Modbus data mapping

Revision history

Date	Version	By	Reason
22nd April 2022	0.6	Mark Jones	Added new data points; modified the revision history to replace author's initials with their name; reworked 32 bit register section for unsigned and signed integers; the behaviour of tick adjusted for changes made in 1.3.9; restructured some sections; reduced the number of fonts used in the mappings tables; added more entries to the glossary; fixed mistakes in the register mapping where the bit width was accidentally omitted.
6th December 2021	0.5	Mark Jones	Updated Table of Contents; combined function code & address columns; removed references to operating mode; added experimental endpoints; adjusted document titles; updated the glossary; and adjusted the revision history for better clarity.
16th November 2021	0.4	Mark Jones	Never formally released - removed references to legacy operational behaviour to improve clarity of document. Added new binary data point (ground fault alarm). Reserved some inputs & registers for future features.
18th October 2019	0.3	Mark Jones	Added new mapping columns for clarity; a glossary section; appendices; a function code section; improved the mapping layout; added footnotes for units; and fixed some typographical errors.
1st September 2019	0.2	Mark Jones	Updated document to provide information on the different operation modes, configuration & adding the unexpected restarts parameter.
5th April 2019	0.1	Mark Jones	Updated to include direct ELM connections & additional monitor parameters (such as available storage).
10th February 2019	0.0	Mark Jones	Initial document.

Table of contents

Introduction

Please note - this document is liable to change. While we make an effort to keep the data points backwards compatible, we don't guarantee that this will always be the case. If in doubt, please contact Eagle Eye Power Solutions (http://www.eepowersolutions.com) for the most recent version of this document.

We are working with our clients and partners to allow the level of desired integration with their systems. There is still scope to update the mapping if further Modbus integration is required.

At the time of writing, the Vigilant BMS only provides Modbus over TCP/IP.

Glossary

- **Auxiliary input** - one of two binary digital inputs to the Vigilant BMS
- **Big endian** - The most significant bit comes first. Given the nybble **1000** - this number would be interpreted as '8'. See also Endianness and Little Endian.
- **BMS** - Battery monitoring system
- **Discrete input** - a read-only Modbus function that is one bit wide. It has function code 2xxxxx.
- **ELM** - electrolyte level monitor. Can be used for standalone electrolyte level monitoring in conjunction with the ELS.
- **ELS** - electrolyte level sensor. Determines whether the electrolyte has fallen below a certain level and whether the unit casing has exceeded a particular temperature threshold. Can be used with the standalone electrolyte level monitor or the Vigilant BMS.
- **Endianness**- describes the ordering the bits in a register. See also Big endian and Little endian.
- **Input register** - read-only Modbus function that is 16 bits wide. It has function code 4xxxxx.

- **Interconnect resistance** - the resistance of the connections between various parts of the battery system. These can include (but are not limited to) - charger connections, load bank connections, inter-unit connections and inter-tier connections.
- **Heartbeat** - see Tick.
- **Least significant register** - when used with another register, this is the byte that contains information pertaining the lower orders of data (2^0 to 2^7). See also Most significant register.
- **Little endian** - The least significant bit comes first. Given the nybble **1000** - this number would be interpreted as '1'. See also Big endian and Endianness.
- **Modbus** - a serial communications protocol published in 1979 by Modicon. Supports communication to & from multiple electronic devices.
- **Most significant register (MSR)** - when used with another register, this is the byte that contains the information pertaining to the higher orders of data (2^8 to 2^{15}). See also Least significant register.
- **Nybble** - half a byte.
- **Ripple voltage** - a periodic perturbation voltage that can be found on DC systems
- **TCP/IP** - literally: "transmission control protocol / internet protocol". Using TCP/IP allows devices to communicate over an Ethernet network in a manner that reduces the likelihood of losing packets in transmission at the expense of speed.
- **Tick** - a value that is updated every time the system refreshes the values. Used to determine if the Modbus data has gone stale. Sometimes referred to as the heartbeat.
- **Two's complement** - a common method of representing signed integers. For a more detailed discussion see the appendix.
- **UI** - user interface. How the user can interact with the system. Usually prefaced with "Web" (as in "Web UI") - and in this combination refers to the on-board system web site.
- **Unit** - an electrochemical container. Also frequently referred to as a "cell", "jar" or "bloc". Usually, multiple units are connected together to form a standby battery.
- **Unit voltage** - for a lead acid battery, a unit will contain one or more cells with (nominal) 2V cells.
- **Unit resistance** - the internal resistance found in most lead acid batteries. This is generally considered to be measured between the two posts on a typical unit.
- **Watchdog** - this is typically either a piece of software or a hardware circuit responsible for ensuring that a system is still functioning properly and has not locked up

Configuration

By default, Modbus is not enabled on the Vigilant BMS. It can be activated (and configured) from the settings page on the web UI.

The system will update its **discrete inputs** and **input registers**. At the time of writing, the Vigilant only outputs data and does not read values for configuration changes. All setting changes should be done through the web UI.

Function codes

The Vigilant BMS is designed to operate as a slave and is accessed by remote master units. The Vigilant BMS only provides access to read-only data.

This is clarified in the chart below:

Function Name	Function Code	Access Type	Data Size
Discrete Inputs	2	Read-only	1 bit

Function Name	Function Code	Access Type	Data Size
Input Registers	4	Read-only	16 bits

To poll the BMS for Unit 2's ripple voltage, one would request the register stored in address 40725. See the mapping section for a detailed list of all the available data points. To get Unit 3's ripple voltage the address is 40726.

Mapping

Discrete Inputs

Each data point is a boolean value (true or false). These are typically used for binary data or tracking error states. Please use the following table to determine how to interpret the data.

Table

Address	Name	Description	Data Interpretation
20001	AUX_1	State of auxiliary input #1	0 = inactive 1 = active
20002	AUX_2	State of auxiliary input #2	0 = inactive 1 = active
20003	LEG_1	Reserved for legacy systems	
…	…	… (and so forth)	…
20007	LEG_5	Reserved for legacy systems	
20008	SHUTOFF	Vigilant self termination warning	0 = Normal operation 1 = System preparing to power down
20009	ELS_EL_1	Sensor 1 electrolyte level [1]	0 = okay 1 = in alarm
20010	ELS_EL_2	Sensor 2 electrolyte level	0 = okay 1 = in alarm
…	…	… (and so forth)	…
20248	ELS_EL_240	Sensor 240 electrolyte level	0 = okay 1 = in alarm
20249	ELS_TL_1	Sensor 1 temperature level	0 = okay 1 = in alarm
20250	ELS_TL_2	Sensor 2 temperature level	0 = okay 1 = in alarm
…	…	… (and so forth)	…
20488	ELS_TL_240	Sensor 240 electrolyte level	0 = okay 1 = in alarm
20489	ELS_EA_1	**Experimental**: Sensor 1 electolyte alarm [1]	0 = okay 1 = in alarm

Address	Name	Description	Data Interpretation
20490	ELS_EA_2	**Experimental**: Sensor 2 electolyte alarm	0 = okay 1 = in alarm
…	…	… (and so forth)	…
20728	ELS_EA_240	**Experimental**: Sensor 240 electolyte alarm	0 = okay 1 = in alarm
20729	RES_1	Reserved	
20730	RES_2	Reserved	
…	…	… (and so forth)	…
20968	RES_240	Reserved	
20969	GROUND_FAULT	Ground fault alarm	0 = okay 1 = in alarm
20970	MODBUS_ERROR	**Experimental:** if the Modbus slave thinks it has detected an issue with data collection	0 = okay 1 = in alarm

Input Registers

Note: each Modbus register is 16 bits. While we have attempted to keep all the data types consistent, not all registers will contain 16-bit unsigned integers. Some are **32-bit** (split over two registers), some are **signed** integers and some are ***both***. Please see the appendices for how to parse the data.

To avoid rounding and other floating-point representation issues, most of the data is transferred in unintuitive units such as (d%, c°F & mV). These should be read as these standardised prefixes. In the prior case the prefixes are deci, centi and milli respectively to make deci-percent, centi-degrees Fahrenheit and millivolts. Please see footnotes in the chart for the less traditional prefixes.

Table

Address	Name	Description	Data Type	Data Interpretation
40001	Tick / Heartbeat	Used to determine if Modbus data has gone stale. Each time the Vigilant begins to refresh the Modbus data, the tick is increased.	16-bit unsigned integer	0 - 65535. If the value would reach 66536, then the ticker resets to 0. If the Modbus module restarts for any reason (for example, Vigilant self-termination is activated), then the tick will also reset to 0.
40002	Critical Alarms	The number of currently active critical alarms in the system	16-bit unsigned integer	Includes sensor alarms
40003	Warnings	The number of currently active warnings in the system	16-bit unsigned integer	Includes sensor warnings
40004	Unit voltage 1	The unit voltage measured by sensor 1	16-bit unsigned integer	mV
40005	Unit voltage 2	The unit voltage measured by sensor 1	16-bit unsigned integer	mV
...	...	... (and so forth)	...	...
40243	Unit voltage 240	The unit voltage measured by sensor 240	16-bit unsigned integer	mv

Address	Name	Description	Data Type	Data Interpretation
40244	Post Temperature C 1	The post temperature measured by sensor 1	16-bit signed two's complement integer	c°C [2]
40245	Post Temperature C 2	The post temperature measured by sensor 2	16-bit signed two's complement integer	c°C
...	...	... (and so forth)	...	...
40483	Post Temperature C 240	The post temperature measured by sensor 240	16-bit signed two's complement integer	c°C
40484	Post Temperature F 1	The post temperature measured by sensor 1	16-bit signed two's complement integer	c°F [3]
40485	Post Temperature F 2	The post temperature measured by sensor 2	16-bit signed two's complement integer	c°F
...	...	... (and so forth)	...	...
40723	Post temperature F 240	The post temperature measured by sensor 240	16-bit signed two's complement integer	c°F
40724	Unit ripple 1	The ripple voltage measured by sensor 1	16-bit unsigned integer	mV
40725	Unit ripple 2	The ripple voltage measured by sensor 2	16-bit unsigned integer	mV
...	...	... (and so forth)	...	...
40963	Unit ripple 240	The ripple voltage measured by sensor 240	16-bit unsigned integer	mV

Address	Name	Description	Data Type	Data Interpretation
40964	Unit resistance 1	The unit resistance measured by sensor 1	16-bit unsigned integer	μΩ
40965	Unit resistance 2	The unit resistance measured by sensor 2	16-bit unsigned integer	μΩ
…	…	…	…	…
41203	Unit resistance 240	The unit resistance measured by sensor 240	16-bit unsigned integer	μΩ
41204	Strap resistance 1	The strap resistance measured by sensor 1	16-bit unsigned integer	μΩ
41205	Strap resistance 2	The strap resistance measured by sensor 2	16-bit unsigned integer	μΩ
…	…	…	…	…
41443	Strap resistance 240	The strap resistance measured by sensor 240	16-bit unsigned integer	μΩ
41444	Unit critical alarms 1	The number of critical alarms measured by sensor 1	16-bit unsigned integer	
41445	Unit critical alarms 2	The number of critical alarms measured by sensor 2	16-bit unsigned integer	
…	…	… (and so forth)	…	…
41683	Unit critical alarms 240	The number of critical alarms measured by sensor 240	16-bit unsigned integer	

Address	Name	Description	Data Type	Data Interpretation
41684	Unit warnings 1	The number of warnings measured by sensor 1	16-bit unsigned integer	
41685	Unit warnings 2	The number of warnings measured by sensor 2	16-bit unsigned integer	
...	...	... (and so forth)	...	...
41923	Unit warnings 240	The number of critical alarms measured by sensor 240	16-bit unsigned integer	
41924	Battery current MSR	The battery current (1 of 2)	32 bit signed integer, two's complement	mA. Most significant register (MSR). Big-endian. To be used in conjunction with 1925. [4]
41925	Battery current LSR	The battery current (2 of 2)	32 bit signed integer, two's complement	mA. Least significant register (LSR). Big-endian. To be used in conjunction with 1924. [4]
41926	Battery voltage	The overall battery voltage	16-bit unsigned integer	cV [5]
41927	Ambient C	Ambient temperature	16-bit signed two's complement integer	c°C
41928	Ambient F	Ambient temperature	16-bit signed two's complement integer	c°F
41929	Memory used	The total % of Vigilant BMS storage used	16-bit unsigned integer	d% [6]
41930	Internal C	Vigilant BMS internal temperature	16-bit signed two's complement integer	c°C

Address	Name	Description	Data Type	Data Interpretation
41931	Internal F	Vigilant BMS internal temperature	16-bit signed two's complement integer	c°F
41932	Mode	Vigilant BMS operating mode	16-bit unsigned integer	0 = normal 1 = discharging
41933	Unexpected restarts	The number of detected unexpected restarts	16-bit unsigned integer	Can be used for compliance purposes. This can be used to determine whether the Vigilant has been unexpectedly power cycled or forcibly restarted by the on-board watchdog circuit. This increases when the Vigilant boots after an improper shutdown.
41934	Update time MSR	**Experimental:** time when the last Modbus update completed (1 of 2)	32-bit unsigned integer	seconds (timezone: UTC)
41935	Update time LSR	**Experimental:** time when the last Modbus update completed (2 of 2)	32-bit unsigned integer	seconds (timezone: UTC)
41936	Average unit voltage	The average unit voltage across the battery	16-bit signed integer	mV
41937	Average post temperature C	The average post temperature across the battery in Celsius	16-bit signed integer	c°C

Address	Name	Description	Data Type	Data Interpretation
41938	Average post temperature F	The average post temperature across the battery in Fahrenheit	16-bit signed integer	c°F
41939	Average unit resistance	The average unit resistance	16-bit signed integer	μΩ
41940	Average strap resistance	The average strap interconnect resistance	16-bit signed integer	μΩ
41941	Reserved	Reserved	-	-

Appendix

How to interpret register data

A single register is a big-endian 16 bit value, ie - most significant bit comes **first**. In this context (`//` is used to make comments about the values) For a typical, unsigned 16 bit register:

```
1000000000000000 // = 32768
0000000000000001 // = 1
```

Signed values will use two's complement notation. If you are inputting the Modbus data into third party software - it's possible that it will be just referred to as "signed" data. The rest of this section will explore how to process the data in more detail.

Signed and unsigned integers

In number theory, unsigned numbers are zero or larger. Signed numbers cover the whole range of numbers, less than zero (negative), zero and higher (positive). An integer is a whole/discrete number - ie 1,2,3,4,5,-100 etc. This is opposed to continuous (real) numbers where you can get fractional values - ie 1.25 and 2.5.

Two's complement

By using two's complement to represent the Vigilant BMS' signed integers, it allows us to provide discrete values without floating point errors and multiple zero values (-0 and +1).

Before two's complement, typically the most significant bit was the sign indicator (0 indicated a positive number, 1 indicated a negative number) - however, this resulted in a situation where -0 would not equal +0 without special handling. In our system, the leftmost bit is the most significant bit.

Two's complement "recycles" the negative 0 value to slightly increase the negative range of the values that can be represented. Using 8-bit numbers for clarity in the following examples:

bits	unsigned integer value	two's complement value
0000 0000	0	0
0101 0101	85	85
0111 1111	127	127
1000 0000	128	-128
1010 1010	170	-86
1111 1111	255	-1

As we can see from the above, the leftmost (most significant) bit still indicates whether the number is positive or negative. However, when the number becomes negative the ordering is less intuitive and the largest representable number in an unsigned integer is actually the smallest negative number in two's complement!

There are a couple of ways to decode a two's complement number:

- Look at the leftmost bit
    - If it is 0, then the number is positive and treat it like an unsigned integer
    - If it is 1, then the number is negative and should be decoded appropriately. This can be done by:
        - calculating the negation of the number, reading it as an unsigned integer and placing a negative sign at the front of it
        - by subtracting the maximum value that can be represented by subtracting a number based on the bitwidth from the unsigned value

Both methods of decoding two's complement numbers are described in the sections below.

Negation

One of the interesting properties of two's complement numbers is how negation is calculated. Put simply, you invert each bit and add 1.

For example to convert 28 into -28 (keeping with 8-bit numbers for clarity's sake):

```
00011100 // 28
11100011 // flip bits ...
11100100 // ... and add one. Result: -28
```

To return from -28 to 28:

```
11100100 // -28
00011011 // flip bits ...
00011100 // and add one. Result: 28
```

We can see from the above that two's complement negation will return to the original value if applied twice in succession!

To convert 1 into -1:

```
00000001 // 1
11111110 // flip bits ...
11111111 // and add one. Result: -1
```

To convert -86 to 86:

```
10101010 // -86
01010101 // flip bits ...
01010110 // and add one. Result: 86
```

Converting a register from an unsigned integer

The following pseudo-code will show you how to convert a register (interpreted as an unsigned integer) into its two's complement equivalent:

```
// convert unsigned to signed
bit_length = 16  // for 16-bit number
if (unsigned_value < 2^(bit_length-1))
    signed_value = unsigned_value  // if the leftmost bit is 0, it's positive
else
    signed_value = unsigned_value - 2^bit_length  // else it's negative
```

Please note that the ^ is being used as "to the power of " - x^{3-1} or 2^{32}

Handling 32-bit integers

Unsigned

In the following pseudo-code, << is the left shift operator and `read(a)` is a function that reads the data from the register with address `a`.

```
A = read(a)  // MSR
B = read(b)  // LSR
unsigned_value = (A << 16) + B
```

In the following hypothetical example:

```
A = read(a)                    // unsigned integer: 1      binary: 0000 0000
0000 0001
B = read(b)                    // unsigned integer: 1      binary: 0000 0000
0000 0001
unsigned_value = (A << 16) + B  // unsigned integer: 66537  binary: 0000 0000
0000 0001 0000 0000 0000 0001
```

Signed

Like the 16-bit signed integers, the 32-bit signed integers are also in two's complement notation. Combine the registers as above in the 32-bit unsigned example. Only the *combined value* is in two's complement notation, not the individual registers. Now you can calculate it in a manner similar to the 16-bit signed integer.

Taking the battery current addresses (**41924** & **41925**), as an example:

```
// combine the two registers
A = read(41924)  // MSR
B = read(41925)  // LSR
unsigned_value = (A << 16) + B

// convert unsigned to signed
if (unsigned_value < 2^31)
    signed_value = unsigned_value  // if the leftmost bit is 0, it's positive
else
    signed_value = unsigned_value - 2^32  // else it's negative
```

If the Vigilant monitor was reporting -100A, then the calculation would look like the following:

```
A = read(41924)                 // unsigned integer: 65534          binary: 1111
1111 1111 1110
B = read(41925)                 // unsigned integer: 31072          binary: 0111
1001 0110 0000
unsigned_value = (A << 16) + B  // unsigned integer: 4294867296     binary:
11111111 11111110 01111001 01100000


// convert unsigned to signed
if (unsigned_value < 2^31)  // 4294867296 is not less than 2147483648
    signed_value = unsigned_value
else
    signed_value = unsigned_value - 2^32  // signed_value = 4294867296 -
2147483648 = -100000 (mA)
```

Standardised prefixes

Chances are, you will have encountered these while working with battery systems already. When you measure a metric (ie - voltage, current), it has a unit (Volt (V), Ampere (A)). Sometimes the voltage and current is small (like 0.032A), it makes more sense to describe the current in **milli**-Amps (mA). 0.032A would become 32 mA. A milli- is a thousandth of a metric. There are a large range of other prefixes available - we have provided some of the more common metric prefixes below.

Name	Symbol	Numbering	English Wording
kilo	k	1,000	thousand
mega	M	1,000,000	million
giga	G	1,000,000,000	billion
tera	T	1,000,000,000,000	trillion
centi	c	0.01	hundredth
milli	m	0.001	thousandth
micro	µ	0.000001	millionth
nano	n	0.000000001	billionth

If you want to read more about these prefixes, this wikipedia article is an excellent starting point.

---

1. when the ELM detects a low electrolyte level, the monitor will reset the sensor 3 times before registering an alarm state. This is to prevent false readings from being registered during excessive charging. ↩ ↩

2. centidegrees Celsius. 1 Celsius = 100 centidegrees. To convert to Celsius, divide this number by 100 ↩

3. centidegrees Fahrenheit. 1 Fahrenheit= 100 centidegrees. To convert to Fahrenheit, divide this number by 100 ↩

4. see Appendix: Handling 32-bit integers ↩ ↩

5. centivolts. 1 Volt = 100cV. To convert to Volts, divide this number by 100. ↩

6. decipercent. 1% = 10 decipercent. To convert to a normal percentage, divide this number by 10 ↩