READING BANKS OF MODBUS DATA

1. Introduction

A Modbus device may place several measurements in a continuous range of holding registers. Reading the entire range in one request can be more efficient than requesting each measurement separately, particularly when the device limits how often it can be polled.

This application note uses a Y4000 water-quality sonde connected to a Senquip ORB as an example. The sonde provides dissolved oxygen, turbidity, conductivity, pH, temperature, chlorophyll and depth. The ORB reads a bank of 26 holding registers, extracts these seven measurements and sends them to Custom Parameters in the Senquip Portal.

The method can be adapted to other Modbus devices by changing the register range and the way the returned values are decoded.

Figure 1 – Y4000 Multi-Parameter Sonde

2. Sonde Register Map

The required measurements lie within holding-register addresses **9728 to 9753**. The ORB reads this entire range in one Modbus function 03 request. Some registers between the required measurements are returned but are not used.

Each measurement occupies two 16-bit registers and is supplied as a 32-bit IEEE 754 floating-point value in **DCBA byte order**.

Custom Parameter	Measurement	First register	Suggested Portal unit
CP1	Dissolved oxygen	9728	mg/L
CP2	Turbidity	9730	NTU
CP3	Conductivity	9732	Confirm against sonde configuration
CP4	pH	9734	pH
CP5	Temperature	9736	°C
CP6	Chlorophyll	9740	µg/L
CP7	Depth	9752	m

The script dispatches the floating-point values without scaling them. Check the suggested Portal units against the sonde configuration and a known measurement. In particular, confirm whether its conductivity reading is expressed in mS/cm or µS/cm.

3. ORB and Serial Configuration

The example requires an ORB running SFW002-5.0.3 or later. Connect the sonde to Serial 1 using RS485.

Setting	Value
ORB firmware	SFW002-5.0.3 or later
Serial interface	Serial 1, RS485
Serial mode	Scripted
Baud rate	9600
Data format	8N1
Sonde Modbus slave address	1
Modbus function	03, Read Holding Registers

Set Serial 1 to **Scripted** mode so the script can transmit the request for the register bank and process the response. Configure CP1 to CP7 with the names shown in the register table.

Confirm the sonde's slave address before using the script. If it is not 1, change `SLAVE_ADDR`.

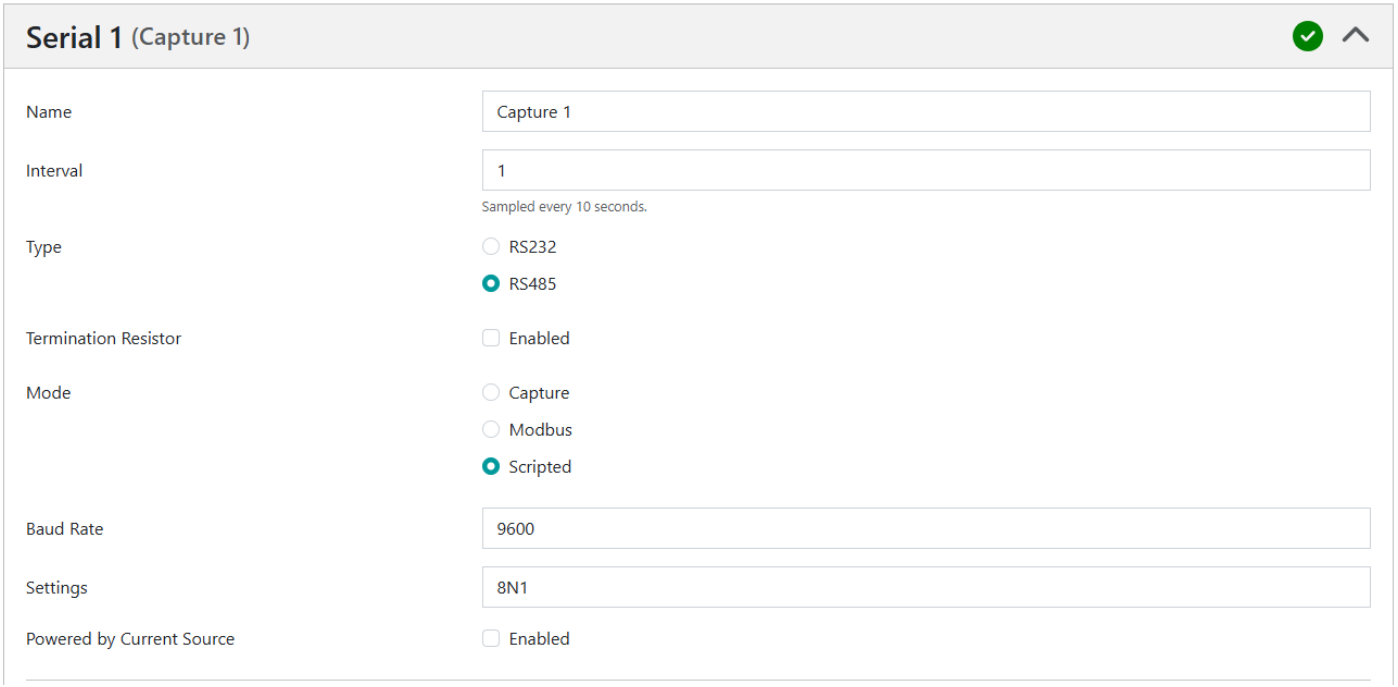

Figure 2 - Senquip ORB Serial Settings

4. Reading the Register Bank

The request starts at decimal address **9728**, which is **0x2600** in hexadecimal. It reads **26 registers**, covering addresses 9728 through 9753. Since each register contains two bytes, a successful response contains **52 register-data bytes**.

Before the Modbus RTU CRC is appended, the request consists of:

01 03 26 00 00 1A

Bytes	Meaning
01	Slave address 1
03	Read Holding Registers
26 00	Starting address 0x2600, decimal 9728
00 1A	Quantity 0x001A, or 26 registers

The script creates these bytes, calculates the CRC and transmits the completed request:

```
let cmd = chr(SLAVE_ADDR) + '\x03\x26\x00\x00\x1A';
let crc = SQ.crc(cmd, cmd.length);
let msg = cmd + SQ.encode(crc, -SQ.U16);

SERIAL.write(SERIAL_CH, msg, msg.length, SERIAL.LOOPBACK);
```

The ORB sends one request when each measurement cycle completes. Set the measurement interval to suit the required update rate and the sonde's permitted polling rate.

5. Receiving the Response

After the ORB sends the function 03 request, the sonde returns the requested register data in a Modbus RTU response. The script uses a ModParse handler to receive that response. ModParse checks the message framing and CRC before calling the script's callback, so the callback processes a complete, valid Modbus message.

The handler is configured as follows:

```
SERIAL.set_modparse_handler(SERIAL_CH, 3, function(evdata) {
  let modbus = SERIAL.getModbusResult(evdata);

  // Extract values from modbus.data.
}, 2000, null);
```

The second argument, `3`, selects responses that complete a previous request. The request is sent using `SERIAL.LOOPBACK`, which lets ModParse observe the outgoing request and match the sonde's response to it. For more information about the ModParse function, please see the Senquip Scripting Guide.

The callback receives `evdata`, which is converted to a Modbus result object with `SERIAL.getModbusResult(evdata)`. The object's `data` field contains the returned register bytes. It also provides the slave address, function code, starting register address, register quantity and data length. The `2000` argument gives ModParse up to 2000 ms to receive a complete message. If no complete response with a valid CRC arrives, the callback does not run.

6. Extracting the Measurements

The sonde returns 26 registers in address order, beginning at register 9728. Each register occupies two bytes. To find a measurement in the returned data, subtract 9728 from its first register address and multiply by two:

byte offset = (measurement's first register − 9728) × 2

For example, chlorophyll begins at register 9740:

(9740 − 9728) × 2 = 24

Its four bytes begin at offset 24 in modbus.data.

Measurement	First register	Byte offset
Dissolved oxygen	9728	0
Turbidity	9730	4
Conductivity	9732	8
pH	9734	12
Temperature	9736	16
Chlorophyll	9740	24
Depth	9752	48

The script stores the register address and Custom Parameter number for each measurement in a table. For each table entry, it calculates the byte offset, decodes four bytes as a DCBA IEEE 754 float and dispatches the result:

```
let channels = [
  {cp: 1, reg: 9728},  // Dissolved oxygen
  {cp: 2, reg: 9730},  // Turbidity
  {cp: 3, reg: 9732},  // Conductivity
  {cp: 4, reg: 9734},  // pH
  {cp: 5, reg: 9736},  // Temperature
  {cp: 6, reg: 9740},  // Chlorophyll
  {cp: 7, reg: 9752}   // Depth
];

.....

let byte_offset = (p.reg - FIRST_REG) * 2;
let value = SQ.parse(modbus.data, byte_offset, 4, 0, -SQ.FLOAT);

if (!isNaN(value)) {
  SQ.dispatch(p.cp, value, 3);
}
```

The 3 in SQ.dispatch() specifies three decimal places in the outgoing data. It does not scale the sonde's measurement.

In the Portal, give each Custom Parameter a descriptive name and the appropriate unit:

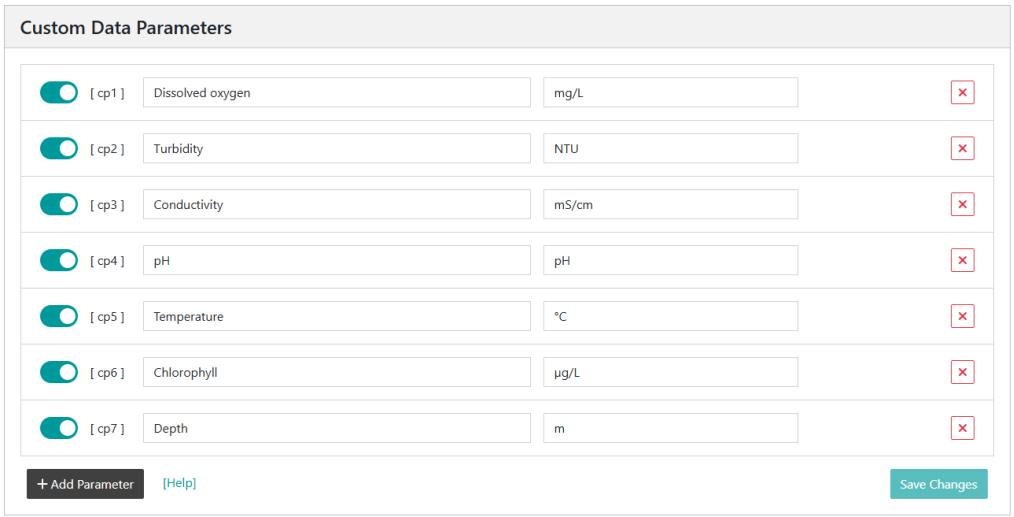

Figure 3 - Custom Parameter Table in the Scripting Window

7. Adapting the Method

For another Modbus device, identify a contiguous range containing the required measurements. Change the request's starting address and quantity, set FIRST_REG to the first address, and update the measurement table. Decode each measurement according to that device's data type, byte order and scaling.

If the required measurements are spread across separate banks, send a request for each bank and identify the response before parsing its values. Observe any request-size or polling-rate limits specified by the manufacturer.

8. Conclusion

The Senquip ORB retrieves the Y4000's 26-register bank with one Modbus request. ModParse delivers the valid response, and the script extracts and dispatches seven floating-point measurements from it. The register table makes it straightforward to select different values from the bank or adapt the approach to another Modbus device.

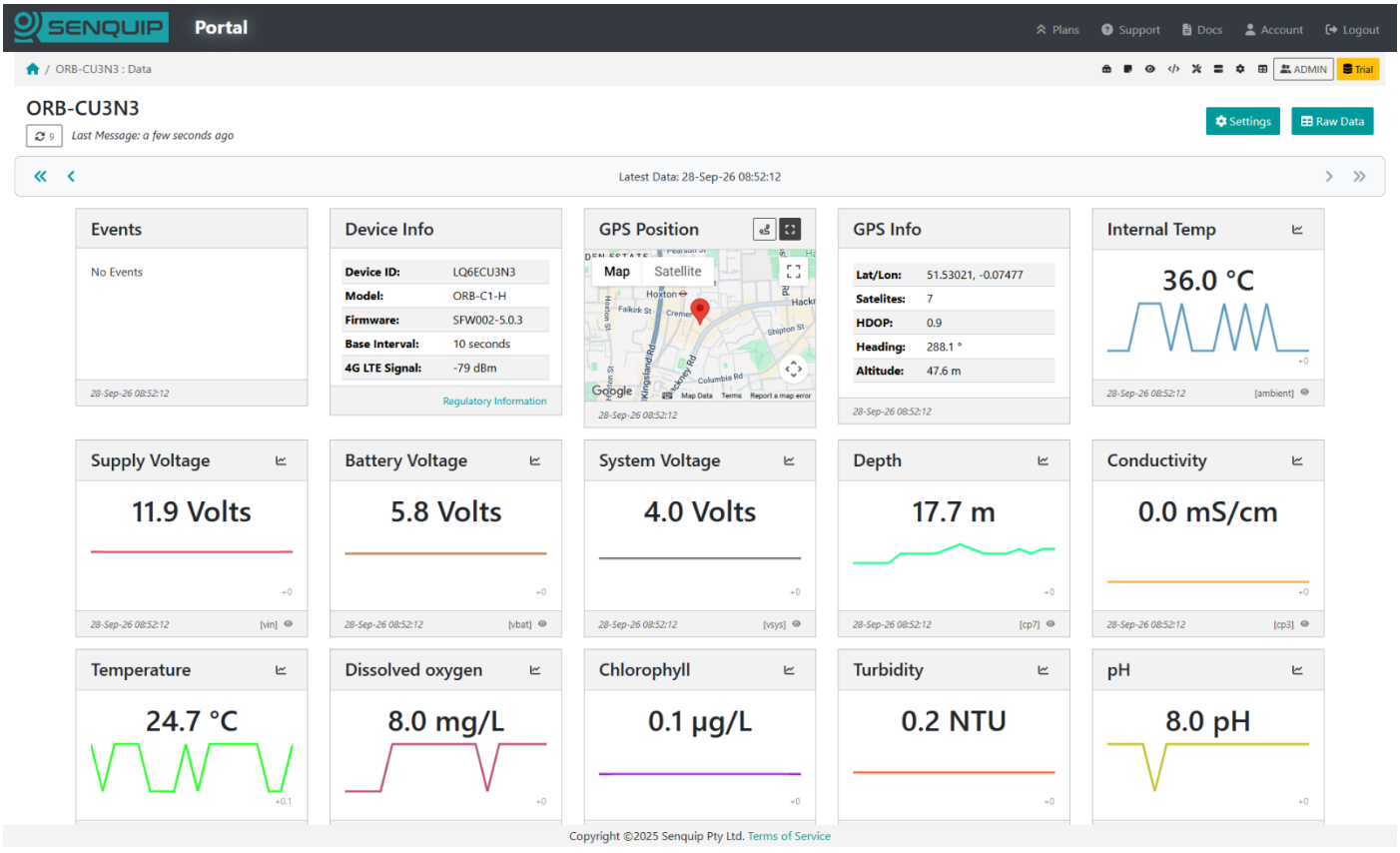

Figure 4 - Y4000 measurements displayed in the Senquip Portal

Appendix A – Full Script

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

let SLAVE_ADDR = 1;
let SERIAL_CH = 1;
let FIRST_REG = 9728;

let channels = [
  {cp: 1, reg: 9728},  // Dissolved oxygen
  {cp: 2, reg: 9730},  // Turbidity
  {cp: 3, reg: 9732},  // Conductivity
  {cp: 4, reg: 9734},  // pH
  {cp: 5, reg: 9736},  // Temperature
  {cp: 6, reg: 9740},  // Chlorophyll
  {cp: 7, reg: 9752}   // Depth
];

function sendRead() {
  // Slave address, function 03, start 0x2600, quantity 0x001A.
  let cmd = chr(SLAVE_ADDR) + '\x03\x26\x00\x00\x1A';
  let crc = SQ.crc(cmd, cmd.length);
  let msg = cmd + SQ.encode(crc, -SQ.U16);

  SERIAL.write(SERIAL_CH, msg, msg.length, SERIAL.LOOPBACK);
}

SERIAL.set_modparse_handler(SERIAL_CH, 3, function(evdata) {
  let modbus = SERIAL.getModbusResult(evdata);

  for (let i = 0; i < channels.length; i++) {
    let p = channels[i];
    let byte_offset = (p.reg - FIRST_REG) * 2;

    // The four received bytes are DCBA.
    let value = SQ.parse(modbus.data, byte_offset, 4, 0, -SQ.FLOAT);

    if (!isNaN(value)) {
      SQ.dispatch(p.cp, value, 3);
    }
  }
}, 2000, null);

SQ.set_data_handler(function() {
  sendRead();
}, null);
```