SNIFFING FOR MODBUS PACKETS

1. Introduction

Deep Sea Electronics (DSE) manufactures a wide range of generator controllers, engine monitors, and power management modules commonly used in industrial and off-highway applications. Many of these controllers—including the Deep Sea E400 series—provide a Modbus RTU interface over RS485, allowing external systems to monitor operating parameters or issue control commands.

In many installations, a Senquip device is not required to act as a Modbus master; instead, it must listen to an existing Modbus network already operating between a PLC and a Deep Sea controller. In these cases, the Senquip ORB or QUAD can be connected in parallel to the RS485 pair and used to sniff Modbus traffic, validating requests and responses on the bus and extracting register values in real time.

This application note describes the process of connecting a Senquip device to an RS485 network between a PLC (master) and a Deep Sea E400 controller (slave). We will configure the device for passive listening, then implement a script to detect valid request/response pairs and publish the captured register data.

Figure 1 – E400 Engine Controller

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.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

2. Wiring the Senquip Device to the Deep Sea Controller

In this application note, we will use an ORB-C1-G wired to RS485 Port 1 on the Deep Sea controller.

The following connections are required:

Connection	Senquip ORB	Deep Sea DSE8610
RS485 B	Pin 6, B	Pin C8, B
RS485 A	Pin 7, A	Pin C2, A
GND	Pin 4, GND	C3, SCR

If the Senquip device and Deep Sea controller are at the end of the line on the RS485 bus, then a 120ohm termination resistor must be placed at each end of the line. The 120 ohm resistor on the Senquip device can be enabled as a setting.

Since the Senquip device and Deep Sea controller share a common power supply ground, the ground connection between pins 4 and C3 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.

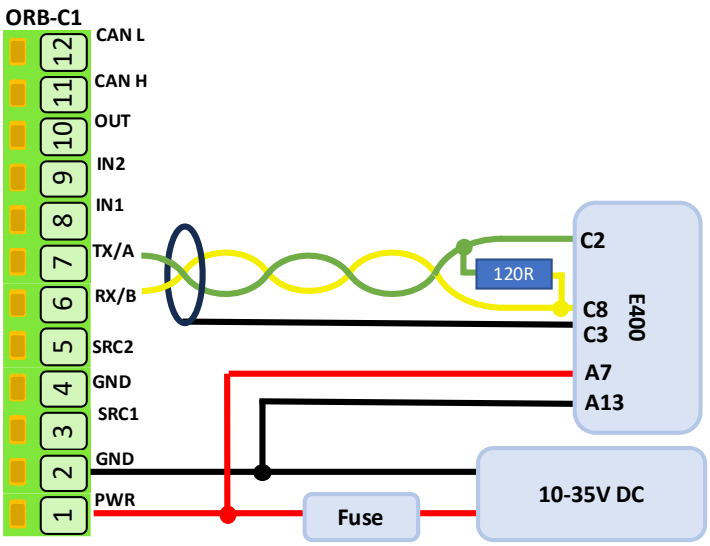

Figure 2 - Senquip ORB to E400 Wiring

3. Senquip Device Configuration

We get the communications specification for the Deep Sea controller from the Deep Sea GenComm Communications Protocol manual. GenComm was devised by Deep Sea Electronics Plc to provide a uniform standard for communicating with any generating set control equipment. It allows all telemetry information relevant to a generating set to be read from the control equipment, regardless of manufacturer or specification.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

Parameter	Value
Baud rate	115200
Data bits	8
Parity	None
Stop bits	1

The *Master inactivity timeout* on the Deep Sea controller should be set to at least twice the value of the Senquip base interval. For example, if the Senquip device reads from the controller every 5 seconds, the timeout should be set to at least 10 seconds.

The Senquip ORB serial port is set to match the Deep Sea controller requirements. Note that the serial port mode is specified as *Scripted*, meaning that the serial port will be completely controlled from within the script.

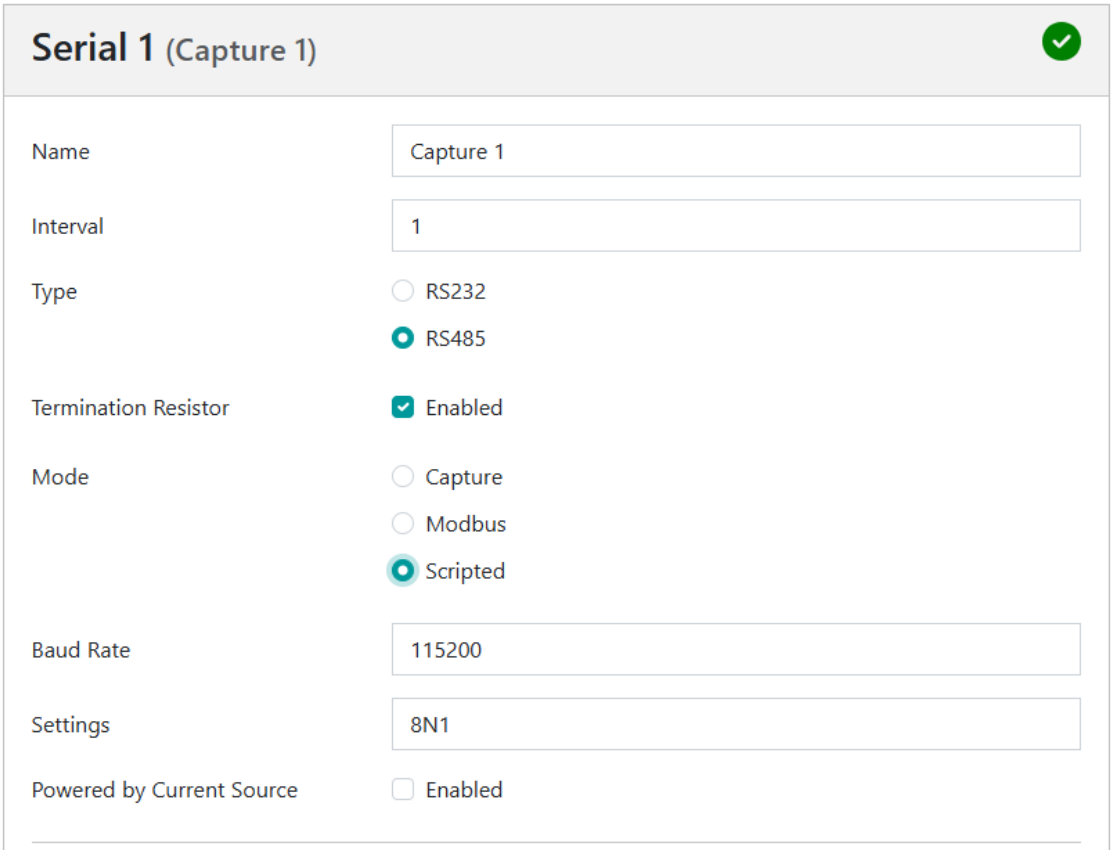

Figure 3 - Senquip ORB serial Port Settings

The default Modbus slave id for the controller is 10. This can be changed, for instance when multiple controllers are to be connected to a single Senquip device.

Registers are divided into 256 pages each containing up to 256 registers, the actual register address is obtained from the formula: register_address =page_number*256+register_offset. Note that the register addresses can change per model.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

We will read (sniff) the following registers from the Deep Sea controller:

Register	Page	Offset	Address	Scaling	Unit	Type
Oil pressure	4	0	1024	1	Kpa	16 bit unsigned
Coolant temperature	4	1	1025	1	°C	16 bit signed
Battery voltage	4	5	1029	0.1	V	16 bit unsigned
Stop led	190	14	48654	1		16 bit unsigned
Gen available led	190	21	48661	1		16 bit unsigned
Gen breaker led	190	19	48659	1		16 bit unsigned
Control mode	3	4	772	1		16 bit unsigned

The meaning of the allowable values for control mode are given below:

Mode	Description
0	Stop mode
1	Auto mode
2	Manual mode
3	Test on load mode
4	Auto with manual restore mode/Prohibit Return
5	User configuration mode
6	Test off load mode
7	Off Mode
8-65534	Reserved
65535	Unimplemented

Stop mode means stop the engine (generator) and in the case of 'automatic mains failure units' transfer the load to the mains if possible.

Auto mode means automatically start the engine (generator) in the event of a remote start signal or a mains-failure, and in the case of 'automatic mains failure units' transfer the load to the generator when available. When the remote start signal is removed or the mains returns, stop the engine (generator) and in the case of 'automatic mains failure units' transfer the load back to the mains.

Manual mode means start the engine (generator). With some control units it will also be necessary to press the start button before such a manual start is initiated. In the case of 'automatic mains failure units' do not transfer the load to the generator unless the mains fails.

4. Sniffing Modbus Registers in a Script

A Senquip device can automatically detect Modbus traffic without acting as a master. The set_modparse_handler() function registers a callback—a user-defined function that is triggered whenever the device sees a valid Modbus transaction on the bus.

In sniffer mode, the callback only runs when the Senquip device has observed a complete request/response pair with a valid CRC. This means the device effectively "eavesdrops" on the PLC reading registers from the Deep Sea controller, and your script receives a decoded structure containing:

- the slave address

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

- the function code
- the register address
- the number of registers returned
- and the raw data bytes

Inside the callback, your script can examine each transaction and extract the specific register values you care about. This allows the Senquip device to gather data simply by watching the existing Modbus traffic—no polling and no interference with the PLC or controller.

4.1. Load Required Libraries

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

These two library files provide the core Senquip functions used in the script:

- **senquip.js -** Includes general scripting utilities such as:
  - SQ.dispatch() for sending data to channels
  - SQ.parse() for decoding raw bytes
  - JSON helpers, timers, etc.
- **api_serial.js** - Provides access to the serial interfaces, including:
  - SERIAL.set_modparse_handler() for Modbus parsing and sniffing
  - SERIAL.getModbusResult() for decoding Modbus frames

Together, these libraries allow the script to read Modbus frames and process them into usable data.

4.2. Define the E400 Data Structure

```
let E400 = {
  'oil_press' : NaN,
  'coolant_temp': NaN,
  'batt_volt': NaN,
  'stop_led': NaN,
  'gen_available_led': NaN,
  'gen_breaker_led': NaN,
  'control_mode': NaN
};
```

This object stores the **latest decoded values** from the Deep Sea E400 controller.
Each field corresponds to one Modbus register:

- Numeric values start as NaN, meaning no data has been read yet.
- As valid Modbus frames arrive, the callback updates these fields.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

This object stores the latest decoded values from the Deep Sea E400 controller. Each field corresponds to one Modbus register:

- Numeric values start as NaN, meaning no data has been read yet.
- As valid Modbus frames arrive, the callback updates these fields.
- Later, the data handler dispatches these values to the Portal.

This provides a simple internal state for the device to work with.

4.3. Lookup Function for the Control Mode Register

```
function Control_Mode(i) {
  let enum_lookup = {
    '0': 'Stop',
    '1': 'Auto',
    '2': 'Manual',
    '3': 'Test on load',
    '4': 'Auto with manual restore',
    '5': 'User configuration',
    '6': 'Test off load',
    '7': 'Off',
  };
  return enum_lookup[i] || 'No Read';
}
```

The control mode register in the E400 returns a numeric code. This helper converts the numeric value into a human-readable string, e.g.:

- 1 → "Auto"
- 0 → "Stop"
- 7 → "Off"

If no valid value exists yet, it returns "No Read".

This improves clarity when displaying data on the Senquip Portal.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

4.4. Modbus Sniffer Configuration

```
// Possible Modbus parsing modes:
// 0 = Disabled
// 1 = Callback triggers for all requests and responses
// 2 = Callback triggers for only requests
// 3 = Callback triggers for valid responses that complete a request (sniffer bus
data)

/*  Definition of the structure passed to the modparse handler
{
  "is_response": false,
  "slave_addr": 1,
  "func": 3,
  "reg_addr": 0,
  "len": 1,
  "qty": 2,
  "data": "FE23"
}*/
let mode = 3;
let timeout_ms = 250;
let serial_ch = 1;
```

This section configures how the Senquip device listens for Modbus traffic. Mode = 3 selects sniffer mode, which triggers the callback only when:

- a valid request has been seen
- the matching valid response has also been seen
- CRCs are correct

This avoids noise and ensures only complete transactions are processed.

- timeout_ms = 250 is the maximum time allowed between a request and response.
- serial_ch = 1 selects **Serial Port 1** (RS485 interface).

The commented JSON block shows the structure passed into the script when a Modbus frame is detected.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

4.5. Register the Modbus Frame Callback

```
SERIAL.set_modparse_handler(serial_ch, mode, function(evdata) {
  let modbus = SERIAL.getModbusResult(evdata);

  SQ.dispatch(8, JSON.stringify(modbus)); // debug

  if (modbus.slave_addr === 10){ // Deepsea Modbus Address
    if (modbus.reg_addr === 1024){
      E400.oil_press = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 1025){
      E400.coolant_temp = SQ.parse(modbus.data,0,2,0,SQ.S16);
    }
    else if (modbus.reg_addr === 1029){
      E400.batt_volt = 0.1 * SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 48654){
      E400.stop_led = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 48661){
      E400.gen_available_led = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 48659){
      E400.gen_breaker_led = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 772){
      E400.control_mode = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
  }
}, timeout_ms, null);
```

What this callback does

Firstly, the callback converts the event data into a Modbus structure to extract:

- slave_addr
- func
- reg_addr
- qty, len
- the 2-byte data payload
- whether it was a request or response

The structure is dispatched to the Senquip Portal for debug purposes.

We then process the frames from the Deep Sea controller, checking that the frames are from the E400 with address 10, and then decoding the specific register. The value, which is passed as a text value is converted into a number with the SQ.parse function. For more detail on this function, see the Senquip Scripting Guide.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

4.6. Periodic Data Dispatch to Portal Channels

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

  SQ.dispatch(1,E400.oil_press);
  SQ.dispatch(2,E400.coolant_temp);
  SQ.dispatch(3,E400.batt_volt);
  SQ.dispatch(4,E400.stop_led);
  SQ.dispatch(5,E400.gen_available_led);
  SQ.dispatch(6,E400.gen_breaker_led);
  SQ.dispatch(7,Control_Mode(E400.control_mode));

}, null);
```

This handler runs when the device finishes collecting measurements. In the handler, we dispatch the collected E400 measurements to the Senquip Portal. Once dispatched, these values appear on charts, dashboards, and alerts exactly like native inputs.

5. Conclusions

By using the Senquip device in Modbus sniffer mode, it is possible to monitor the interaction between an existing PLC and a Deep Sea E400 controller without sending any Modbus requests of our own. The script presented in this application note passively listens on the RS485 bus, identifies valid request/response pairs, decodes the register values of interest, and publishes them to the Senquip Portal as standard telemetry channels.

This approach has several advantages:

- **Zero impact on the PLC or controller** – the device never becomes a Modbus master and never competes for bus access.
- **Always in sync with the PLC** – the Senquip device reports the same data the PLC is using, with no risk of reading registers the PLC is not polling.
- **Simple and extensible** – additional registers can be decoded simply by adding cases in the callback.
- **Robust** – the built-in Modbus parser handles CRC checks and request/response matching, ensuring only valid data is processed.

Modbus sniffing provides a powerful and non-intrusive way to extract operational information from systems where direct Modbus polling is not desired or not permitted. The example shown here provides a foundation that can be adapted to any Modbus-based controller or device on an RS485 network.

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

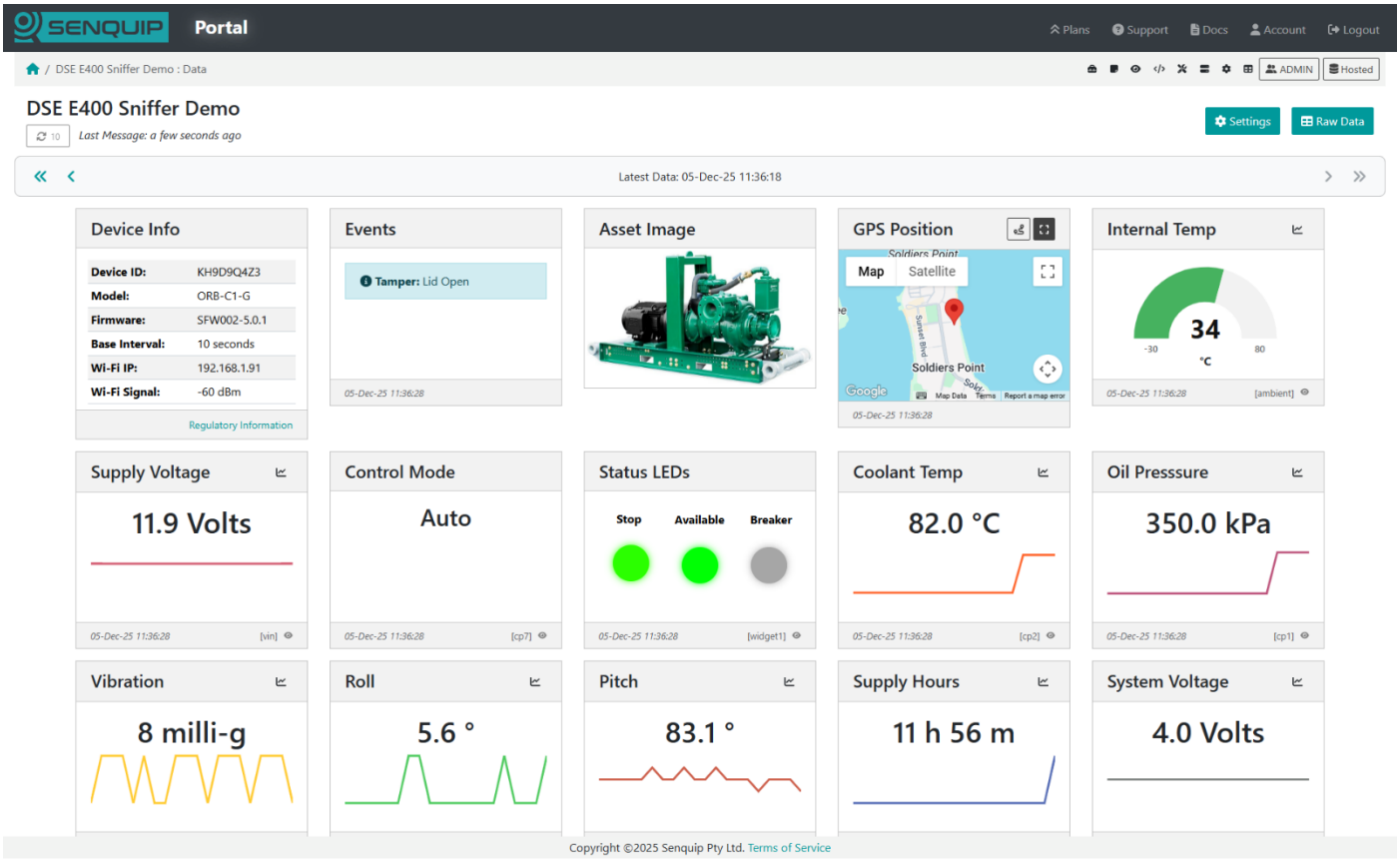

Figure 4 - Typical Portal Display with Minimal Parameters Shown

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd ("the Company") is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

6. Appendix A – Full Application Script

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

let E400 = {
  'oil_press' : NaN,
  'coolant_temp': NaN,
  'batt volt': NaN,
  'stop_led': NaN,
  'gen_available_led': NaN,
  'gen_breaker_led': NaN,
  'control_mode': NaN
};

function Control_Mode(i) {
  let enum_lookup = {
    '0': 'Stop',
    '1': 'Auto',
    '2': 'Manual',
    '3': 'Test on load',
    '4': 'Auto with manual restore',
    '5': 'User configuration',
    '6': 'Test off load',
    '7': 'Off',
  };
  return enum_lookup[i] || 'No Read';
}


// Possible Modbus parsing modes:
// 0 = Disabled
// 1 = Callback triggers for all requests and responses
// 2 = Callback triggers for only requests
// 3 = Callback triggers for valid responses that complete a request (sniffer bus data)

/*  Definition of the structure passed to the modparse handler
{
  "is_response": false,
  "slave_addr": 1,
  "func": 3,
  "reg_addr": 0,
  "len": 1,
  "qty": 2,
  "data": "FE23"
}*/
let mode = 3;
let timeout_ms = 250;
let serial_ch = 1;

SERIAL.set_modparse_handler(serial_ch, mode, function(evdata) {
  let modbus = SERIAL.getModbusResult(evdata);

  SQ.dispatch(8, JSON.stringify(modbus)); // debug

  if (modbus.slave_addr === 10){ // Deepsea Modbus Address
    if (modbus.reg_addr === 1024){
      E400.oil_press = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 1025){
      E400.coolant temp = SQ.parse(modbus.data,0,2,0,SQ.S16);
    }
    else if (modbus.reg_addr === 1029){
      E400.batt_volt = 0.1 * SQ.parse(modbus.data,0,2,0,SQ.U16);
```

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.

```
    }
    else if (modbus.reg_addr === 48654){
      E400.stop_led = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 48661){
      E400.gen_available_led = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 48659){
      E400.gen_breaker_led = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
    else if (modbus.reg_addr === 772){
      E400.control_mode = SQ.parse(modbus.data,0,2,0,SQ.U16);
    }
  }
}, timeout_ms, null);


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

  SQ.dispatch(1,E400.oil_press);
  SQ.dispatch(2,E400.coolant_temp);
  SQ.dispatch(3,E400.batt_volt);
  SQ.dispatch(4,E400.stop_led);
  SQ.dispatch(5,E400.gen_available_led);
  SQ.dispatch(6,E400.gen_breaker_led);
  SQ.dispatch(7,Control_Mode(E400.control_mode));

}, null);
```

Copyright © 2022 Senquip Pty Ltd. Senquip Pty Ltd (“the Company”) is the owner of the copyright and all confidential information in this document. The document and its text, images, diagrams, data and information it contains must not be copied or reproduced in whole or in part, in any form or by any means, without the prior written consent of the Company.