LabCon

MODBUS Specification

Version 03

1 Document history

DVers.:	Date	Reviser	Update Task	State
01	29.04.13	PI-FG	created	created
02	17.06.13	PI-FG	content updated	1st release
03	09.10.13	PI-FG	formatting changed	2nd release

2 Contents

3 Legal

pikkerton GmbH
Kienhorststr. 70
13403 Berlin
Germany

Telefon +49 (0) 30 3300724 -0
Telefax +49 (0) 30 3300724 -24
Internet www.pikkerton.de

4 General Information

According to the Modicon MODBUS specification, following register types where used:

16-bit input registers (read only access), address space: 30000 – 39999

16-bit holding registers (read and write access), address space: 40000 – 49999

Therefore, following MODBUS functions (function codes) have been implemented:

3	(0x03)	: read holding registers
4	(0x04)	: read input registers
6	(0x06)	: preset single register
16	(0x10)	: preset multiple registers
22	(0x16)	: mask/write 4X register
23	(0x17)	: read/write 4X register

All ZBS-Value instances (e.g. threshold 'TXT') are represented as multiple encoded MODBUS registers.

The Encoding of ZBS-Value instances is based on the ASN.1 encoding rules (refer to standard ITU-T recommendation X.690), which gives a specification of basic encoding rules (BER).

All Bit-register presentations refer to big endian registers.

ZBS-Parameters with read only access are represented as sets of MODBUS input registers.

ZBS-Parameters with read and write access are represented as sets of MODBUS holding registers.

5 ZBS-Value Encoding

All ZBS-Values consist of at least 3 16-bit MODBUS registers:

Tag Register																State (First Content-) Register																Second Content Register															
Tag Byte								Length Byte								State HighByte								State LowByte								HighByte								LowByte							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The **tag register** encoding refers to the standard X.690 BERs. Here, only primitive types of class 'Universal' are used (means upper 3 bits of tag byte (15-13) hold '0' always). Following data types have been implemented (with their appropriate tag):

ASN1 Boolean (1): 0x01

ASN1 Integer (2): 0x02

ASN1 OctetString (4): 0x04

ASN1 Real (Float) (9): 0x09

ASN1 UTC Time (23): 0x17

APPLICATION SPECIFIC:

There are two types of unsigned integers, 16-bit and 32-bit unsigned integers. They own the same type tag (0x02), and can only be distinguished by the length byte, which holds the byte count of the represented ZBS-Value type (Length-Byte = number of bytes required, to read the full represented ZBS value, including subsequently state register in counting, see section 7 for details).

To satisfy the needs of the wireless ZigBee backend, a **state register** has been introduced. Following states will be presented via this register:

State '***OK'***,	Code: 0x0000:	indicates successful read and write operations @ appropriate ZBS device
State '***OutOfDate'***,	Code: 0x0001:	indicates that no value has been read from ZBS device yet, thus, initial value is hold
State '***OutOfRange'***,	Code: 0x0002:	indicates that the target value for a ZBS write operation is out of the given ZBS-value range
State '***WriteFail'***,	Code: 0x0004:	indicates erroneous write operation @ ZBS device
State '***NotAvailable'***,	Code: 0x0008:	indicates that the desired value is just not available
State '***NotReadable'***,	Code: 0x0010:	indicates erroneous read operation @ ZBS device
State '***InvalidTarget'***,	Code: 0x0020:	indicates that desired ZBS device does not exist jet
State '***WriteIncomplete'***,	Code: 0x0040:	indicates forbidden ZBS value writings if MODBUS data is incomplete (see section 6 for details)

Important Notice:

Write operations on tag registers or state registers can take place at any time, without any fear to cause any unexpected system behavior or encoding errors, because writings to this registers will always be ignored. The LabCon MODBUS encoder updates this registers to values, representing valid ZBS-Value types and states automatically, whenever an arbitrary MODBUS query appears.

Encoding Examples:

A 16-bit unsigned integer, with read state '***NotAvailable***' and value '***128***' is encoded as follows:

Tag Byte: 0x02
Length Byte : 0x04
Tag Register : 0x0204
State Register : 0x0008
Value Register : 0x0080

Therefore the first three addresses in the appropriate address space (address 0 to 2) hold the values [0x0204 0x0008 0x0080], which accordingly, represent the type, state and value of a **single** ZBS-Parameter (e.g. 'TXT').

A Boolean holding '***True***', with write state '***WriteFail***'' is encoded as follows:

Tag Byte: 0x01
Length Byte : 0x04
Tag Register : 0x0104
State Register : 0x0004
Value Register : 0x0001
Resulting MODBUS registers: [0x0104 0x0004 0x0001]

A 32-bit unsigned integer holding '***4276993775*** ', with write state '***OK***' is encoded as follows:

Tag Byte: 0x02
Length Byte : 0x06
Tag Register : 0x0206
State Register : 0x0000
Value Register1 : 0xFEED
Value Register2 : 0xBEEF
Resulting MODBUS registers: [0x0206 0x0000 0xFEED 0xBEEF]

The Float encoding is completely equivalent to the methods presented in standard IEEE 754: 32-bit single precision floating point binary representation and therefore, not explained in detail here.

As you can assume, a ZBS device parameter of type float takes at least 2 registers for the 32-bit value plus 2 registers encoding overhead (tag register and state register).

A 32-bit signed float holding '-1.0001 ', with write state '***OutOfRange***' is encoded as follows:

Tag Byte: 0x09
Length Byte : 0x06
Tag Register : 0x0906
State Register : 0x0002
Value Register1 : 0xBF80
Value Register2 : 0x0347
Resulting MODBUS registers: [0x0906 0x0002 0xBF80 0x0347]

The type UTC Time is encoded like presented in ITU-T X.690, except omitting the trailing 'Z' character. A ZBS-parameter of this type takes 3 MODBUS registers plus 2 registers encoding overhead.

The following picture illustrates the content registers encoding of the date: 01.01.1970 00:00:00, which is the minimum value for the time type!

Content Register 2																Content Register 3																Content Register 4															
Year								Month								Day								Hour								Minutes								Seconds							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0							0	1							0	1							0	0							0	0							0	0						

Thus, MODBUS registers look like: [0x1708 0x0000 0x4601 0x0100 0x0000], if read/write result state leads to '***OK***'.

Furthermore, LabCon MODBUS encodes string values as single ASCII-characters. Therefore, each MODBUS register can take at most 2 ASCII-characters. In contrast to all other data types described here, a string value can have a variable length, which means that the tag register of a string type can change. During instantiation, each ZBS-String-Parameter is being initialized with a maximum length, to preserve all MODBUS registers required, to present the full string value. When reading or writing a string, the length byte of the tag register will be updated, according to the actual computed length of the actual read string, which will be of course, not the maximum length of the string in most cases. So if you want to determine exactly, how many registers have to be processed to read/overwrite the full string value, read out the length byte of the tag register first, and modify further requests accordingly.

As you can assume, padding of unwritten string registers is essential. To meet the requirement, the ASCII-Null-character (HEX-Code: 0x00) is used for padding. During string decoding, all trailing string padders will be removed from the targeted string value automatically.

String Encoding Example:

Assume a ZBS-Parameter of type string which could be read from ZBS device correctly (State Code: 0x0000), initialized with maximum length of 7 characters and omitting an initial value (empty string), is expressed as following MODBUS register range:

Registers: [0x0402 0x0000 0x0000 0x0000 0x0000 0x00**00**]

As you can see, the last byte is a string **padder**, because of the odd character length of 7. You can see further, the length byte of the tag register holds 0x02 (= 2 bytes for state register plus 0 bytes for content registers), which is confusing at a first glance (refer to initialized length of 7 chars = 4 MODBUS registers (8 bytes) required), but logical. As mentioned before, we omitted the initial value for the string, which leads to an empty string with length of zero, and so also will be encoded correctly as far as all other ASCII-chars in range 0 to 255.

Writing the string '***Hallo!***' to a ZBS device, which owns a string parameter with a specification described above, assuming a successful ZBS write operation, leads to following MODBUS register values: [0x0408 0x0000 0x4861 0x6C6C 0x6F21 0x0000]

6 Address Mapping

When reading or writing a range of MODBUS registers, the requested register addresses (numbers) are being mapped to the appropriate ZBS-Parameter.

Assume a ZBS-MODBUS-Device representation owning holding registers in following address space:

Addresses: [[0, 1, 2], [3, 4, 5], [6, 7, 8, 9], [10, 11, 12, 13], [14, 15, 16, 17, 18]]

where each subset in the address space represents a ZBS-Parameter (=MODBUS-Instance (MBI)).

Let's say: [0, 1, 2] = TXT (MBI 0)
[3, 4, 5] = POW (MBI 1)
[6, 7, 8, 9] = FREQ (MBI 2)
[10, 11, 12, 13] = LOAD (MBI 3)
[14, 15, 16, 17, 18] = LGTS (MBI 4)

Let's say we have the request: from address 0, read 4 values: this leads to request address range:
[0, 1, 2, 3]. MBI 0 (TXT) is read completely, which of course triggers a read operation for this parameter @ the appropriate ZBS device. When reaching MBI 1, only value at address 3 is requested but: to ensure the MODBUS register values are consistent with the current values of the appropriate ZBS device, the reading of only 1 register of a MBI results in a complete update of all registers of the MBI, so MBI 1 (POW) will get a full register update in range [3 4 5]!

When writing MODBUS registers, a write operation to the appropriate ZBS device will be triggered immediately, if all required content registers of the appropriate MBIs have been written. Afterwards, the result of the write operation will be encoded to the state register and the new values will be saved to the appropriate MODBUS registers.
If not all required content registers of an MBI have been written, the write operation will be ignored at the designated ZBS device. Instead, the latest value read from the ZBS device will be held to ensure consistency between the values of the ZBS devices and the values held in the MODBUS registers. Furthermore, the incomplete write operation will be indicated via the state register of the appropriate MBI, using state code 0x0040 ('*WriteIncomplete*').

Thus, a write request of 5 values, starting @ address 2, leads to successful ZBS write triggering of MBI 0 and MBI 1 (TXT and POW), but results in a reread of the latest ZBS value known @ MBI 2 (FREQ), because of the incomplete writing of the MBIs' content registers. We of course assume here, that the write query holds valid values appropriate to the MBIs' types, causing no other errors.

Exception: writing string types. As said before, strings have variable length and thus will not be written completely in most cases. Therefore, they are allowed to be updated to the designated ZBS device parameter, if any content register(s) has (have) been written correctly. Furthermore, target values of type string, which exceed the maximum length of the designated MBI, will be cut right-sided automatically to fit in.

7 Computing Value Length and Value Register Addresses

To determine the actual length of the targeted ZBS parameter (MBI) value, the length byte has been introduced. It holds the length **in byte** and is built as following:

Length-Byte = (RegisterCount * RegisterWidth) + 2 bytes

Parameter "*RegisterCount*" is of course varying according to the type of the MBI. Parameter "*RegisterWidth*" is tied to 2 bytes (use of 16-bit MODBUS registers only) and the two additional bytes represent the length of the state register, which is owned by all MBIs.

Length Encoding Example:

Assume a MBI of type float (*RegisterCount* = 2), which could be read from ZBS device correctly (State Code: 0x0000). The length is encoded as follows: **Length = (2 * 2) + 2 = 6 bytes**
This indicates that a read operation of subsequently 6 bytes (= 3 MODBUS registers) is required, to acquire the full ZBS value representation (value itself plus state).

Computing the MODBUS register address of a targeted MBI is quietly similar. All you need to know is the MODBUS start address and type of the preceding MBI, because the type of a MBI determines the maximum length of the MBI. Also the encoding overhead is required, which has a fix value of 2 registers (= 4 bytes) regarding to tag- and state-registers.

Exception: MBIs of type string, where MBIs' length byte holds the byte-length of the current string value, not its maximum length. In this case, the maximum length of the according MBI is required, instead of current value held in length-byte, to do correct MODBUS address computations.

Address Computing Example:

Assume the given address range described in top of preceding section 6. Thus, the tag-register of
MBI 0 (TXT) holds value [0x0204] (16-bit integer). To determine the start address of subsequently MBI 1 we compute (unit: number of register):

Start address of MBI 1 = Start address of MBI 0 + encoding overhead + register count of MBI 0
= 0 + 2 + 1
= 3

Where: ***register count*** of MBI 0 ***= (length-byte – 2 bytes) / RegisterWidth*** = (4 – 2) / 2 = 1

Start address of MBI 4 = Start address of MBI 3 + encoding overhead + register count of MBI 3
= 10 + 2 + (6 – 2) / 2
= 14

8 References

MODBUS Modicon Specification:

http://modbus.org/docs/PI_MBUS_300.pdf (Effective 17.06.2013)

Standard ITU-T recommendation X.690:

http://www.itu.int/ITU-T/studygroups/com17/languages/X.691-0207.pdf (Effective 17.06.2013)

IEEE 754-2008 - 32-bit single precision floating point binary representation:

https://en.wikipedia.org/wiki/IEEE_floating_point (Effective 17.06.2013)

http://en.wikipedia.org/wiki/Single-precision_floating-point_format (Effective 17.06.2013)

http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4610933 (Effective 17.06.2013)