Bandweaver

ACCESS MODBUS APPLICATION NOTE

DISTRIBUTED TEMPERATURE SENSING

Bandweaver Product Family: **Access**

DTS-AC-01-MAN-002

Revision 1.1, September 2019

Bandweaver Technology Ltd., Unit 22 Clasford Farm Stables, Aldershot Road, Guildford GU3 3HQ, UK
info@bandweaver.com

Revision History

Rev	Date	Reason for issue	Author	Reviewer	Reviewer
1	09-Sep-2019	First release	Kin Wei Lee	Ryan Da Costa	
1.1	11-Sep-2019	Corrected unlinked references	E. Fernandes		

Contents

Revision History 2
1 Access Modbus Application 4
1.1 Essential Reading 4
1.2 Implemented Functions 4
1.3 Function Code Categories 4
1.4 Handled Exceptions: 5
1.5 Access Supported Modbus Functions 5
1.6 Communication Parameters 6
2 Basic Conversions 7
3 Representation of data 8
3.1 Temperatures and Distances 8
3.2 Other Scalings 8
4 MODBUS Worked Examples 9
5 Basic Operations 10
5.1 Starting and Stopping the measurement 10
5.1.1 Auto mode example 10
5.1.2 Stop measurement example 10
5.2 Read Temperature 11
5.2.1 Example 11
5.2.2 Data Processing 12
5.3 Zone Configuration 12
5.3.1 Reading Zone Configuration 13
5.3.2 Data Processing 13
5.4 Modifying Zones 14
5.4.1 Zone Modification 14
5.4.2 Permanent data save to flash 15
5.5 Zone Status 15
5.5.1 Data Conversion Example 16
5.6 Alarm Status 17
5.6.1 Data Conversion Example 17
5.7 Reset Alarms 17
6 Intermediate Operations 18
6.1 Time 18
6.2 Read 18
6.3 Data Conversion Example 19
6.4 Write to Real Time Clock (RTC) 19
6.5 Save to RTC 19
6.6 Measurement Parameters 20
6.6.1 Reading Measurement Parameters 20
6.6.2 Writing Measurement Parameters 20
6.6.3 To Save Measurement Parameters To Flash 21
7 Advanced Operations 22
7.1 Historical Data 22
7.2 System Diagnostics 22
7.2.1 System Information 22
7.2.2 Modifying Core Parameters 23
7.3 Files Modes 23
7.4 Modbus Exception Handling 23
7.4.1 Illegal Function test 24
7.4.2 Illegal data address test 24
7.4.3 Illegal data value test – incorrect data length 24
7.4.4 Slave Device Failure Test 24
8 Application Development 25
8.1 Basic Operation 25
8.1.1 Reading Temperature Data 1 25
8.1.2 Database Structure 25
8.2 Zone id 26
8.3 Build > 3xx 26
8.4 CRC 26

1 Access Modbus Application

Access uses MODBUS as the main message protocol for client/server communications. It is implemented using an asynchronous serial transmission over EIA/TIA-485, or more commonly RS485.

Access has a built in RS232 to USB converter for the users' convenience, as this allows a direct USB cable connection to emulate RS232 communications.

Access can be configured, and data viewed using any MODBUS master software.

Access can be accessed (pun?) using the supplied software (ACE) however it may be necessary to integrate Access directly into a 3rd party SCADA system. The following information is a guide to implementing a MODBUS interface with Access, using working examples of code.

1.1 Essential Reading

- Modbus Application Protocol V1.1b (www.modbus.org)
- MODBUS over Serial Line specification and implementation guide V1.02 Essential Documents (www.modbus.org)
- Access Modbus Registry Map vX.X.xls – supplied by Bandweaver

The examples below have been taken from a working DTS and use working commands. Once you have worked through the Basic Operations you will be familiar with the MODBUS protocol and its operation with Access DTS.

1.2 Implemented Functions

RS232/RS485

Implementation of Modbus model

Dynamic mapping between Modbus data model and device application (this essentially mean the DTS doesn't have a dedicated memory stack for the MODBUS data but rather responds to data request by reading and then sending data from existing data structures.)

1.3 Function Code Categories

Access implementation of the MODBUS protocol results in 4 main registry tables.

Function 2: 1 bit data that can be read only

Function 5: 1 bit data that can be read and written

Function 4: 16 bit data that can be read only

Function 3/16: 16 bit data that can be read and written

The reason for this data separation from a temperature sensor point of view is clear. For example, the temperature data generated by the DTS should only be read and not modified. However, the measurement parameters, such as the measurement time, must be modifiable and readable.

An action (1bit data) such as start and stop measurement must be settable however a fault status should be read only.

Two additional functions are also used.

Function 20: File Record Read

Function 21: File Record Write

These functions allow for large data structures to be transferred to and from Access. These large data structures may include a mix of text and numbers which don't fit well with the existing 16bit numerical data of the other functions.

Information using the Records format:

Machine information

User and Version details

Reading raw data

1.4 Handled Exceptions:

01 illegal function

02 illegal data address

03 illegal data value

04 slave device failure

1.5 Access Supported Modbus Functions

Discrete output Coils

0x5 write single coil

Discrete input coils

0x02 read discrete inputs

Analog output holding register

0x03 read multiple

0x06 write single

0x16 write multiple

Analog Input Registers

0x04 Read Multiple

1.6 Communication Parameters

Baud Rate:	115200bps
Parity:	None
Stop Bits:	1

2 Basic Conversions

Signed, Unsigned Integers, 16 bit or 32 bit numbers?

The MODBUS protocol does not include data type information, for example whether or not the data is signed +ve or –ve. This information must be explicitly defined by the system design beforehand and is usually documented in the registry map.

A negative number will be stored in the registry in Two's Complement format. The host software will need to know that it is dealing with signed values in order to handle them correctly. (If the value is > 0x7FFF deal as if it were unsigned and then subtract 2^16).

Larger 32bit numbers can be stored in the MODBUS registry by occupying two data locations. However, it is upto the system designer to define the order of the upper and lower 16bits.

3 Representation of data

In general, the Access registry map will indicate the integer type, signed or unsigned, and a scaling factor required to modify a value. When scaling factors are used the integer registers must be multiplied by this value to be read correctly in the given units. And conversely when a value is written into the registry map the value must be divided by this scaling factor and rounded to the closest integer.

3.1 Temperatures and Distances

Temperature is saved into the registry as a scaled fixed point 16bit number. The register value multiplied by the scale factor (0.01) will give the true temperature in degrees centrigrade (ºC). In this case the 16bit signed integer can save temperatures in the range -327.68ºC to +327.67ºC. This is also true for temperature thresholds for the alarms.

Distance units are signed integers with a scale factor of 1. In this case distances are in the range -32768m to +32767m.

3.2 Other Scalings

Slope, refractive index, and sample_length (MB_SLOPE, MB_REFRACTIVE_INDEX, and MB_SAMLE_LENGTH) need to be multiplied by 0.001, to obtain their real world values.

4 MODBUS Worked Examples

The following instructions describe actual MODBUS commands and their use with ACCESS.

At a basic level a MODBUS command message consists of the following data

Slave Address	Unique ID number between 1 and 255 given to a MODBUS device
Function Code	Indication of the data access method
Registry Address	A discrete location on the registry map where data may be located
Registry Data	The actual data
CRC	Error checking code – pseudo code for this algorithm can be found in MODBUS over Serial Line Specification and Implementation Guide V1.02

For readability any ascii code that is displayed is split up the correct data location.

For example:

COMMAND:

0f 06 0000 0000 88e4

5 Basic Operations

The following are the basic operations to integrate Access into a MODBUS system.

1. Starting and stopping the measurement
2. Reading the distributed temperature
3. Viewing the zone configuration
4. Modifying the zones configuration and saving to permanent memory
5. Zone status
6. Alarm status
7. Reading and modifying measurement parameters
8. Reset Alarms

5.1 Starting and Stopping the measurement

This is handled by the MB_OP_MODE register, 0x0000, and the examples are for a slave at 15 (0x0F).

5.1.1 Auto mode example

This starts the measurement and set the Access to continuously collect and process data.

Address	1 byte	Slave 15	0x0f
Function code	1 byte		0x06
register address	2 bytes	MB_OP_MODE	0x0000
register Value	2 bytes	0 - auto	0x0000
CRC	2 bytes		0x4131

COMMAND:
0f 06 0000 0000 88e4
REPLY:
0f 06 0000 0000 88e4

5.1.2 Stop measurement example

Address	1 byte	Slave 15	0x0f
Function code	1 byte		0x06
register address	2 bytes	MB_OP_MODE	0x0000

register Value	2 bytes	3 - stop	0x0003
CRC	2 bytes		0xc8e5

COMMAND:
0f 06 0000 0003 c8e5
REPLY:
0f 06 0000 0003 c8e5

5.2 Read Temperature

The maximum data size of a received Modbus PDU (MODBUS message) is 253 bytes so we are limited to about 125 registers. The temperature data is about 550 points so multiple registers reads will be required. A time separation of >20ms is required between Modbus commands.

It is also recommend to read the temperature header information which includes the measurement id, timestamp, block size, data start position and data sample length.

The temperature data in the registry map has been multiplied by 100, so that the stored 16bit integer can represent a number between -327.68ºC to +327.67ºC. After obtaining the value from the registry map it must be divided by 100 and stored locally as a float.

The measurement must be stopped before changing the following parameters.

5.2.1 Example

The following code reads 125 (0x7d) samples the location MB_TEMPERATURE_START + 125, from slave 15 (0x0F) from table 0x04. 0x7D is the maximum number of registers that can be read and that will fit in a Modbus message.

	Format	Description	command
Address	1 byte	Slave 15	0x0f
Function code	1 byte		0x04
Output address	2 bytes	MB_TEMPERATURE_START + 125	0x00eb
Output Value	2 bytes		0xff00
CRC	2 bytes		0x4131

COMMAND:
0f 04 00eb 007d 4131
REPLY:
0f # slave address
04 # function code
fa # number of bytes

0a99 0aa9 0ac2 0abf 0ab1 0ac6 0aa9 0aaf 0aaf 0abb 0a88 0abf 0a99 0a91 0a96 0ab4 0aa0 0ab2 0ab1 0ae4 0abd 0a8c 0a3e 0a6e 0a91 0aa6 0a89 0ab6 0ab6 0a85 0ac5 0ab8 0aa6 0a93 0a6d 0aa3 0a5e 0a7a 0a8c 0a8b 0a89 0a44 0a42 0a4f 0a87 0a50 0a96 0a88 0a9d 0a57 0aa6 0a48 0a72 0a74 0a16 0a6e 0a4d 0a85 0a0e 0a3c 0a1b 0a40 09fa 09ab 09cd 09e4 0a0b 0a11 0a0d 0a0d 09d3 0a24 0a14 0a05 09e7 0a38 0a03 0a0b 0a0e 0a0d 0a10 0a0f 09e1 0a37 0a07 0a01 0a24 0a08 0a3c 09e6 0a0d 0a0e 0a1f 09ff 0a20 0a0e 0a2e 09d8 0a1f 0a1a 0a23 09bd 09d7 09f9 0a05 09f1 09e1 0a24 0a0a 0a14 09dc 09fb 09dc 0a03 09e8 0a0e 0a04 0a2c 09f8 0a04 0a2e 09f8 09ef 09e6 0a14

2703

5.2.2 Data Processing

Taking the first 4 values of the output data as an example:

Raw value	0x0a99	0x0aa9	0x0ac2	0x0abf	0x0ab1
Type	signed	signed	signed	signed	signed
Converted	+2713	+2713	+2754	+2751	+2737
scaling	x0.01	x0.01	x0.01	x0.01	x0.01
final	27.13ºC	27.13ºC	27.54ºC	27.51ºC	27.37ºC

5.3 Zone Configuration

In the registry map the zones configuration is held in the Analog Output Holding Registers. They can be read using function 0x03 or modified with 0x10.

Each zone has the following parameters

Zone n id
zone n max enable
zone n min enable
zone n rate enable
zone n relay
zone n zone start
zone n zone stop
zone n max_temp level
zone n min_temp level
zone n rate_level

The max_temp level, min_temp level, and rate_level all define a temperature threshold. When reading these temperature thresholds the obtained values must be multiplied by 0.01 to scale to real temperature. Conversely when writing these values to the registry the real temperature thresholds must be divided by 0.01 and rounded to an integer, before saving to the registry table.

5.3.1 Reading Zone Configuration

The following command reads from slave 15 (0x0f), the analog output holding registers with function 0x03, from MB_ZONE_CONFIG_START (0x06). 100 (0x064) registers values are taken.

Address	1 byte	Slave 15	0x0f
Function code	1 byte	Read multiple values	0x03
starting address	2 bytes	MB_ZONE_CONFIG_START	0x0006
Quantity of Registers	2 bytes	100 registers	0x0064
CRC	2 bytes		0xa50e

COMMAND:

0f 03 0006 0064 a50e

REPLY:

0f # header
03 # function
c8 # number of bytes
0000 0001 0001 0000 0003 000c 0025 1f40 03e8 ff38 #zone 0
0001 0001 0001 0001 0003 002d 0046 1388 03e8 0096
0002 0001 0001 0001 0003 004d 0066 1388 03e8 0096
0003 0001 0001 0001 0003 0066 007f 1388 03e8 0096
0004 0000 0001 0001 0004 00c8 00e1 1388 03e8 0096
0005 0000 0001 0001 0005 00fa 0113 1388 03e8 0096
0006 0001 0001 0001 0006 012c 0145 1388 03e8 0096
0007 0001 0001 0001 0007 015e 0177 1388 03e8 0096
0008 0000 0001 0000 0003 0190 01a9 07d0 03e8 00c8
0009 0001 0001 0000 0004 01c2 01db 1388 07d0 012c #zone 9

5.3.2 Data Processing

Here we will process the raw data to view the parameters stored in zone 0.

	Id	Max enabled	Min enabled	Rate enabled	Relay	Zone start	Zone stop	Max temp level	Min temp level	Rate level
Raw val	0000	0001	0001	0000	0003	000c	0025	1f40	03e8	ff38
	0	1	1	0	3	+12	+37	+8000	+1000	-200
Scale								x0.01	x0.01	x0.01
Final	Zone 0	Yes	Yes	No	3	+12m	+37m	+80.00°C	+10.00°C	-2.00°C

5.4 Modifying Zones

This comes in two parts

1. Update the values in the registry map for example maximum temperature level, zone position etc.
2. Choose whether to save to Flash to make the values permanent after a power cycle.

5.4.1 Zone Modification

The following command changes zone 0. In the registry zone 0 starts from MB_ZONE_CONFIG_START (0x06). The data is written sequentially into the registry table starting at the start address.

Address	1 byte	Slave 15	0x0f
Function code	1 byte	Write multiple values	0x10
Starting Address	2 bytes	MB_ZONE_CONFIG_START	0x0006
Quantity of Registers	2 bytes	10 registers	0x000a
Byte count	1 byte	Bytes count = 10 x 2	0x14
Data0	2 bytes	zone n id (fixed)	0x0000
Data1	2 bytes	zone n max enable	0x0001
Data2	2 bytes	zone n min enable	0x0001
Data3	2 bytes	zone n rate enable	0x0001
Data4	2 bytes	zone n relay - 4	0x0004
Data5	2 bytes	zone n zone start +10m	0x000a
Data6	2 bytes	zone n zone stop +40m	0x0028
Data7	2 bytes	zone n max_temp level +75.00ºC	0x1d4c
Data8	2 bytes	zone n min_temp level -5.00ºC	0xfe0c
Data9	2 bytes	zone n rate_level+2.00ºC	0x00c8
CRC	2 bytes		0xf665

COMMAND:

0f 10 0006 000a 14 0000 0001 0001 0001 0004 000a 0028 1d4c fe0c 00c8 f665

REPLY:

0f 10 0006 000a a121

The reply acknowledges the slave address (15), function code (0x10), and the number of registers (10) that have been modified

5.4.2 Permanent data save to flash

Writing to MB_SAVE_ZONES (0x06) in the Discrete Output Coils using function 0x05 will commit the zone configuration (zones 0 to zone 9) to permanent flash memory. In the event of a power cycle the DTS will load the zone configuration (and other configurations) from flash memory. Therefore it is advisable to do this after any zone modification.

Address	1 byte	Slave 15	0x0f
Function code	1 byte	Write bits	0x05
Output address	2 bytes	MB_SAVE_ZONES	0x0006
Output Value	2 bytes	0xff0 set bit	0xff00
CRC	2 bytes		0x6d15

COMMAND:

0f 05 0006 ff00 6d15 #ff00 indicates that we wish to set

REPLY:

0f 05 0006 ff00 6d15

The reply is the same as the sent command and confirms command has been received

5.5 Zone Status

In the registry map the zone status, i.e. the zone temperature information, is held in the analog input registers and can be read using function 0x04. Each zone has the following parameters, and this is repeated for 10 zones.

Zone n id
Zone n average value
Zone n max temp
Zone n max temp pos
Zone n min temp
Zone n min temp pos
Zone n temperature rate
Zone n max rate pos

The following command reads from slave 15 (0x0f), the analog input registers with function 0x04, from MB_ZONE_INFO_START (0x13). 80 (0x50) registers values are taken.

Address	1 byte	Slave 15	0x0f

Function code	1 byte		0x04
start address	2 bytes	MB_ZONE_INFO_START	0x0013
Num registers	2 bytes	80	0x0050
CRC	2 bytes		0x00dd

COMMAND:

0f04 0013 0050 00DD

REPLY:

0f # slave
04 # function code
a0 # number of bytes
0000 1b89 1bbf 000c 1b5a 0021 ffe2 0011 # zone 0 data
0001 0fb6 0fda 003f 0f7d 0031 0013 002d
0002 0a61 0a8c 004d 0a0f 0062 000d 0052
0003 09f7 0a7b 0065 0972 0070 0012 006e
0004 09e2 0a11 00c7 09b5 00d0 0021 00c8
0005 09ee 0a21 0111 09c0 010b 000f 00fc
0006 09ed 0a1d 0142 09a9 013f 0028 013c
0007 09e1 0a0d 0162 09a5 016f 0013 0173
0008 09e3 0a3c 01a2 09b4 0190 002b 01a2
0009 09ec 0a2f 01c3 09a4 01c6 001d 01ca # zone 9 data
7ffc

5.5.1 Data Conversion Example

The line of data after the header information is converted in this example. The temperature related values must be converted to a signed decimal and then scaled by 0.01 to get the actual temperature values. The id and position values are obtained directly from the registry values and don't require scaling.

	Id	Average value	Max temp	Max temp pos	Min temp	Min temp pos	Max Temp rate	Max rate pos
Raw value	0x0000	0x1b89	0x1bbf	0x000c	0x1b5a	0x0021	0xffe2	0x0011
type	Unsign	Signed	Signed	Signed	Signed	Signed	Signed	Signed
Intermediate calc	0	+7049	+7103	+12	+7002	+33	-30	+17
Scale	None	x0.01	x0.01	none	x0.01	none	x0.01	none
Final	0	70.49ºC	71.03ºC	12m	70.02º	33m	-0.30ºC	17m

5.6 Alarm Status

In the Input Coils table we can read the alarm status of each zone, for example we can see if a maximum temperature has been triggered in a particular zone. Each zone has the following alarm status parameters and this is repeated for 10 zones giving a total of 40 registers.

zone 0 alarm status	General state of a zone
zone 0 max alarm status	Maximum alarm status
zone 0 min alarm status	Minimum alarm status
zone 0 rate alarm status	Rate alarm status

The following command reads from slave 15 (0x0f), all the alarm status registers from MB_STATUS_START to MB_STATUS_END, 40 registers.

Address	1 byte	Slave 15	0x0f
Function code	1 byte		0x02
start address	2 bytes	MB_STATUS_START	0x000d
Quantity of inputs	2 bytes	40	0x0028
CRC	2 bytes		0x60f9

COMMAND:

0f 02 000d 0028 e8f9

REPLY:

0f 02 05 0009000000 2042

5.6.1 Data Conversion Example

hex	00	09	00	00	00
bits	0000 0000	0000 1001	0000 0000	0000 0000	0000 0000

From the bit expansion, zone 3 has the general alarm triggered and a rate alarm.

The discrete inputs in the response message are packed as one input per bit of the data field. The number of bytes is padded to the next whole 8 bits, in order to fill a whole byte.

5.7 Reset Alarms

TBD

6 Intermediate Operations

The following intermediate functions deal with

Reading and writing to the real time clock

Reading and modifying the measurement parameters

6.1 Time

The time is located in the analog output holding registers and can be read using function 0x03 and written to using function codes 0x06 or 0x16.

year		uint16	120	78	MB_RTC_YEAR
month		uint16	121	79	MB_RTC_MONTH
date		uint16	122	7A	MB_RTC_DATE
hours		uint16	123	7B	MB_RTC_HOURS
minutes		uint16	124	7C	MB_RTC_MINUTES
seconds		uint16	125	7D	MB_RTC_SECONDS

6.2 Read

Address	1 byte		0x0f
Function code	1 byte		0x03
starting address	2 bytes	MB_RTC_YEAR	0x0078
Quantity of Registers	2 bytes	6 registers	0x0006
CRC	2 bytes		0x44ff

COMMAND:

0f 03 0078 0006 44ff

REPLY:

0f 03 0c # slave, function code, number of registers
07e0 0008 0003 000d 000f 0038 #yr, mon, date, hr, mn, sec
f7ac # crc

6.3 Data Conversion Example

	Year	Month	Date	Hour	Minute	seconds
Raw val	07e0	0008	0003	000d	000f	0038
Final	2016	August	3	13	15	56

6.4 Write to Real Time Clock (RTC)

Setting the time is a two function process. First we write the values to the registry map and then we need to save the time (MB_SAVE_TIME). The second action will write the new values permanently to the RTC.

Address	1 byte		0x0f
Function code	1 byte	Write multiple	0x10
Starting Address	2 bytes	MB_RTC_YEAR	0x0078
Quantity of Registers	2 bytes	6 registers	0x0006
Byte count	1 byte	Bytes count = 6 x 2	0x0c
Year	2 bytes	2016	0x07e0
month	2 bytes	August (8)	0x0008
date	2 bytes	8	0x0008
hours	2 bytes	13 (24 hour clock)	0x000d
minutes	2 bytes	00	0x0000
seconds	2 bytes	00	0x0000
CRC	2 bytes		0x23d5

COMMAND:

0f 10 0078 0006 0c 07e0 0008 0008 000d 0000 0000 23d5

REPLY:

0f 10 0078 0006 c13c

6.5 Save to RTC

The second action here will write the new values permanently to the RTC

Address	1 byte		0x0f

Function code	1 byte		0x05
Output address	2 bytes	MB_SAVE_TIME	0x000a
Output Value	2 bytes		0xff00
CRC	2 bytes		0xad16

6.6 Measurement Parameters

The measurement parameters are grouped into the following 4 registry entries.

uint16	1		MB_AVERAGING_TIME	seconds
uint16	2		MB_MEASUREMENT_TIME	seconds
uint16	3		MB_SLOPE_ADJUST	
uint16	4		MB_OFFSET_ADJUST	

6.6.1 Reading Measurement Parameters

COMMAND:

0f 03 0001 0004

REPLY:

0f 03 08 # slave, function code, bytes to expect (8)
0a8c 0a8c 0064 00fa
bc61

	Averaging time	Measurement time	Slope adjust	Offset adjust
Raw val	0a8c	0a8c	0064	00fa
	2700	2700	100	+250
Scale factor			0.01	0.01
	45minutes	45minutes	1.0	+2.50ºC

6.6.2 Writing Measurement Parameters

Once any of the four items have been modified these new values are used immediately even if the system is in a current measurement. A bit write to MB_SAVE_M_PARMATERS ensures that the values are saved into FLASH.

Setting the measurement parameters is a two function process. First we write the values to the registry map and then we need to save the time (MB_SAVE_M_PARAMETERS). The second action,

which is optional, will write the new values permanently to the flash memory. If the values are saved to flash memory these values will be used after a power cycle.

Address	1 byte	Slave 15	0x0f
Function code	1 byte	Write multiple	0x10
Starting Address	2 bytes	MB_AVERAGING_TIME	0x0001
Quantity of Registers	2 bytes	4 registers	0x0004
Byte count	1 byte	Bytes count = 4 x 2	0x08
Averaging time	2 bytes	5 seconds	0x0005
Measure time	2 bytes	10 seconds	0x000a
slope	2 bytes	1.0 (/0.01)	0x0064
offset	2 bytes	2.5ºC (/0.01)	0x00fa
CRC	2 bytes		0xc920

COMMAND:

0f 10 0001 0004 08 0005 000a 0064 00fa c920

Reply

0f 10 0001 0004 9124

6.6.3 To Save Measurement Parameters To Flash

MB_SAVE_M_PARAMETERS (0x07) must be set

COMMAND:

0f 05 0007 ff00 3cd5

REPLY:

0f 05 0007 ff00 3cd5

7 Advanced Operations

The following operation require several steps to achieve their intended function.

7.1 Historical Data

Here we extract the saved data from the SD card and present it on the MODBUS registry. Once on the registry the data can be read back using the previously described method. The following process allows the existing registry locations to be used.

- System acquisition must be stopped
- Download current sector details (start, end, total size), start and end are the current position of the first data acquisition and the current (end) location.
    - MB_SD_SECTOR_SIZE_LSB (6 values)
- Each sector is 512 bytes and it take 10 sectors to write all the temperature and alarm information. Once we know the start and we can write a sectors value to MB_SD_SECTOR_LSB MB_SD_SECTOR_MSB. The sector size on the SD card is 32bit. (Access units using V1.1 of the registry map have a difference sector size please contact Bandweaver for exact details)
- Next step is to Upload. This will copy the old SD data into the current registry location. This process will take a 60milliseconds to complete
    - MB_UPLOAD_HISTORICAL cmd 5
- Read the registry locations as normal. The trace date (MB_TRACE_YEAR to MB_TRACE_SECONDS) should also be recorded so that the data has a timestamp

7.2 System Diagnostics

7.2.1 System Information

Reading Event Information
Event data is located between sectors 1024L and 2047L

1. First write the sector value to MB_SD_SECTOR_LSB and MB_SD_SECTOR_MSB. Although the sector location can fit in 16bits, 32bit unsigned integer is needed.

2. Write the bit to copy the SD data to the temperature data structure (MB_TEMPERATURE_START)

3. read the ifirst 12registers from the table starting from MB_TEPMERATURE_START.

The data must be handled as a structure

Uint16_t	event id
Uint8_t	seconds
Uint8_t	minutes
Uint8_t	hours
Uint8_t	date

Uint8_t	month
Uint8_t	current year - 2000
Uint8_t	event type
Int16_t	data1
Int16_t	data2
Uint8_t	data3
Uint32_t	measurement id
Uint32_t	sector
Uint16_t	crc

7.2.2 Modifying Core Parameters

The core parameters affect the way the temperature is calculated and therefore care has to be taken not to enter invalid values.

To modify any parameters MB_C_PARAM_PASSWORD needs to be entered

The core parameters can be changed up simply uploading new values to

MB_GAMMA .. to MB_REFRACTIVE_INDEX.

MB_SAVE_C_PARAMETERS must then be used to save the value (function 0x05).

7.3 Files Modes

Commands 0x14 Read File Record allows for reading data without the restriction of the Modbus memory map. The data transfer is treated as a binary format as such the data structures relate to the internal c structures of the embedded system.

The file number determines the data type. Each file can contain upto 10000 records 16bit 20000byte of data. The amount of data transferable in a Modbus response is still limited to the MODBUS PDU 253bytes. Multiple reads will be required to retrieve some information e.g. distributed temperature.

7.4 Modbus Exception Handling

The section refers to how the system responds to incorrect data and commands.

In a normal response, the server echoes the function code of the original request. In an exception the server sets the most significant bit of the function code to 1. This makes the function code value in an exception response eactly 80 hexadecimals higher than the value in a normal response.

In general, an exception response consists of

Address	1 byte	Slave 15
Function code	1 byte	Function code + 0x80
Exception code	1 bytes	0x01 to 0x0B
crc	2 bytes	

Access DTS supports error codes 01 to 04; the reader is advised to consult the MODBUS Application Protocol Specification for more details

Code	Name	Meaning
01	ILLEGAL FUNCTION	function code is not implemented in Access DTS
02	ILLEGAL DATA ADDRESS	data address is not in the allowable address range of the DTS
03	ILLEGAL DATA VALUE	indicates a fault in the structure of the request, it does not indicate the data item is outside the expectation of the application program. Often the date length of the request is incorrect
04	SLAVE DEVICE FAILURE	Unrecoverable error occurred while performing the requested action. Inside the Modbus code an error was raised

7.4.1 Illegal Function test

In this test we send a function code 0x11 which has no function.

COMMAND:
0F 11 000 0001 3D27
REPLY:
0f 91 01 ed93

7.4.2 Illegal data address test

In this test we try to read values beyond the end of the registry table for function code 0x02. The last item is at 0x0043. However, we tried to read 2 (0x0002) samples.

COMMAND:
0f 02 0043 0002 0931
REPLY:
0f 82 02 a0a2

7.4.3 Illegal data value test – incorrect data length

In this test we attempt to download 127 (0x007F) registers. However, the max. size is 125 (0x007D).

COMMAND:
0f 04 003b 007f c0f0
REPLY:
0f 84 03 62c2

7.4.4 Slave Device Failure Test

currently only used if the password to change the core parameters is incorrect

COMMAND:
0f 06 006a 0000 a8f8
REPLY:
0f 86 04 2260

8 Application Development

So, you want to write your own application around Access hardware? The following flow charts outline steps to interrogate the system.

8.1 Basic Operation

8.1.1 Reading Temperature Data 1

Slave id of a single machine is already known, and the machine is powered on

Code Outline

1. Start
2. Check Slave id is correct 0F 03 0080 0001 84CC
3. Read temperature data
    a. 125 registers 0F 04 0074 007D 711F
    b. Wait ~20ms
    c. 125 registers 0F 04 00F1 007D 60F6
    d. Wait ~20ms
    e. 125 registers 0F 04 016E 007D 5124
    f. Wait ~20ms
    g. 50 registers 0F 04 01EB 0032 0139
4. Multiply temperature data by 0.01
5. Wait 5s
6. Goto 3

8.1.2 Database Structure

There is one problem with the previous example in that we don't know:

1. The number of registers to download (MB_BLOCKSIZE)
2. Start point (MB_X0)
3. The spacing between each reading (MB_SAMPLE_LENGTH)

MB_X0 is a signed integer and is generally always a negative number (e.g. -41.75m) The registry value will be in 2's complement notation. Subtract 65536 (2^16) to get the desired number and then multiply by 0.01.

Once the blocksize is known the total number of temperature data registers required can be adjusted.

These three values are defined in three consecutive registers and so only one command needs to be sent. Without these three values we cannot view the data on the graph correctly.

Code Outline

1. Start
7. Check slave id is correct 0F 03 0080 0001 84CC
8. Read blocksize, x0, and sample length 0F 04 0071 0003 E13E
2. Scale values

   a. Blocksize no change (e.g. 0x244 = 580)
   b. x0 * 0.01 (e.g. 0xEFB1 → (65536 – 61361) * 0.01 = -41.75m)
   c. sample length * 0.001 (e.g. 0x03f8 -> 1016 * 0.001 = 1.016m)
3. read temperature data, as before
   a. scale temperature data by 0.01
4. wait 5s
5. goto 5

8.2 Zone id

Average temperature of the zone

Maximum temperature in the zone

Max temperature position

Minimum temperature in the zone

Min temperature position

Maximum rate in the zone

Max rate position

Apart from the zone id each value is a signed value, so these registry values will be in 2's complement notation. You will need to convert these values in order to be handled correctly.

There are 10 zones and each zone have 8 associated registers. This means we can read all the zone information with one command (don't forget the max number of registers that can be read at once is 125).

8.3 Build > 3xx

Finer distance resolution for zone temperature positions.

8.4 CRC

It is highly recommended to use the CRC Generation Function from MODBUS over serial line specification and implementation guide V1.02. The function code is written in C but can be easily converted to other software languages. The function code uses a look up table to perform the CRC calculation and is by far the most efficient method.