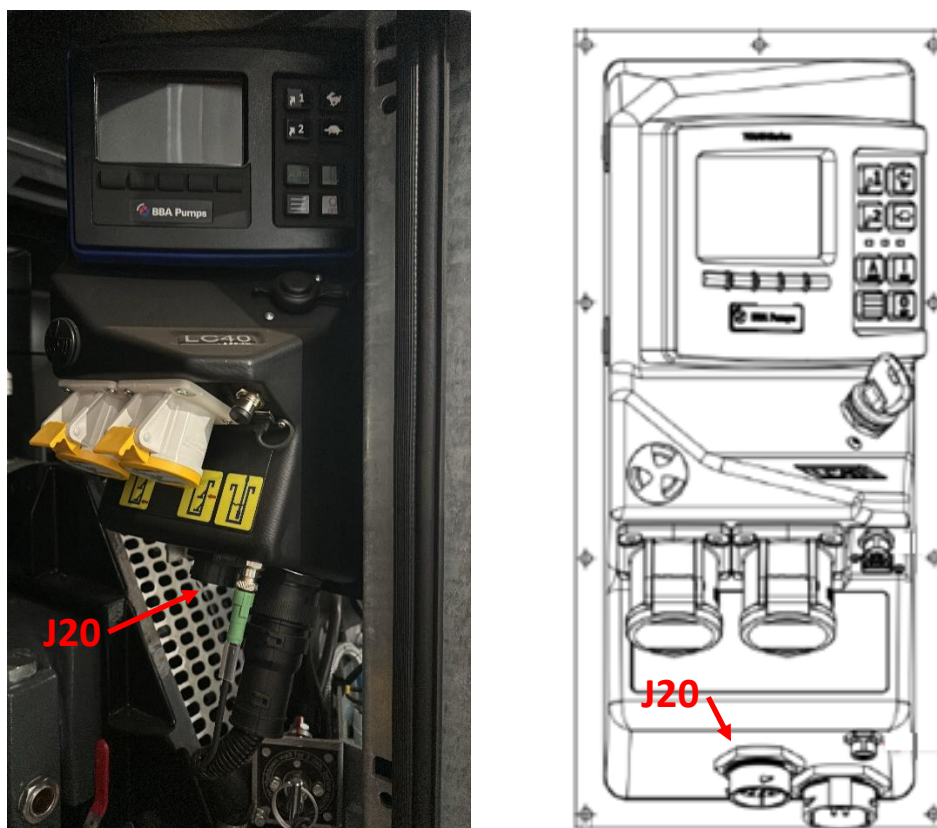


Figure 3 - CAN Connector on the BBA Controller

The matching connector for the LC40 control panel external device connector is a Deutsch HDP24-18-14-S-N. A pre-made cable is available from BBA. The following connections are required:

Connection	Senquip QUAD	BBA LC40 J20
CAN H	Pin 11, CAN1 H	Pin G, CAN HIGH
CAN L	Pin 12, CAN1 L	Pin H, CAN LOW
GND	Pin 8, GND	Pin F, GROUND
PWR	Pin 1, PWR	Pin A, BATTERY + FUSED
GND	Pin 2, GND	Pin F, Ground

Since the Senquip device and Deep Sea controller share a common power supply ground, the ground connection between pins 8 and F is not required. If a screened wire is available, it should be connected to either the Senquip or Deep Sea controller ground but not both. Connecting to both can create a ground loop which will be susceptible to magnetic fields.

Document Number
APN0041

Revision
A

Title
CAN Integration with BBA Pump Controller

Prepared By
JG

Approved By
NB

Page
4 of 21



Figure 4 - LC40 External Device Connector Plug

If the Senquip device and BBA controller are at the end of the line on the CAN network, then a 120ohm termination resistor must be placed at each end of the line.

QUAD-C2

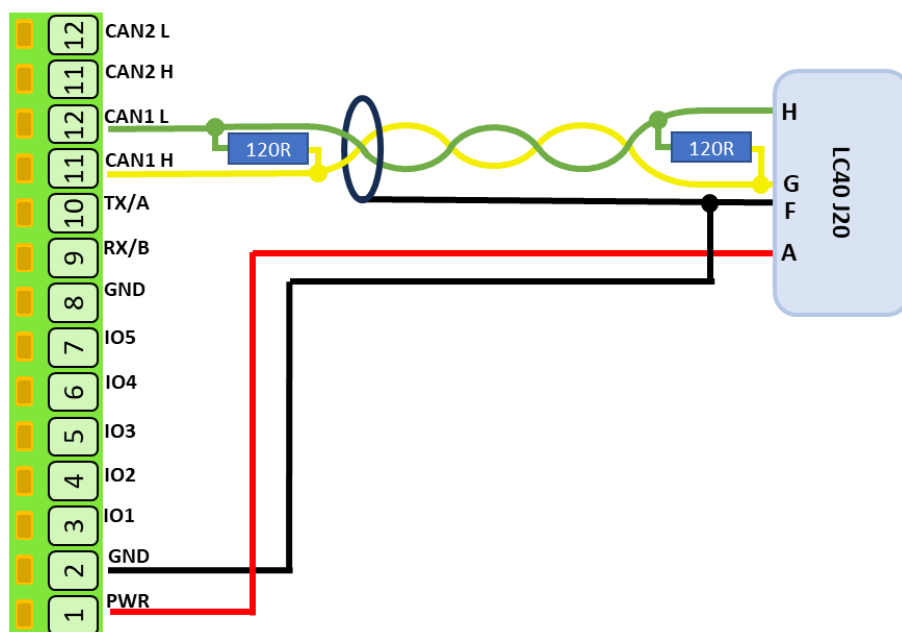


Figure 5 - Senquip QUAD to LC40 Wiring

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
5 of 21

4. Senquip Device Configuration

We get the communications specification for the BBA LC40 controller from the [CATTRON CANplus Panel API Manual](#). A newer 2023 version of this manual is available from BBA.

The CAN Bus Specifications are found to be as follows. The Senquip device CAN port is configured accordingly.

Parameter	Value
Bit rate	250kbit/s
Protocol	J1939

The *CAN Capture Time* is set to be the same as the base interval, 5 seconds, so that all CAN messages are captured. *TX Enable* is ticked to allow messages to be transmitted onto the CAN network. For debug purposes, *Send Raw Data* is also ticked. This can be turned off once the application is completed.

CAN 1

Name

CAN 1

Interval

1

Nominal Baud Rate

250

kbit/s

Capture Time

5

Seconds

TX Enable

☒ Enabled

ID Capture List

ID Capture List

Send Raw Data

☒ Enabled

Figure 6 - Senquip QUAD CAN Port Settings

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
6 of 21

5. LC40 CAN Messaging

The LC40 control panel communicates over a J1939-based CAN bus using a mix of standard and proprietary PGNs. While some broadcast messages use standard J1939 PGNs, most command and data exchanges—such as reading sensor values or issuing control commands—are handled using the proprietary PGN 65525 (Database Command). This PGN provides a mechanism for accessing and modifying database parameters within the controller by specifying a parameter address, command type, priority, and optional time to live.

Parameter Group Number	65525 (0xFF5)
Transmission Repetition Rate	On Request
Data Length	8

The 8 data bytes are defined as follows:

Start Position	Length	Parameter Name
1.1	5 bits	Database Command
1.6	2 bits	Database Priority
2.1	1 byte	Time to Live
3.1	2 bytes	Database Parameter Address
5.1	4 bytes	Database Parameter Value

Valid Database Commands are below. We will use command number 1 to read engine coolant temperature, oil temperature, engine speed, engine hours, controller status, fuel rate, fuel level, and torque. Other parameters are available if required.

We will use command 2 to change the controller state between Auto and Manual, and to start and stop the pump.

Command Number	Description
1	Read a database parameter
2	Write a database parameter
3	Response to a read command
4	Response to a write command
5	Subscription to publish

Priority is defined as below. The API manual is not clear on priorities for read requests and so a priority of Normal will be chosen. When sending remote start or stop commands priority Override will be used. While not explicitly documented, this priority likely ensures the command is acted on regardless of the panel's operating mode. We will only allow remote start from Manual mode.

Priority	Decode Value
Low	0
Normal	1
External	2
Override	3

The TTL (Time to Live) determines how long the written value remains valid in volatile memory for runtime parameters. When writing to configuration registers, (CAN address less than 1000), it is recommended to use TTL

Document Number APN0041	Revision A	Prepared By JG	Approved By NB
Title CAN Integration with BBA Pump Controller			Page 7 of 21

Infinite in which case the change will be saved across power cycles. For reads, TTL is not defined and will be set to Infinite.

States	Decode Value
Invalid	0
TTL Duration	0 - 254
Infinite TTL	255

The full list of parameters that can be read can be found in the CANplus API manual. We will read the following.

Parameter	CAN Address	Offset	Scaling	Unit
Engine Coolant Temperature	1003	-40	1°C / bit	°C
Engine Oil Temperature	1004	0	1 kPa / bit	kPa
Engine Speed	1000	0	1 RPM / bit	RPM
Engine Hours	1013	0	0.05 hours / bit	hours
Engine Fuel Rate	1248	0	0.05 l/h / bit	l/h
Fuel Level	1091	0	.01 % / bit	%
Panel Operating Mode	1036			
Engine Running	1041			

Panel operating modes are defined as below. We will enable remote Auto Start / Manual control and remote Start / Stop. Although not clearly documented, we are told that remote start can only be used from Auto Start mode.

Mode	Name	Description
0	Manual Start	Panel in manual mode. Engine will start/stop based on configured triggers.
1	Auto Start	Panel in automatic mode. No auto-start behavior.
2	Manual Start Sleep	System in low-power state after manual operation; monitoring and output functionality may be reduced.
3	Auto Start Sleep	Low-power mode but Auto Start conditions are still monitored.
4	Shutting Down	Panel is actively executing a shutdown sequence (e.g., after stop command or fault).

By combining the Database Command, Database Priority, Time to Live, Database Parameter Address, and Database Parameter Value, we can assemble the full CAN message for each of the parameters we want to read. Note how the address and data fields are reversed as J1939 is little endian.

	Byte 1				Byte 2	Byte 3-4	Byte 5-8	Full Message
Parameter	Spare bit 8	Priority (bits 6-7)	Command (bits 1-5)	Hex (1 byte)	TTL (1 byte)	Address (2 bytes)	Value (4 bytes)	Data Bytes (8 bytes)
Engine Coolant Temperature	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0xEB03	0x0000 - NA	21 FF EB 03 00 00 00 00
Engine Oil Temperature	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0xEC03	0x0000 - NA	21 FF EC 03 00 00 00 00
Engine Speed	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0xE803	0x0000 - NA	21 FF E8 03 00 00 00 00
Engine Hours	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0xF503	0x0000 - NA	21 FF F5 03 00 00 00 00

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
8 of 21

Engine Fuel Rate	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0xE004	0x0000 - NA	21 FF E0 04 00 00 00 00
Fuel Level	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0x4304	0x0000 - NA	21 FF 43 04 00 00 00 00
Panel Operating Mode	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0x0C04	0x0000 - NA	21 FF 0C 04 00 00 00 00
Engine Running	0	01 - Normal	00001 - Read	0x21	0xFF - Infinite	0x1104	0x0000 - NA	21 FF 11 04 00 00 00 00

Likewise, we can define the registers that we want to write.

Warning: While the CATTRON CANplus API manual lists the "Panel Operating Mode" at database address **1036**, this address may not function on BBA pump controllers. In field testing, and as confirmed by Cattron technical support, the correct operational address is **1308 (0x051C)**.

	Byte 1				Byte 2	Byte 3-4	Byte 5-8	Full Message
Parameter	Spare bit 8	Priority (bits 7-6)	Command (bits 5-1)	Hex (1 byte)	TTL (1 byte)	Address (2 bytes)	Value (4 bytes)	Data Bytes (8 bytes)
Panel Operating Mode – set to Manual	0	11 - Override	00010 - Write	0x62	0xFF - Infinite	0x1C05	0x0000 – Manual	62 FF 1C 05 00 00 00 00
Panel Operating Mode – set to Auto Start	0	11 - Override	00010 - Write	0x62	0xFF - Infinite	0x1C05	0x1000 – Auto Start	62 FF 1C 05 01 00 00 00
Start/Stop Override – Start Engine	0	11 - Override	00010 – Write	0x62	0xFF - Infinite	0x0404	0x1000 - Start	62 FF 04 04 01 00 00 00
Start/Stop Override – Stop Engine	0	11 - Override	00010 - Write	0x62	0xFF - Infinite	0x0404	0x0000 - Stop	62 FF 04 04 00 00 00 00

5.1. Source Address

When transmitting onto the CAN bus, we need a source address. In SAE J1939-71 (Vehicle Application Layer), Source Address 248 is designated as "Miscellaneous" or "Other" in. It is typically used for devices that do not fit into the predefined functional categories (e.g., engine, transmission, or data logger) or for proprietary or auxiliary devices. We will use Source Address 248 (0xF8).

5.2. Priority

The API documentation does not specify a required priority for reads. We will use a low priority of 6 for read operations. Using the [CAN ID Calculator](#) on the Senquip [documents](#) page, we can calculate the CAN IDs for a read.

CAN ID Calculator Rev 1.1			
SENQUIP			
Message ID to PGN			
INPUT:	Message ID (HEX)	HEX	DEC
		18FFF5F8	419427832
OUTPUT:		HEX	DEC
	PGN (DEC)	FFF5	65525
	Source Address (DEC)	F8	248
	Priority (DEC)	6	6
PGN to Message ID			
INPUT:		DEC	HEX
	PGN (DEC)	65525	FFF5
	Source Address (DEC)	248	F8
	Priority (DEC)	6	18
OUTPUT:	Message ID (HEX)	DEC	HEX
		419427832	18FFF5F8

Figure 7 - Senquip CAN ID Calculator

Although a read priority is not specifically stated in the API, a high priority is consistent with remote commands being treated as override commands of high priority. For a write, we will use priority 3.

Operation	CAN ID
Read	0x18FFF5F8
Write	0x0CFFF5F8

6. Implementing Remote Monitoring and Control in a Script

We will now write a script to enable remote monitoring and starting and stopping of a BBA pump using the LC40 controller. The full script is available in Appendix B. It is assumed that the reader has scripting access, and that they have a fair knowledge of the Senquip scripting language. Further details on the Senquip scripting language can be found in the [Senquip Scripting Guide](#).

Warning: Because of new features used in this script, please update your firmware version to SFW003-6.0.0 or later.

6.1. Initialisation

First, we load the required libraries. A structure is created that contains all the parameters that are to be read. If extra parameters are required, they need to be added to this structure.

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

let params = [
  {'addr' : 1003, 'value' : NaN }, // Engine Coolant Temperature
  {'addr' : 1004, 'value' : NaN }, // Engine Oil Temperature
  {'addr' : 1000, 'value' : NaN }, // Engine Speed
  {'addr' : 1013, 'value' : NaN }, // Engine Hours
  {'addr' : 1248, 'value' : NaN }, // Engine Fuel Rate
  {'addr' : 1091, 'value' : NaN }, // Fuel Level
  {'addr' : 1036, 'value' : NaN }, // Panel Operating Mode
  {'addr' : 1041, 'value' : NaN }]; // Engine Running
```

6.2. Reading the CAN Parameters

The 8 CAN parameters will be requested in the order in which they appear in the parameters structure. Ideally all 8 parameters should be requested and received back during the measurement cycle and before the *data handler* is called. If they are not available before the data handler is called, they will not be able to be interpreted and dispatched for display on the Senquip Portal. A small delay between reads is recommended in the API manual. To achieve this, a new *pre handler*, available in SFW006 will be used.

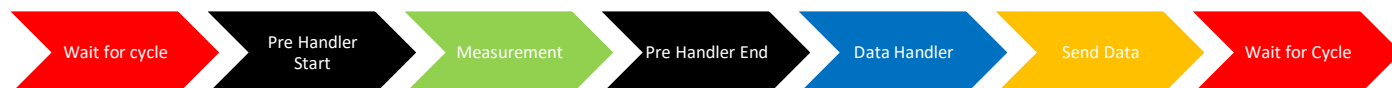


Figure 8 – Pre Handler Order of Operations

The API manual suggests a delay between CAN read requests to prevent overwhelming the CAN Bus. Although this is unlikely, a 100msec delay is inserted between read requests.

The *pre handler* is initialised with the optional flag *SQ.PRE_DURING*. This flag directs the pre handler to run during the measurement cycle. This allows for future implementations of the handler that run, for instance, before the measurement cycle.

In the *pre handler*, a call is made to function *read_can_timed()* to request the first CAN parameter. The function initialises a timer that will timeout in 100msec and call function *read_param()*, which prepares a parameter request message and transmits it. If the parameter being requested is not the last, then *read_can_timed()* is called again to schedule the next parameter request. When all the parameters have been requested, *SQ.pre_handler_complete()* is called, indicating that the pre handler is complete, and that the data handler can start when the measurement cycle completes.

```

/* Read a single CAN parameter */
function read_param(index) {
    let message = "\x21\xff" + SQ.encode(params[index].addr, -SQ.U16) + "\x00\x00\x00\x00"; // assemble the
    request message
    CAN.tx(1, 0x18FFF5F8, message, 8, CAN.EXT); // transmit the request

    index++; // Increment the index
    if (index < params.length) {
        read_can_timed(index); // Start the next timed read
    }
    else{
        SQ.pre_handler_complete(); // allow the data handler to run
    }
}

/* Cycles through all the requests with a 100msec interval */
function read_can_timed(index) {
    Timer.set(100, 0, function(userdata) { // Pass index to the timer function as the userdata parameter
        read_param(userdata); // request a single parameter
    }, index);
}

/* Start the CAN requests */
let type = SQ.PRE_DURING;
SQ.set_pre_handler(type, function() {
    read_can_timed(0); // request the first CAN parameter
}, null);

```

In the main *data handler*, the JSON format measurements are parsed into an object. If data has been measured on CAN1, then all the received messages are cycled through, in each case looking for PGN 65525. If the PGN is found, the command code is checked for the value 3, in which case it is a response to a read request.

When a read response is found, the address is checked to find what parameter has been received. The data bytes are extracted from the CAN message and are scaled and offset to create the required parameter. The use of the *SQ.parse()* function is well described in the Scripting Guide and other application notes and will not be covered further here. Only a few messages are shown below, the full script is available in Appendix B.

For the Panel Operating Mode and Engine Running parameters, simple logic is applied to the value to allow a status text message to be dispatched to the Senquip Portal.

To simplify integration and streamline data handling, Senquip has introduced a new feature: Standard Keys. When dispatching parameters to the Senquip Portal, users previously had to define custom parameters for commonly used engine metrics—such as engine speed, oil temperature, and coolant pressure. This process could be time-consuming and inconsistent across devices. With the introduction of Standard Keys, these common parameters can now be referenced using predefined, recognised names. This enhances data uniformity, reduces setup time, and allows for easier display and analysis on the Senquip Portal.

The following standard keys are available:

Key	Unit	Key	Unit
eng_hrs	Hours	eng_load	Percent
idle_hrs	Hours	eng_temp	Degrees C
run_hrs	Hours	coolant_temp	Degrees C
total_fuel	Litres	coolant_level	Percent

Document Number APN0041	Revision A	Prepared By JG	Approved By NB
Title CAN Integration with BBA Pump Controller			Page 12 of 21

idle_fuel	Litres	coolant_pressure	kPa
run_fuel	Litres	oil_temp	Degrees C
fuel_level	Percent	oil_pressure	kPa
fuel_rate	Litres/hour	oil_level	Percent
trip_fuel	Litres	trans_temp	Degrees C
fuel_economy	km/litre	trans_pressure	kPa
distance_km	Kilometres	trans_level	Percent
eng_speed	RPM		

```

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++) {
            if (pgn(obj.can1[i].id) === 65525) { // Database Command PGN
                let command = SQ.parse(obj.can1[i].data, 0, 2, -16); // retrieve the command byte
                if ((command & 0x1F) === 3) { // response to a read command
                    let addr = SQ.parse(obj.can1[i].data, 4, 4, -16); // byte 2-3 contain address of the parameter
                    if (addr === 1003) {
                        let EngineCoolantTemp = SQ.parse(obj.can1[i].data, 8, 8, -16) - 40;
                        SQ.dispatch("coolant_temp", EngineCoolantTemp);
                    }
                    if (addr === 1013) {
                        let EngineHours = SQ.parse(obj.can1[i].data, 8, 8, -16) * 0.05;
                        SQ.dispatch('eng_hrs', EngineHours);
                    }
                    if (addr === 1036) {
                        let PanelOperatingMode = SQ.parse(obj.can1[i].data, 8, 8, -16);
                        if (PanelOperatingMode === 0) {
                            SQ.dispatch(1, "Manual Start");
                        }
                        else if (PanelOperatingMode === 1) {
                            SQ.dispatch(1, "Auto Start");
                        }
                        else if (PanelOperatingMode === 4) {
                            SQ.dispatch(1, "Shutting Down");
                        }
                    }
                    if (addr === 1041) {
                        let EngineRunning = SQ.parse(obj.can1[i].data, 8, 8, -16);
                        if (EngineRunning === 0) {
                            SQ.dispatch_event(2, "Not Running");
                        }
                        else if (EngineRunning === 1) {
                            SQ.dispatch_event(2, "Running");
                        }
                    }
                }
            }
        }
    }
}, null);

```

6.3. Remote Control

We will use trigger buttons on the Senquip Portal to implement functions to put the generator in Manual, and Auto Start mode, and to Start and Stop the pump. We create 4 *Trigger Parameters* on the device scripting page. We have named the triggers Manual, Auto, Start, and Stop, and have made them yellow, blue, green, and red.

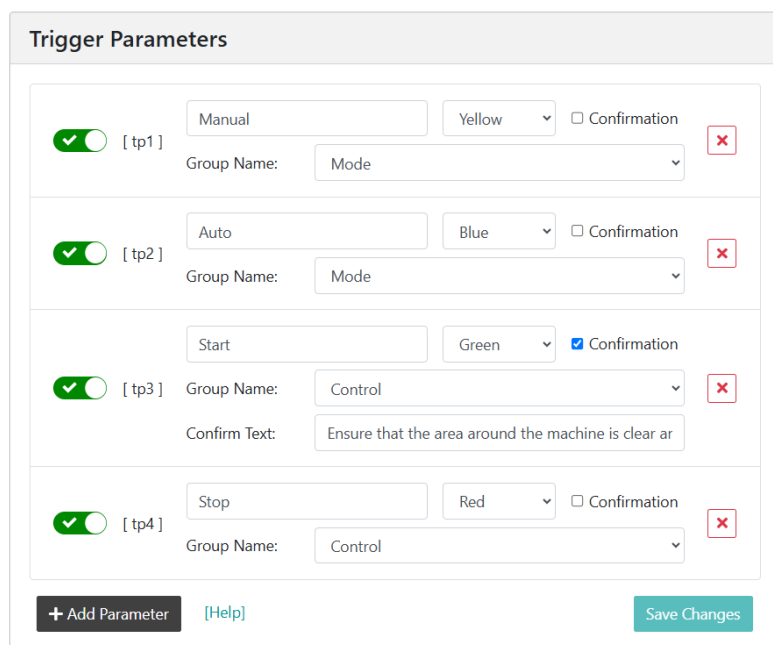


Figure 9 – Creating the Trigger Parameters

Note the confirmation message that will appear when a user activates that start button.

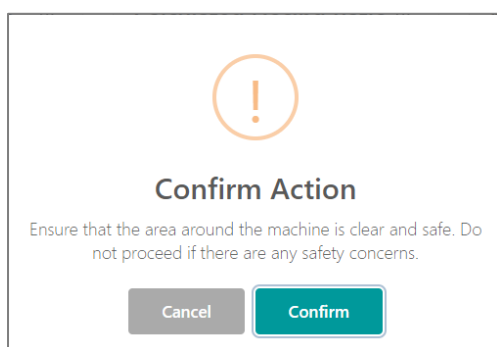


Figure 10 - Example Trigger Button Confirmation Message

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 appropriate parameter write command is assembled and sent. For start and stop commands, an alert message is dispatched to the Senquip Portal.

Simple logic is applied to the start command to only allow start of the pump from Auto Start mode.

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
14 of 21



Figure 11 - Order of Operations

```

SQ.set_trigger_handler(function(tp){
  if (tp === 1){ //Manual
    let write = "\x62\xff\x1c\x05\x00\x00\x00\x00";
    CAN.tx(1, 0x0CFFF5F8, write, 8, CAN.EXT); // change to Manual mode
  }
  if (tp === 2){ //Auto Start
    let write = "\x62\xff\x1c\x05\x01\x00\x00\x00";
    CAN.tx(1, 0x0CFFF5F8, write, 8, CAN.EXT); // change to Auto Start mode
  }
  if (tp === 3){ // Start
    if(params[6].value === 1){
      SQ.dispatch_event(1,SQ.INFO,"Engine Starting");
      let write = "\x62\xff\x04\x04\x01\x00\x00\x00";
      CAN.tx(1, 0x0CFFF5F8, write, 8, CAN.EXT);
    }
    else {
      SQ.dispatch_event(1,SQ.INFO,"Must be in Auto Start Mode to Start Engine");
    }
  }
  if (tp === 4){ // Stop
    SQ.dispatch_event(1,SQ.INFO,"Engine Stopping");
    let write = "\x62\xff\x04\x04\x00\x00\x00\x00";
    CAN.tx(1, 0x0CFFF5F8, write, 8, CAN.EXT);
  }
},null);

```

Document Number
APN0041

Revision
A

Title
CAN Integration with BBA Pump Controller

Prepared By
JG

Approved By
NB

Page
15 of 21

7. Conclusions

The Senquip scripting language makes it simple to interface to a BBA LC40 pump controller. Most BBA controllers use the CATTRON CANplus standard for CAN communication 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.

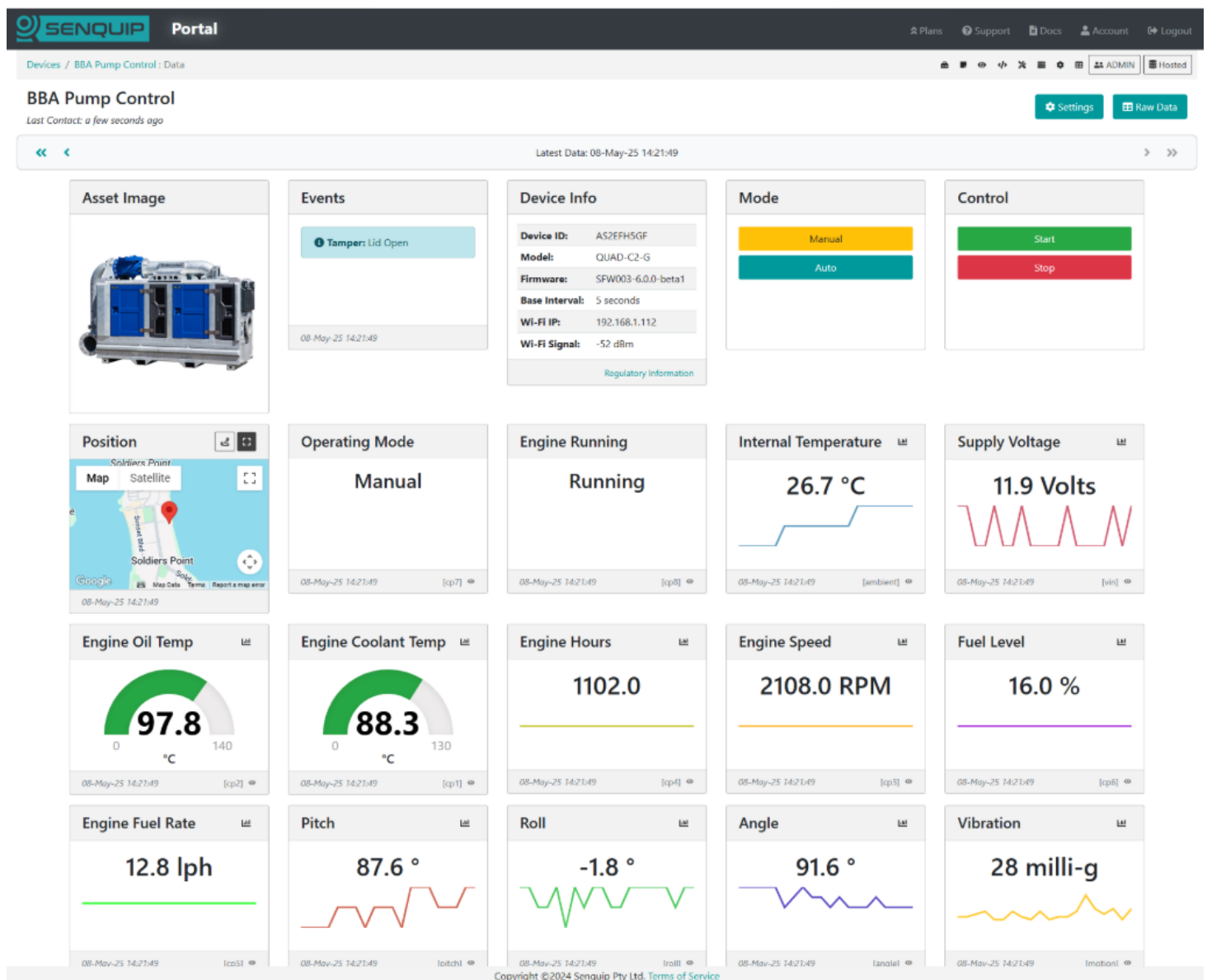


Figure 12 - Typical Portal Display with Minimal Parameters Shown

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
16 of 21

8. Appendix A – Database Parameters

CAN Addr	MODBUS Addr	Description	Scaling	Range	Notes
1	2	Major Version Number			
2	4	Minor Version Number			
8	16	Idle Speed	1 RPM/bit	0 - 8000	
9	18	Intermediate Speed	1 RPM/bit	0 - 8000	
10	20	Run Speed	1 RPM/bit	0 - 8000	
11	22	Max Speed	1 RPM/bit	0 - 8000	
43	86	Maintain Target Point	0.1 %/bit	0 - 1000	
45	90	Maintain Gain	0.0001 /bit	0 - 65535	
70	140	Autostart Transducer High Setpoint	0.1 %/bit	0 - 1000	
71	142	Autostart Transducer Low Setpoint	0.1 %/bit	0 - 1000	
78	156	Transducer 1 4mA Setup Value	1 /bit		32 bit floating point
79	158	Transducer 1 20mA Setup Value	1 /bit		32 bit floating point
81	162	Transducer 1 Display Units			
82	164	Transducer 1 Display Name Left String			
83	166	Transducer 1 Display Name Right String			
84	168	Transducer 2 4mA Setup Value	1 /bit		32 bit floating point
85	170	Transducer 2 20mA Setup Value	1 /bit		32 bit floating point
87	174	Transducer 2 Display Units			
88	176	Transducer 2 Display Name Left String			
89	178	Transducer 2 Display Name Right String			
90	180	Transducer 3 4mA Setup Value	1 /bit		32 bit floating point
91	182	Transducer 3 20mA Setup Value	1 /bit		32 bit floating point
93	186	Transducer 3 Display Units			
94	188	Transducer 3 Display Name Left String			
95	190	Transducer 3 Display Name Right String			
96	192	Transducer 4 4mA Setup Value	1 /bit		32 bit floating point
97	194	Transducer 4 20mA Setup Value	1 /bit		32 bit floating point
99	198	Transducer 4 Display Units			
100	200	Transducer 4 Display Name Left String			
101	202	Transducer 4 Display Name Right String			
102	204	Transducer 5 4mA Setup Value	1 /bit		32 bit floating point
103	206	Transducer 5 20mA Setup Value	1 /bit		32 bit floating point
105	210	Transducer 5 Display Units			
106	212	Transducer 5 Display Name Left String			
107	214	Transducer 5 Display Name Right String			
108	216	Transducer 6 4mA Setup Value	1 /bit		32 bit floating point

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
17 of 21

109	218	Transducer 6 20mA Setup Value	1 /bit		32 bit floating point
111	222	Transducer 6 Display Units			Decode Table
112	224	Transducer 6 Display Name Left String			
113	226	Transducer 6 Display Name Right String			
120	240	Pulse 1 Maximum Frequency	1 Hz/bit		32 bit floating point
121	242	Pulse 1 Maximum Flow Rate	1 /bit		32 bit floating point
122	244	Pulse 1 Flow Rate Display Units			
123	246	Pulse 1 Flow Rate Setup Units			
124	248	Pulse 1 Display Name Left String			
125	250	Pulse 1 Display Name Right String			
126	252	Pulse 2 Maximum Frequency	1 Hz/bit		32 bit floating point
127	254	Pulse 2 Maximum Flow Rate	1 /bit		32 bit floating point
128	256	Pulse 2 Flow Rate Display Units			
129	258	Pulse 2 Flow Rate Setup Units			
130	260	Pulse 2 Display Name Left String			
131	262	Pulse 2 Display Name Right String			
132	264	Pulse 1 Totalized Pulse Count Raw (Saved)	1 /bit		Raw Unscaled Data
133	266	Pulse 1 Totalized Setup Units			
134	268	Pulse 1 Totalized Display Units			
135	270	Pulse 1 Pulses Per 1 Unit Of Measurement	1 /bit		32 bit floating point
136	272	Pulse 2 Totalized Pulse Count Raw (Saved)	1 /bit		Raw Unscaled Data
137	274	Pulse 2 Totalized Setup Units			
138	276	Pulse 2 Totalized Display Units			
139	278	Pulse 2 Pulses Per 1 Unit Of Measurement	1 /bit		32 bit floating point
1000	2000	Current Engine Speed	1 RPMs/bit		
1003	2006	Engine Coolant Temperature	1 C/bit; Offset: -40		
1004	2008	Engine Oil Pressure	1 kPa/bit		
1005	2010	Battery Voltage	0.05 Volts/bit		
1006	2012	DPF Soot Load	1 %/bit		
1007	2014	DPF Ash Load	1 %/bit		
1013	2026	Engine Hours	0.05 Hours/bit		
1018	2036	Override Request Speed	1 RPMs/bit		
1028 (1028)	2056	Start/Stop Override			
1036 (1308)	2072	Panel Operating Mode			
1041	2082	Engine Running		0 - 1	
1054	2108	Number Of Active Faults	1 /bit		
1072	2144	Autostart Switch 1		0 - 1	
1073	2146	Autostart Switch 2		0 - 1	
1091	2182	Fuel Level	0.01 %/bit	0 - 10000	

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
18 of 21

1140	2280	Transducer 1 Current	1 uA/bit	0 - 65535	
1141	2282	Transducer 1 Percentage	0.1 %/bit	0-1000	
1142	2284	Transducer 2 Current	1 uA/bit	0 - 65535	
1143	2286	Transducer 2 Percentage	0.1 %/bit	0-1000	
1144	2288	Transducer 3 Current	1 uA/bit	0 - 65535	
1145	2290	Transducer 3 Percentage	0.1 %/bit	0-1000	
1146	2292	Transducer 4 Current	1 uA/bit	0 - 65535	
1147	2294	Transducer 4 Percentage	0.1 %/bit	0-1000	
1148	2296	Transducer 5 Current	1 uA/bit	0 - 65535	
1149	2298	Transducer 5 Percentage	0.1 %/bit	0-1000	
1150	2300	Transducer 6 Current	1 uA/bit	0 - 65535	
1151	2302	Transducer 6 Percentage	0.1 %/bit	0-1000	
1184	2368	Pulse 1 Totalized Pulse Count Raw (Runtime)	1 /bit		Raw Unscaled Data
1185	2370	Pulse 1 Frequency	1 Hz/bit		32 bit floating point
1186	2372	Pulse 1 Percent of Full Scale	0.1 %/bit	0-1000	
1187	2374	Pulse 2 Totalized Pulse Count Raw (Runtime)	1 /bit		Raw Unscaled Data
1188	2376	Pulse 2 Frequency	1 Hz/bit		32 bit floating point
1189	2378	Pulse 2 Percent of Full Scale	0.1 %/bit	0-1000	
1190	2380	Pulse 1 Scaled Total	1 /bit		32 bit floating point
1191	2382	Pulse 2 Scaled Total	1 /bit		32 bit floating point
1227	2454	Actual Torque	1 %/bit; Offset: -125		
1230	2460	Air Inlet Temperature	1 C/bit; Offset: -40		
1243	2486	Engine Oil Temperature	0.03125 C/bit; Offset: -273		
1248	2496	Engine Fuel Rate	0.05 l/h/bit		
1255	2510	Load Percentage At RPM	1 %/bit		
1256	2512	Requested Torque	1 %/bit; Offset: -125		
1257	2514	DEF Level	0.1 %/bit		
1268	2536	Scaled Transducer 1 Value			32 bit floating point
1269	2538	Scaled Transducer 2 Value			32 bit floating point
1270	2540	Scaled Transducer 3 Value			32 bit floating point
1271	2542	Scaled Transducer 4 Value			32 bit floating point
1272	2544	Scaled Transducer 5 Value			32 bit floating point
1273	2546	Scaled Transducer 6 Value			32 bit floating point

Document Number APN0041	Revision A	Prepared By JG	Approved By NB
Title CAN Integration with BBA Pump Controller			Page 19 of 21

9. Appendix B – Full Application Script

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

let params = [
  {'addr' : 1003, 'value' : NaN }, // Engine Coolant Temperature
  {'addr' : 1004, 'value' : NaN }, // Engine Oil Temperature
  {'addr' : 1000, 'value' : NaN }, // Engine Speed
  {'addr' : 1013, 'value' : NaN }, // Engine Hours
  {'addr' : 1248, 'value' : NaN }, // Engine Fuel Rate
  {'addr' : 1091, 'value' : NaN }, // Fuel Level
  {'addr' : 1036, 'value' : NaN }, // Panel Operating Mode
  {'addr' : 1041, 'value' : NaN }]; // Engine Running

/* Read a single CAN parameter */
function read_param(index) {
  let message = "\x21\xff" + SQ.encode(params[index].addr, -SQ.U16) + "\x00\x00\x00\x00"; // assemble the request message
  CAN.tx(1, 0x18FF5F8, message, 8, CAN.EXT); // transmit the request

  index++; // Increment the index
  if (index < params.length) {
    read_can_timed(index); // Start the next timed read
  }
  else{
    SQ.pre_handler_complete(); // allow the data handler to run
  }
}

/* Cycles through all the requests with a 100msec interval */
function read_can_timed(index) {
  Timer.set(100, 0, function(userdata) { // Pass index to the timer function as the userdata parameter
    read_param(userdata); // request a single parameter
  }, index);
}

/* Start the CAN requests */
let type = SQ.PRE_DURING;
SQ.set_pre_handler(type, function() {
  read_can_timed(0); // request the first CAN parameter
}, null);

/* Given the CAN id, returns the PGN */
function pgn(id) {
  return((id >> 8) & 0x0003FFFF);
}

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++) {
      if (pgn(obj.can1[i].id) === 65525) { // Database Command PGN
        let command = SQ.parse(obj.can1[i].data, 0, 2, -16); // retrieve the command byte
        if ((command & 0x1F) === 3) { // response to a read command
          let addr = SQ.parse(obj.can1[i].data, 4, 4, -16); // byte 2 and 3 contains address of the parameter
          if (addr === 1003) {
            let EngineCoolantTemp = SQ.parse(obj.can1[i].data, 8, 8, -16) - 40;
            SQ.dispatch("coolant_temp", EngineCoolantTemp);
          }
          if (addr === 1004) {

```

Document Number
APN0041

Revision
A

Prepared By
JG

Approved By
NB

Title
CAN Integration with BBA Pump Controller

Page
20 of 21

```

        let EngineOilTemperature = SQ.parse(obj.can1[i].data, 8, 8, -16);
        SQ.dispatch("oil_temp", EngineOilTemperature);
    }
    if (addr === 1000) {
        let EngineSpeed = SQ.parse(obj.can1[i].data, 8, 8, -16);
        SQ.dispatch("eng_speed", EngineSpeed);
    }
    if (addr === 1013) {
        let EngineHours = SQ.parse(obj.can1[i].data, 8, 8, -16) * 0.05;
        SQ.dispatch('eng_hrs', EngineHours);
    }
    if (addr === 1248) {
        let EngineFuelRate = SQ.parse(obj.can1[i].data, 8, 8, -16) * 0.05;
        SQ.dispatch('fuel_rate', EngineFuelRate);
    }
    if (addr === 1091) {
        let FuelLevel = SQ.parse(obj.can1[i].data, 8, 8, -16) * 0.01;
        SQ.dispatch('fuel_level', FuelLevel);
    }
    if (addr === 1036) {
        let PanelOperatingMode = SQ.parse(obj.can1[i].data, 8, 8, -16);
        if (PanelOperatingMode === 0) {
            SQ.dispatch(1, "Manual Start");
        }
        else if (PanelOperatingMode === 1) {
            SQ.dispatch(1, "Auto Start");
        }
        else if (PanelOperatingMode === 4) {
            SQ.dispatch(1, "Shutting Down");
        }
    }
    if (addr === 1041) {
        let EngineRunning = SQ.parse(obj.can1[i].data, 8, 8, -16);
        if (EngineRunning === 0) {
            SQ.dispatch_event(2, "Not Running");
        }
        else if (EngineRunning === 1) {
            SQ.dispatch_event(2, "Running");
        }
    }
    }
    }
    }, null);

//-----
SQ.set_trigger_handler(function(tp) {
    if (tp === 1) { //Manual
        let write = "\x62\xff\x1c\x05\x00\x00\x00\x00";
        CAN.tx(1, 0x0CFFF5F8, write, 8, CAN.EXT); // change to Manual mode
    }
    if (tp === 2) { //Auto Start
        let write = "\x62\xff\x1c\x05\x01\x00\x00\x00";
        CAN.tx(1, 0x0CFFF5F8, write, 8, CAN.EXT); // change to Auto Start mode
    }
    if (tp === 3) { // Start
        if(params[6].value === 1) {
            SQ.dispatch_event(1, SQ.INFO, "Engine Starting");
            let write = "\x62\xff\x04\x04\x01\x00\x00\x00";
            CAN.tx(1, 0x0CFFF5F8, write, 8, CAN.EXT);
        }
        else {
            SQ.dispatch_event(1, SQ.INFO, "Must be in Auto Start Mode to Remotely Start Engine");
        }
    }
});

```

Document Number
APN0041Revision
APrepared By
JGApproved By
NBTitle
CAN Integration with BBA Pump ControllerPage
21 of 21

```
}  
if (tp === 4) { // Stop  
  SQ.dispatch_event(1, SQ.INFO, "Engine Stopping");  
  let write = "\x62\xff\x04\x04\x00\x00\x00\x00";  
  CAN.tx(1, 0x0CFF5F8, write, 8, CAN.EXT);  
}  
, null);
```