

# **ET-M8194H**

## **Motion Control Module**

### **User Manual**

( Version 1.0 )



**ICP DAS CO., LTD.**

---

## Warranty

All products manufactured by ICPDAS Inc. are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

## Warning

ICPDAS Inc. assumes no liability for damages consequent to the use of this product. ICPDAS Inc. reserves the right to change this manual at any time without notice. The information furnished by ICPDAS Inc. is believed to be accurate and reliable. However, no responsibility is assumed by ICPDAS Inc. for its use, or for any infringements of patents or other rights of third parties resulting from its use.

## Copyright

Copyright 1997-2007 by ICPDAS Inc., LTD. All rights reserved worldwide.

## Trademark

The names used for identification only maybe registered trademarks of their respective companies.

## License

The user can use, modify and backup this software on a single machine. The user may not reproduce, transfer or distribute this software, or any copy, in whole or in part.

<b>1. PREFACE .....</b>	<b>6</b>
1.1 Introduction.....	6
1.2 ET-M8194H Features .....	0
1.3 Function description .....	0
1.4 Function categories description.....	0
<b>2. BASIC SETTINGS.....</b>	<b>5</b>
2.1 Axes Code Definition .....	5
2.2 Reset the Motion Module .....	0
2.3 Pules Output Mode Setting .....	0
2.4 Set the Maximum Speed.....	0
2.5 Setting the Active Level of Hardware Limit Switches.....	0
2.6 Setting the Motion Stop Mode when Limit Switch being turn on.....	0
2.7 Setting the Trigger Level of the NHOME Sensor .....	0
2.8 Setting the Trigger Level of the HOME Sensor .....	0
2.9 Setting and Clearing the Software Limit.....	0
2.10 Setting the Encoder Related Parameters.....	0
2.11 Setting the Servo Driver (ON/OFF) .....	0
2.12 Setting the SERVO ALARM Function .....	0
2.13 Setting the Active Level of the In-Position Signals .....	1
2.14 Setting the Time Constant of Digital Filter .....	2
2.15 Position Counter Variable Ring.....	0
2.16 Triangle prevention of fixed pulse driving .....	0
2.18 External Pulse Input.....	0
2.18.1 Handwheel (Manual Pulse Generator) Driving .....	0
2.18.2 Fixed Pulse Driving Mode .....	0
2.18.3 Continuous Pulse Driving Mode.....	1
2.18.4 Disabling the External Signal Input Functions.....	0
2.19 Read/Write User defined Variables (VAR).....	0
2.20 Read/Write Data for Power outage carry-over (MD) .....	0
<b>3. READING AND REGISTER SETTING .....</b>	<b>5</b>
3.1 Setting and Reading Command Position.....	5
3.2 Setting and Reading the Encoder Counter.....	0
3.3 Reading the Current Velocity .....	0
3.4 Reading the Current Acceleration.....	1
3.5 Reading the DI Status .....	2

3.6 Reading and Clearing the ERROR Status .....	0
<b>4. FRNET FUNCTIONS.....</b>	<b>4</b>
4.1 Read FRnet DI Signals .....	4
4.2 Write data to FRnet DO .....	0
4.3 Read back FRnet DO Signals .....	2
<b>5. AUTO HOMING SEARCH .....</b>	<b>3</b>
5.1 Set Up Homing Speed .....	3
5.2 Using an Limit Switch as the Home sensor .....	0
5.3 Setting the Homing Mode.....	0
5.4 Starting the Homing Sequence .....	0
<b>6. BASIC MOTION CONTROL .....</b>	<b>1</b>
6.1 Independent Motion Control for each axis .....	1
6.1.1 Setting the Acceleration/Deceleration Mode .....	1
6.1.2 Setting the Starting Speed .....	0
6.1.3 Setting the Desirted Speed .....	0
6.1.4 Setting the Acceleration .....	0
6.1.5 Setting the Deceleration .....	0
6.1.6 Setting the Acceleration Rate .....	0
6.1.7 Setting the Deceleration Rate .....	0
6.1.8 Setting the Value of the Remaining Offset of Pulses.....	0
6.1.9 Fixed Pulse Output .....	0
6.1.10 Continue Pulse Output .....	0
6.2 Interpolation Commands .....	0
6.2.1 Assigning the Axes for Interpolation.....	0
6.2.2 Setting the Speed and Acc/Dec Mode .....	0
6.2.3 Setting the Vector Starting Speed .....	0
6.2.4 Setting the Vector Speed.....	0
6.2.5 Setting the Vector Acceleration .....	0
6.2.6 Setting the Vector Deceleration Value.....	0
6.2.7 Setting the Vector Acceleration Rate .....	0
6.2.8 Setting the Vector Deceleration Rate .....	0
6.2.9 Setting the Number of Remaining Offset Pulses .....	0
6.2.10 2-Axis Interpolation Motion.....	0
6.2.11 3-Axis Interpolation Motion .....	0
6.2.12 2-Axis Circular Interpolation Motion (an Arc).....	0

6.2.13 2-Axis Circular Interpolation Motion (an Complete Circle) .....	0
<b>6.3 Synchronous Actions.....</b>	<b>2</b>
6.3.1 Setting the Synchronous Action.....	2
6.3.2 Setting the COMPARE Value .....	0
6.3.3 Get the LATCH Value .....	0
6.3.4 Set the PRESET data for synchronous acton.....	0
6.3.5 Set the OUT Data .....	0
6.3.6 The nterrupt Function of motion module (i-8094H) .....	0
6.3.7 The Interrupt of i-8094H for controller system .....	0
<b>6.4 Continuous Interpolation .....</b>	<b>4</b>
6.4.1 2-Axis Rectangular Motion .....	4
6.4.2 2-Axis Continuous Linear Interpolation .....	0
6.4.3 3-Axis Continuous Linear Interpolation .....	0
6.4.4 Mixed Linear and Circular 2-axis motions in Continuous Interpolation .....	0
6.4.5 3-Axis Helical Motion .....	0
6.4.6 2-Axis Ration Motion .....	0
6.4.7 Synchronous Line Scan Motion .....	0
<b>6.5 Other Functions .....</b>	<b>6</b>
6.5.1 Holding theDriving Command .....	6
6.5.2 Release the Holding Status and Start the Driving.....	0
6.5.3 Stopping the Axes.....	0
6.5.4 Clear the Stop Status .....	0
6.5.5 End of Interpolation .....	0

## **7. ADDITIONAL FUNCTIONS SUPPORTED BY I8094H**

.....	<b>1</b>
<b>7.1 Initial Parameter Table .....</b>	<b>0</b>
<b>7.2 Create Macro Program (MP).....</b>	<b>0</b>
7.2.1 Create Macro Program Codes.....	0
7.2.2 Execute Maro Program (MP) .....	0
7.2.3 User Defined Variables .....	0
7.2.4 Command Loop (FOR~NEXT) .....	0
7.2.5 Condition Command (IF~ELSE).....	0
7.2.6 TIMER .....	0
7.2.7 Wait Motion Stop(For MP Only) .....	0
7.2.8 User-defined RINT .....	0

## **8. ET-M8194H API..... 0**

<b>8.1 API Function Types:</b>	<b>0</b>
<b>8.2 API Function Descriptions:</b>	<b>0</b>
8.2.1 Communication Functions:	0
8.2.2 MODBUS Functoin Code 16	2
8.2.3 MODBUS Functoin Code 04	0
8.2.4 MODBUS Functoin Code 03	0
8.2.5 MODBUS Functoin Code 01	1
8.2.6 MODBUS Functoin Code 02	1
8.2.7 MODBUS Functoin Code 05	0
8.2.8 MODBUS Functoin Code 15	0
8.2.9 Other Functions	0
<b>8.3 ErrorCode:</b>	<b>0</b>
<b>APPENDIX</b>	<b>1</b>
<b>A. Input Registers</b>	<b>1</b>
<b>B. Holding Registers (the base address is staring from 0)</b>	<b>0</b>
<b>C. Sub Function Code mapping table</b>	<b>0</b>
<b>D. FRnet DI/O mapping MODBUS Address (FC01, 02, 05, 15)</b>	<b>0</b>
<b>E. ET-M8194H LED Description</b>	<b>0</b>
<b>F. Lock_IP Setting</b>	<b>0</b>
<b>G. IP Configuration</b>	<b>0</b>
<b>H. Update Firmware</b>	<b>0</b>
<b>I. History of Versions</b>	<b>0</b>

---

# 1. Preface

---

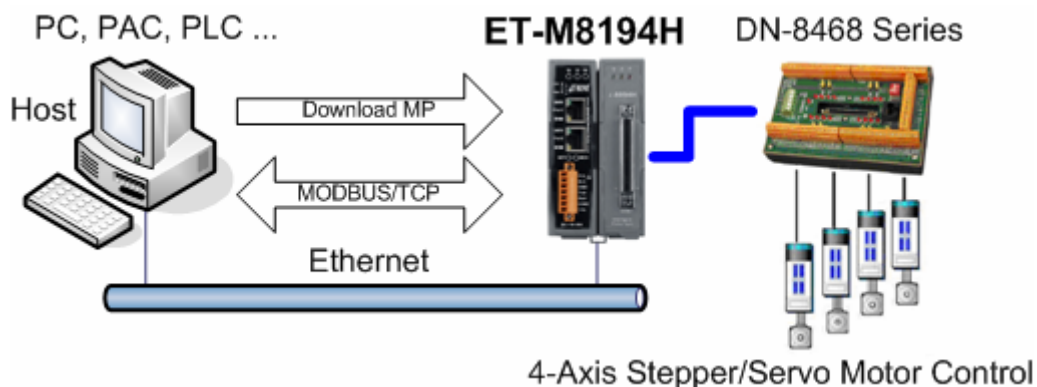
## 1.1 Introduction

- This manual includes eight chapters and nine appendices. This chapter gives a brief description of this manual. The Chapter 2 to 7 provide the MODBUS command of the relevant ET-M8194H API; and the Chapter 8 is category of ET-M8194H API. The Appendices include the definitions of MODBUS address, Sub Function Code of ET-M8194H API, the LED of ET-M8194H and Lock\_IP feature.
- This documentation helps to control the ET-M8194H via MODBUS/TCP. Most functions are implemented with the FC=16, start\_address = 8000. Other state-related functions use FC=03, 04, 16, 01, 02, 05 and 15 to access ET-M8194H.

## 1.2 ET-M8194H Features

ET-M8194H is a new product of ICP DAS to implement the Ethernet remote motion controller for the remote motion solution. It includes an i-8094H module (a 4-axis stepping/pulse-type servo motor control module). By using the intelligent function of ET-M8194H, we can reach the remote motion control to finish various motion applications via MODBUS/TCP.

Also, ET-M8194H can be applied in many platforms with MODBUS/TCP protocol (for example: PC, PAC, PLC). Additionally, it is easy and simple to use many ET-M8194Hs at the same time to implement the multi-axis motion control by the serial connection of ET-M8194H. We also provide EzMove Utility and API Library for users to quickly configure ET-M8194H and develop their own control applications easily.





## 1.3 Function description

- **Function\_name( parameter1, parameter2 .....)**

**Description:** Explanation of this function.

**Category:** Function categories description.

**Parameters:** Definitions of the parameters and how to use them.

**Return:** The return value of this function.

**Example:** Simple example program.

**Remark:** Comments.

### MODBUS Register Table:

For instance, the commands of MODBUS are defined as follows:

(1) **TID** : Transaction ID.

(2) **PID** : Protocol ID.

(3) **Field Length** : Length of data-field (in Byte). The data-field begins from UID and end at last Register.

(4) **UID** : unit ID (CardNo ).

(5) **FC** : Function Code.

(6) **St\_Addr.** : Starting Address.

(7) **Word Count** : The length of the following Registers (in Word, 16-bit).

(8) **Byte Count** : The length of the following Registers (in Byte, 8-bit).

(9) **Register** : Content of parameter (16-bit).

(9-1) **Sub\_Function Code**: The pre-defined code for each ET-M8194H function.

(9-2) **Axis**: The target Axis/Axes of the that ET-M8194H function.

For each function, the the contents of each field and the required Registers and will be listed after function description. The related Registers only support 16-bit data. Therefore, two Registers will be involved for the 32-bit parameters, high-word (MSW) and then low-word (LSW).

For instance, the *Sub\_Function Code* of ETM\_SET\_LP() is 0x0A28, and one 32-bit value will be involved. The contents of MODBUS Register Table will be listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 28		<i>Sub_funciton code</i>			
1		00 0F		<i>axis (F → AXIS_XYZU)</i>			
2		00 00		<i>MSW of wdata</i>			
3		00 00		<i>LSW of wdata</i>			

## 1.4 Function categories description

**RTC** (Real Time Command): Enable i8094H to execute real time command

**MP** (Macro Program): Functions to be executed after MP\_CREATE.

**ISR** (Interrupt Service Routine): Functions to be executed in ISR after MP\_CREATE.

**IT** (Initial Table): Functions to be executed in parameters table.

Maximum number of Function Line for ISR1 ~ ISR20 and MP1 ~ MP157::

ISR(6)	ISR1	ISR2	ISR3	ISR4	ISR5	ISR6			
Total:	8	8	8	8	8	8			
ISR(9)	ISR7	ISR8	ISR9	ISR10	ISR11	ISR12	ISR13	ISR14	ISR15
Total:	16	16	16	16	16	16	16	16	16
ISR(3)	ISR16	ISR17	ISR18						
Total:	32	32	32						
ISR(2)	ISR19	ISR20							
Total:	64	64							
MP(40)	MP1	~	MP40						
Total:	8		8						
MP(50)	MP41	~	MP90						
Total:	16		16						
MP(40)	MP91	~	MP130						
Total:	32		32						
MP(20)	MP131	~	MP150						
Total:	64		64						
MP(5)	MP151	MP152	MP153	MP154	MP155				
Total:	128	128	128	128	128				
MP(2)	MP156	MP157							
Total:	512	512							

In the following Function Table, most functions in sections 2, 3, 4, 5 and 6 could be used in i8094H\_MP\_CREATE (Please refer to Section 7.2.1), all values could be replaced by variables (when applied to MP or ISP).

bvarNo: User-defined variables: bVAR0 ~ bVAR127 (Data type :**BYTE**)

varNo: User-defined variables: VAR0 ~ VAR511 (Data type :**long**or**DWORD**)

**Note:** In the following sections \* indicates functions applied to MP.

In the following sections \* indicates functions applied to ISR.

The mark ◎ indicates the function is available in that category.

The mark ◎x2 means the function is available in that category with

two MP function lines.

Basic settings					
Section	Function	RTC	MP	ISR	IT
2.3	ETM_RESET_CARD				
2.3	ETM_CLEAR_CARD_BUFFER				
2.4	ETM_SET_PULSE_MODE				
2.5	ETM_SET_MAX_V				
2.6	ETM_SET_HLMT				
2.7	ETM_LIMITSTOP_MODE				
2.8	ETM_SET_NHOME				
2.9	ETM_SET_HOME_EDGE				
2.10	ETM_SET_SLMT				
2.10	ETM_CLEAR_SLMT				
2.11	ETM_SET_ENCODER				
2.12	ETM_SERVO_ON				
2.12	ETM_SERVO_OFF				
2.13	ETM_SET_ALARM				
2.14	ETM_SET_INPOS				
2.15	ETM_SET_FILTER				
2.16	ETM_VRING_ENABLE				
2.16	ETM_VRING_DISABLE				
2.17	ETM_AVTRI_ENABLE				
2.17	ETM_AVTRI_DISABLE				
2.18	ETM_EXD_MP				
2.18	ETM_EXD_FP				
2.18	ETM_EXD_CP				
2.18	ETM_EXD_DISABLE				
2.19	ETM03_READ_bVAR				
2.19	ETM_WRITE_bVAR				
2.19	ETM03_READ_VAR				
2.19	ETM_WRITE_VAR				
2.20	ETM03_READ_MD				
2.20	ETM_WRITE_MD				
Status Reading and Setting					
3.1	ETM_SET_LP				
3.1	ETM_GET_LP				
	ETM03_GET_LP				

3.2	ETM_SET_EP				
Section	Function	RTC	MP	ISR	IT
3.2	ETM_GET_EP				
	ETM03_GET_EP				
3.3	ETM03_GET_CV				
3.4	ETM03_GET_CA				
3.5	ETM_GET_DI				
	ETM04_GET_DI				
3.5	ETM_GET_DI_ALL				
	ETM04_GET_DI_ALL				
3.6	ETM_GET_ERROR				
	ETM04_GET_ERROR_STATE				
3.6	ETM_GET_ERROR_CODE				
	ETM04_GET_ERROR_CODE				
3.7	ETM04_GET_FREE_BUFFER				
FRnet Function					
4.1	ETM_FRNET_IN				
	ETM04_GET_FRNET_GPDI				
4.2	ETM_FRNET_OUT				
4.3	ETM03_GET_FRNET_GPDO				
Home Search					
5.1	ETM_SET_HV				
5.2	ETM_HOME_LIMIT				
5.3	ETM_SET_HOME_MODE				
5.4	ETM_HOME_START				
Independent Axis Motion Control					
6.1.1	ETM_NORMAL_SPEED				
6.1.2	ETM_SET_SV				
6.1.3	ETM_SET_V				
6.1.4	ETM_SET_A				
6.1.5	ETM_SET_D				
6.1.6	ETM_SET_K				
6.1.7	ETM_SET_L				
6.1.8	ETM_SET_AO				
6.1.9	ETM_FIXED_MOVE				
6.1.9	ETM_SET_PULSE				
6.1.10	ETM_CONTINUE_MOVE				
Interpolation Motion					

6.2.1	ETM_AXIS_ASSIGN				
<b>Section</b>	<b>Function</b>	<b>RTC</b>	<b>MP</b>	<b>ISR</b>	<b>IT</b>
6.2.2	ETM_VECTOR_SPEED				
6.2.3	ETM_SET_VSV				
6.2.4	ETM_SET_VV				
6.2.5	ETM_SET_VA				
6.2.6	ETM_SET_VD				
6.2.7	ETM_SET_VK				
6.2.8	ETM_SET_VL				
6.2.9	ETM_SET_VAO				
6.2.10	ETM_LINE_2D				
6.2.11	ETM_LINE_3D				
6.2.12	ETM_ARC_CW	x2	x2	x2	
6.2.12	ETM_ARC_CCW	x2	x2	x2	
6.2.13	ETM_CIRCLE_CW				
6.2.13	ETM_CIRCLE_CCW				
<b>Synchronous Actions</b>					
6.3.1	ETM_SYNC_ACTION	x2	x2	x2	
6.3.2	ETM_SET_COMPARE				
6.3.3	ETM_GET_LATCH				
6.3.4	ETM_SET_PRESET				
6.3.5	ETM_SET_OUT				
<b>Enable / Disable Interrupt Function</b>					
6.3.6	ETM_ENABLE_INT				
6.3.6	ETM_DISABLE_INT				
6.3.6	ETM_INTFACTOR_ENABLE				
6.3.6	ETM_INTFACTOR_DISABLE				
6.3.7	ETM_ENABLE_RINT				
6.3.7	ETM_DISABLE_RINT				
6.3.7	ETM04_GET_RINT_STATE_ALL				
<b>Continuous Interpolation</b>					
6.4.1	ETM_RECTANGLE	x4	x4		
6.4.2	ETM_LINE_2D_INITIAL	x2	x2		
6.4.2	ETM_LINE_2D_CONTINUE				
6.4.3	ETM_LINE_3D_INITIAL	x2	x2		
6.4.3	ETM_LINE_3D_CONTINUE				
6.4.4	ETM_MIX_2D_INITIAL	x2	x2		
6.4.4	ETM_MIX_2D_CONTINUE	x2	x2		

6.4.5	i8094H_CONTINUE_INTP				
<b>Section</b>	<b>Function</b>	<b>RTC</b>	<b>MP</b>	<b>ISR</b>	<b>IT</b>
6.4.5	I8094H_CONTINUE_INTP_ARRAY				
6.4.6	ETM_HELIX_3D	x3	x3		
6.4.7	ETM_RATIO_INITIAL	x2	x2		
6.4.7	ETM_RATIO_2D				
6.4.8	ETM_LINE_SCAN				
6.4.8	ETM_LINE_SCAN_START				
6.4.8	ETM_LINE_SCAN_OFFSET2				
<b>Other Functions</b>					
6.5.1	ETM_DRV_HOLD				
6.5.2	ETM_DRV_START				
6.5.3	i8094H_STOP_WAIT				
6.5.4	ETM_STOP_SLOWLY				
6.5.4	ETM_STOP_SUDDENLY				
6.5.4	ETM_VSTOP_SLOWLY				
6.5.4	ETM_VSTOP_SUDDENLY				
6.5.5	ETM_CLEAR_STOP				
6.5.6	ETM_INTP_END				
<b>Additional Functions supported by i-8094H</b>					
7.1	ETM_LOAD_INITIAL				
7.2.1	ETM_MP_CREATE				
7.2.1	ETM_MP_CLOSE				
7.2.2	ETM_MP_CALL				
7.2.3	ETM_MP_SET_VAR				
7.2.3	ETM_MP_SET_RVAR				
7.2.3	ETM_MP_VAR_CALCULATE				
7.2.4	ETM_MP_FOR				
7.2.4	ETM_MP_NEXT				
7.2.5	ETM_MP_IF				
7.2.5	ETM_MP_ELSE				
7.2.5	ETM_MP_IF_END				
7.2.6	ETM_MP_TIMER				
7.2.7	ETM_MP_STOP_WAIT				
7.2.8	ETM_MP_SET_RINT				

## 2. Basic Settings

### 2.1 Axes Code Definition

The definition of axis assignments are as below: X=1, Y=2, Z=4, and U=8. If you assign X and Y axes simultaneously, the code will be 3. In a similar way,  $AXIS\_YZ = 2+4 = 0x6$ ; and  $AXIS\_XYZU = 1+2+4+8 = 0xf$ . You could assign single axis as well as multiple axes at the same time. Available axis codes are listed below:

Table 2-1 Axis assignments and their corresponding codes

Axis	X	Y	Z	U	XY	XZ	XU	YZ
Code	0x1	0x2	0x4	0x8	0x3	0x5	0x9	0x6
Variable	AXIS_X	AXIS_Y	AXIS_Z	AXIS_U	AXIS_XY	AXIS_XZ	AXIS_XU	AXIS_YZ
Axis	YU	ZU	XYZ	XYU	XZU	YZU	XYZU	
Code	0xa	0xc	0x7	0xb	0xd	0xe	0xf	
Variable	AXIS_YU	AXIS_ZU	AXIS_XYZ	AXIS_XYU	AXIS_XZU	AXIS_YZU	AXIS_XYZU	

Write the setting values into the IT parameter table without making a change of other current settings (**please refer to Section 7.1**), the definitions are as follow:

Table(2-1a)

Axis	X	Y	Z	U	XY	XZ	XU	YZ
Code	0x11	0x12	0x14	0x18	0x13	0x15	0x19	0x16
Variable	INITIAL_X	INITIAL_Y	INITIAL_Z	INITIAL_U	INITIAL_XY	INITIAL_XZ	INITIAL_XU	INITIAL_YZ
Axis	YU	ZU	XYZ	XYU	XZU	YZU	XYZU	
Code	0x1a	0x1c	0x17	0x1b	0x1d	0x1e	0x1f	
Variable	INITIAL_YU	INITIAL_ZU	INITIAL_XYZ	INITIAL_XYU	INITIAL_XZU	INITIAL_YZU	INITIAL_XYZU	

**Not apply to Macro Program (MP).**



## 2.2 Reset the Motion Module

- **int** ETM\_RESET\_CARD (**SOCKET** s, **BYTE** cardNo)

Description:

This function enables motion module (I8094H) to restore the power-on default settings.

Category:

MODBUS sub\_function; RTC.

Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
cardNo: Module number.

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

```
ETM_RESET_CARD (S, 1); //Reset the module1
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A 0A		Sub_funciton code			

- **int** ETM\_CLEAR\_CARD\_BUFFER (**SOCKET** s, **BYTE** cardNo)

Description:

Clear all data in i-8094H command buffer.

Category:

MODBUS sub\_function; RTC.

Parameters:

s: The SOCKET handle, please refer to the **ETM\_Connect** function in Section [8.2.1](#).

cardNo: Module number

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

ETM\_CLEAR\_CARD\_BUFFER (S, 1); //clear data buffer in module 1

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A 0B		Sub_funciton code			

The other function, ETM04\_GET\_FREE\_BUFFER, gets the block-number of available command-buffer (max. block-number is 30):

**int** ETM04\_GET\_FREE\_BUFFER(**SOCKET** s, **BYTE** cardNo,  
**BYTE\*** FreeBufNum);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	04	00 06	00 01

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Block_Num (hex)
00 01	00 00	00 05	01	04	02	00 1E

**The returned value, 0x1E, means 30 blocks of command-buffer are available.**

## 2.3 Pules Output Mode Setting

- **\*int** ETM\_SET\_PULSE\_MODE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nMode)

### Description:

This function sets the pulse output mode to be either CW/CCW or PULSE/DIR for the specific axes and also sets their direction definition.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)

**nMode:** Assigned mode (Please refer to Table 2-2)

Table 2-2 List of pulse output modes

	mode	Pulse output signa	
		nPP	nPM
CW / CCW	0	CW(risig edge)	CCW(risig edge)
	1	CW(falling edge)	CCW(falling edge)
PULSE / DIR	2	PULSE (rising edge)	DIR (LOW:+dir/ HIGH:-dir)
	3	PULSE (falling edge)	DIR (LOW:+dir/ HIGH:-dir)
	4	PULSE (rising edge)	DIR (HIGH:+dir/ LOW:-dir)
	5	PULSE (falling edge)	DIR (HIGH:+dir/ LOW:-dir)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_PULSE_MODE (s, 1, AXIS_XYZ, 2);  
// set the pulse mode of X, Y, and Z axes to be mode 2 in module 1  
ETM_SET_PULSE_MODE (s, 1, INITIAL_XYZU, 0);  
//set the pulse mode of U axis to be mode 3 in module 1
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 0C		<i>Sub_funciton code</i>			
1		00 07		<i>axis</i>			
2		00 02		<i>nMode</i>			

## 2.4 Set the Maximum Speed

- **\*int** ETM\_SET\_MAX\_V (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

This function sets the maximum rate for the output pulses (speed). A larger value results in a rougher resolution, and vice versa(8000 speed segments). For example, if the maximum speed is set as 8000 PPS, the resolution will be 1 PPS;if the maximum speed is set as 16000 PPS, the resolution will be 2 PPS; if the maximum speed is set as 80000 PPS, the resolution will be 10 PPS, etc. Maximum value 4,000,000 PPS means the resolution of speed will be 500 PPS. This function will change the resolution of speed to reach the desired maximum speed. Since the scale in hardware is changed, other parameters will be updated accordingly too; such as the starting speed, the acceleration, and the jerk. It is recommended to set the maximum speed value to be an integral multiplier of 8000.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**data:** maximum speed, single axis (8,000~4,000,000 PPS)  
interpolation motion maximum speed, the second axis(8,000~2,828,854 PPS) interpolation motion maximum speed, the third axis (8,000~2,309,468 PPS).

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_MAX_V (S, 1, AXIS_XY, 200000L);  
//The maximum speed for the X and Y axes of module 1 is 200KPPS.  
//The resolution of the speed will be 200000/8000 = 25 PPS.
```

The MODBUS command is listed as follows:

TID	PID	Field	UID	FC	St_Addr.	Word	Byte
-----	-----	-------	-----	----	----------	------	------

(hex)	(hex)	Length (hex)	(hex)	(hex)	(hex)	Count (hex)	Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 0D		<i>Sub_funciton code</i>			
1		00 03		<i>axis</i>			
2		00 03		<i>MSW of data</i>			
3		0D 40		<i>LSW of data (200000 = 0x30D40)</i>			

## 2.5 Setting the Active Level of Hardware Limit Switches

- **\*int** ETM\_SET\_HLMT (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nFLEdge, **BYTE** nRLEdge)

### Description:

This function sets the active logic level of the hardware limit switch inputs.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**nFLEdge:** Active level setting for the forward limit switch.  
0 = low active; 1 = high active

**nRLEdge:** Active level setting for the reverse limit switch.  
0 = low active; 1 = high active

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_HLMT (s, 1, AXIS_XYZU, 0, 0);  
//set all the trigger levels as low-active for all limit switches n module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 0E		Sub_funciton code			
1		00 0F		Axis			
2		00 00		nFLEdge			
3		00 00		nRLEdge			



## 2.6 Setting the Motion Stop Mode when Limit Switch being turn on

- **\*int** ETM\_LIMITSTOP\_MODE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nMode)

### Description:

This function configures the settings of motion stop mode of the axes when the corresponding limit switches being turn on.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**nMode:** 0: stop immediately; 1: decelerating to stop

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_LIMITSTOP_MODE (s, 1, AXIS_X, 0);  
// //set X axis to stop immediately if any limit switch on X axis is turned on.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 0F		Sub_funciton code			
1		00 01		axis			
2		00 00		nMode			

## 2.7 Setting the Trigger Level of the NHOME Sensor

- **\*int** ETM\_SET\_NHOME (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nNHEdge)

### Description:

This function enables to set up the trigger level of the near home sensor (NHOME).

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**nNHEdge:** Active level setting for for the near home sensor  
0 = low active; 1 = high active

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_NHOME (s, 1, AXIS_XY, 0);  
// set the trigger level of NHOME of X and Y axes on module 1 to be active low.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A10		Sub_funciton code			
1		00 03		Axis (3 → AXIS_XY)			
2		00 00		nNHEdge			

## 2.8 Setting the Trigger Level of the HOME Sensor

- **\*int** ETM\_SET\_HOME\_EDGE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nHEdge)

### Description:

This function sets the trigger level of the home sensor (HOME)

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**nHEdge:** Active level setting for the home sensor  
0 = low active; 1 = high active

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_HOME_EDGE (s , 1, AXIS_XYZU, 1);  
//set the trigger level as high active for all home sensors on module 1
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 11		Sub_funciton code			
1		00 0F		Axis (F → AXIS_XYZU)			
2		00 01		nHEdge			

## 2.9 Setting and Clearing the Software Limit

- **\*int** ETM\_SET\_SLMT (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long** dwFL, **long** dwRL, **BYTE** nType)

### Description:

This function sets the Positive and Negative software limits.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**dwFL:** Value of the forward software limit  
(-2,000,000,000 ~ +2,000,000,000)

**dwRL:** Value of the reverse software limit  
(-2,000,000,000 ~ +2,000,000,000)

**nType:** Position counter to be compared  
0 = logical position counter (LP), i.e., the command position  
1 = encoder position counter (EP), i.e., the real position

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_SLMT (s, 1, AXIS_XYZU, 20000, -3000, 0);  
//set the forward software limit to be 20000 and the reverse  
//software limit to be -3000 for all axes on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 15	01	10	1F 40	00 07	0E
Register[]		Value (hex)		Remarks			
0		0A 12		Sub_funciton code			
1		00 0F		axis			
2		00 00		MSW of dwFL			

3	4E 20	LSW of dwFL (20000 → 0x4E20)
4	FF FF	MSW of dwRL
5	F4 48	LSW of dwRL (-3000 → 0xFFFFF448)
6	00 00	nType

- **\*int ETM\_CLEAR\_SLMT (SOCKET s, BYTE cardNo, BYTE axis)**

**Description:**

This function clears the software limits.

**Category:**

MODBUS sub\_function; RTC, MP and IT.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_CLEAR_SLMT (s, 1, AXIS_XYZU);
//clear the software limits for all axes on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 13		Sub_funciton code			
1		00 0F		Axis (F → AXIS_XYZU)			

## 2.10 Setting the Encoder Related Parameters

- **\*int** ETM\_SET\_ENCODER (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nMode, **BYTE** nDivision, **BYTE** nZEdge)

### Description:

This function sets the relevant parameters for encoder input.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**nMode:** Encoder input type: 0 = A quad B; 1 = up/down

**nDivision:** Division setting for A quad B input signals:  
0 = 1/1  
1 = 1/2  
2 = 1/4

**nZEdge:** Sets the trigger level for the Z phase  
0 = low active; 1 = high active

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_ENCODER (s, 1, AXIS_XYZU, 0, 0, 0);  
//set the encoder input type as A quad B; the division is module 1; //and the Z  
phase is low active.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			
0		0A 14		Sub_funciton code			
1		00 0F		axis (F → AXIS_XYZU)			
2		00 00		nMode (0 → AB phase)			
3		00 00		nDivision (0 → 1:1)			

4	00 00	<i>nZEdge (0→Z phase is low-active)</i>



## 2.11 Setting the Servo Driver (ON/OFF)

- **\*int** ETM\_SERVO\_ON (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

Description:

This function outputs a DO signal (ENABLE) to enable the motor driver.

Category:

MODBUS sub\_function; RTC, MP and IT.

Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

axis: Axis or axes (Please refer to Table 2-1a)

Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

```
ETM_SERVO_ON (s, 1, AXIS_XYZU);  
//enables all drivers on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 15		Sub_funciton code			
1		00 0F		axis (F → AXIS_XYZU)			

- **\*int ETM\_SERVO\_OFF (SOCKET s, BYTE cardNo, BYTE axis)**

**Description:**

This function outputs a DO signal (ENABLE) to disable the motor driver.

**Category:**

MODBUS sub\_function; RTC, MP and IT.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_SERVO_OFF (s, 1, AXIS_XYZU);
//disables all drivers on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 16		Sub_funciton code			
1		00 0F		axis (F → AXIS_XYZU)			

## 2.12 Setting the SERVO ALARM Function

- **\*int** ETM\_SET\_ALARM (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nMode, **BYTE** nAEdge)

### Description:

This function sets the ALARM input signal related parameters.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**nMode:** Mode: 0 = disable ALARM function;  
1 = enable ALARM function

**nAEdge:** Sets the trigger level  
0 = low active; 1 = high active

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_ALARM (s, 1, AXIS_ZU, 1, 0);  
//enable the ALARM for the Z and U axes on module 1 and set them  
//as low-active.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 17		Sub_funciton code			
1		00 0C		axis (C → AXIS_ZU)			
2		00 01		nMode (1 → enable)			
3		00 00		nAEdge (0 → low-active)			

## 2.13 Setting the Active Level of the In-Position Signals

- **\*int** ETM\_SET\_INPOS (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nMode, **BYTE** nlEdge)

### Description:

This function sets the INPOS input signal related parameters.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**nMode:** Mode: 0 = disable INPOS input;  
1 = enable INPOS input

**nlEdge:** Set the trigger level  
0 = low active; 1 = high active

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_INPOS (s, 1, AXIS_X, 1, 0);  
//enable the INPOS function of the X axis on module 1 and set it to be low-active.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 18		Sub_funciton code			
1		00 01		axis (1 → <i>AXIS_X</i> )			
2		00 01		nMode (1 → <i>enable</i> )			
3		00 00		nlEdge (0 → <i>low-active</i> )			

## 2.14 Setting the Time Constant of Digital Filter

- **\*int** ETM\_SET\_FILTER (**SOCKET** *s*, **BYTE** *cardNo*, **BYTE** *axis*, **BYTE** *FEn*, **BYTE** *FLn*)

### Description:

This function selects the axes and sets the time constant for digital filters of the input signals.

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

- s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).
- cardNo:** Module number
- axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).
- FEn:** Enabled filters.  
The sum of the code numbers (0~31) are used to select input signals. Please refer to the following table.

Code Number	Enabling filters
0	Disable
1	EMG, nLMTP, nLMTM, nIN0, nIN1
2	nIN2
4	nINPOS, nALARM
8	nEXPP, nEXPM, EXPLSN
16	nIN3

**FLn:** Sets the filter time constant (0~7) as follows.

Code	Removable max. noise width	Input signal delay time
0	1.75 $\mu$ SEC	2 $\mu$ SEC
1	224 $\mu$ SEC	256 $\mu$ SEC
2	448 $\mu$ SEC	512 $\mu$ SEC
3	896 $\mu$ SEC	1.024mSEC
4	1.792mSEC	2.048mSEC
5	3.584mSEC	4.096mSEC
6	7.168mSEC	8.192mSEC
7	14.336mSEC	16.384mSEC

**Return:**

**0: Success; Others: Fail (Please refer to Sectin 8.3)**

**Sample:**

```
ETM_SET_FILTER (s, 1, AXIS_XYZU, 21, 3);  
//set the filter time constants of X, Y, Z, and U axes as 1.024mSEC.  
//These filters include EMG, nLMTP, nLMTM, nIN0, nIN1, nINPOS, nALARM,  
//and nIN3.  
//(21 = 1+4+16) 1: EMG + nLMP + nLMPM + nIN0 + nIN1;  
// 4: nINPOS + nALARM;  
// 16: nIN3.
```

**Note:** The default wiring design is: nIN0 is connected to the NEAR HOME (NHOME) sensors; nIN1 is connected to the HOME sensors; and nIN2 is connected to the index of Encoder input (Z phase).

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 19		Sub_funciton code			
1		00 0F		axis (F → AXIS_XYZU)			
2		00 15		FEn (21 = 0x15, enable several noise filters)			
3		00 03		FLn (using filter 3)			

## 2.15 Position Counter Variable Ring

- **\*int** ETM\_VRING\_ENABLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** nVRing)

### Description:

This function enables the linear counter of the assigned axes as variable

### Category:

MODBUS sub\_function; RTC, MP and IT.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

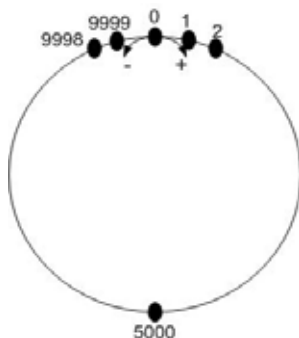
**nVRing:** Maximum value of the ring counter  
(0 ~ +2,000,000,000)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_VRING_ENABLE (s, 1, AXIS_X, 9999);  
//set the X axis of module 1 to be a ring counter. The encoder  
//values will be 0 to 9999.
```



The encoder value ranges from 0 to 9999.  
When the counter value reaches 9999, one more adding pulse will cause the counter to reset to 0. When the counter value is 0, a lessening pulse will let the counter set to

Max. ring encoder value = 9999

- Note:**
1. This function will set the LP and EP simultaneously.
  2. If this function is enabled, the software limit function cannot be used.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
--------------	--------------	--------------------------	--------------	-------------	-------------------	------------------------	------------------------

00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 1A		Sub_funciton code			
1		00 01		axis			
2		00 00		MSW of nVRing			
3		27 0F		LSW of nVRing (9999 = 0x270F)			



- **\*int** ETM\_VRING\_DISABLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

**Description:**

This function disables the variable ring counter function.

**Category:**

MODBUS sub\_function; RTC, MP and IT.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_VRING_DISABLE (s, 1, AXIS_X);
//disable the ring counter function for the X axis on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 1B		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			

## 2.16 Triangle prevention of fixed pulse driving

- **\*int** ETM\_AVTRI\_ENABLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

Description:

This function prevents a triangle form in linear acceleration (T-curve) fixed

Category:

MODBUS sub\_function; RTC, MP and IT.

Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

axis: Axis or axes (Please refer to Table 2-1a)

Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

```
ETM_AVTRI_ENABLE (s, 1, AXIS_X);  
//set the X axis of module 1 not to generate a triangle form in its speed profile.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 1C		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			

- **\*int** ETM\_AVTRI\_DISABLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

**Description:**

This function disables the function to prevent a triangle form in linear acceleration fixed pulse driving.

**Category:**

MODBUS sub\_function; RTC, MP and IT.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
Write the setting values into the parameter table without making a change of current settings(Please refer to Table 2-1a).

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_AVTRI_DISABLE (s, 1, AXIS_X);
//enable the X axis of module 1 to generate a triangle form in its
//speed profile if the pulse number for output is too low.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 1D		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			

## 2.17 External Pulse Input

**Cannot write settings for external input driving into the parameter Table.**

### 2.17.1 Handwheel (Manual Pulse Generator) Driving

- **int** ETM\_EXD\_MP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

**Description:**

This function outputs pulses according to the input pulses from a handwheel.

**Category:**

MODBUS sub\_function; RTC.

**Parameters:**

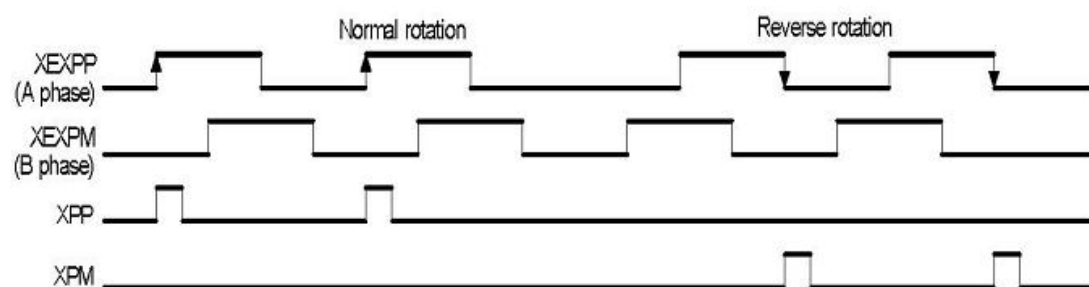
**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)

The axis can be either X,Y, Z or U.

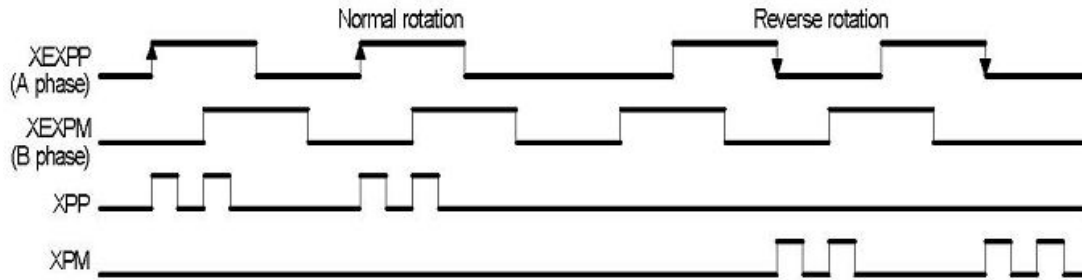
**data:** Number of command pulses (Rang: 0 ~ +2,000,000,000).



**ETM\_EXD\_MP (s, 1, AXIS\_X, 2);**

**//Each time the handwheel inputs a pulse to the X axis**

**//on module 1, the controller will output 2 pulses to the motor driver.**



The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 1E		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			
2		00 00		MSW of data			
3		00 01		LSW of data			

## 2.17.2 Fixed Pulse Driving Mode

- **int** ETM\_EXD\_FP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

This function outputs fixed pulses according to the trigger input (the falling edge of the nEXP+ signal) from a handwheel.

### Category:

MODBUS sub\_function; RTC.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).

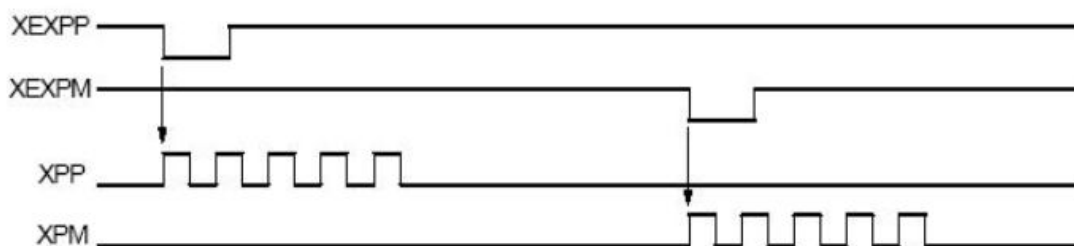
**data:** Number of command pulses (Rang: 0 ~ +2,000,000,000).

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_EXD_FP (s, 1, AXIS_X, 5);  
//Each time the controller detects a falling edge of an XEXP+  
//signal, it will output 5 pulses to the X axis.
```



The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 1F		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			
2		00 00		MSW of data			
3		00 05		LSW of data			

## 2.17.3 Continuous Pulse Driving Mode

- **int** ETM\_EXD\_CP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

The controller will continuously output pulses in positive direction if the falling edge of nEXP+ signal from a handwheel is detected. On the contrary, it will continuously output pulses in negative direction if the falling edge of nEXPsignal is detected.

### Category:

MODBUS sub\_function; RTC.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).

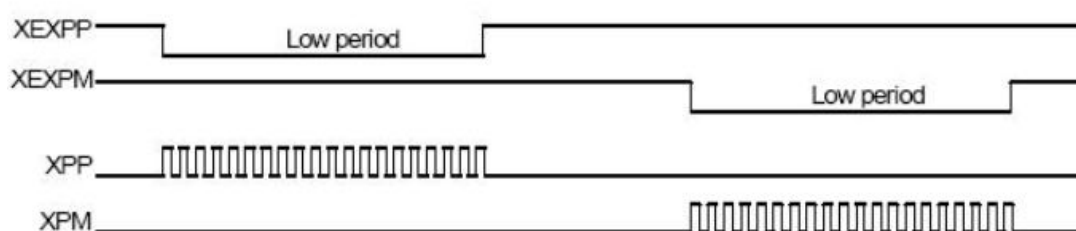
**data:** Number of command pulses (Rang: 0 ~ +2,000,000,000).

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_EXD_CP (s, 1, AXIS_X, 20);  
//Each time the controller detects a falling edge of an XEXP+  
//signal, it will continuously drive X axis at the speed of 20 PPS.
```



The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 20		Sub_funciton code			
1		00 01		axis			

2	00 00	<i>MSW of data</i>
3	00 14	<i>LSW of data (20 = 0x14)</i>



## 2.17.4 Disabling the External Signal Input Functions

- **int** ETM\_EXD\_DISABLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

### Description:

This function turns off the external input driving control functions.

### Category:

MODBUS sub\_function; RTC.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1a)  
The axis can be either X,Y, Z or U (1 or 2 or 4 or 8).

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_EXD_DISABLE (s, 1, AXIS_X);  
//disable the external input driving control function  
//of X axis on module 1
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 21		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			

## 2.18 Read/Write User defined Variables (VAR)

- **int** ETM03\_READ\_bVAR (**SOCKET** *s*, **BYTE** *cardNo*, **BYTE** *bvarNo*, **BYTE\*** *bVARValue*)

### Description:

This function read variable bVARn, it could be passed to Macro Program(MP), for detail information, please refer to Chapter 7.

### Category:

MODBUS table; RTC.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**bvarNo:** custom variable: bVAR0 ~ bVAR127  
**bVARValue:** The pointer to the memory that stores bVARn (0 ~ +255).

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

Suppose the value of bVAR100 is 100, or 0x64 .  
The hexadecimal value of (128+100) is E4.

```
BYTE bVARValue;  
ETM03_READ_bVAR (s, 1, bVAR100, &bVARValue);  
//read value of VAR100 in module 1
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	00 E4	00 01

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 Value (hex)
00 01	00 00	00 05	01	03	02	00 64

- **int** ETM\_WRITE\_bVAR (**SOCKET** s, **BYTE** cardNo, **BYTE** bvarNo, **BYTE** bVar)

**Description:**

This function write variable bVARn, it could be passed to Macro Program (MP), for detail information, please refer to Chapter 7.

**Category:**

MODBUS table, MODBUS sub\_function; RTC.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**bvarNo:** custom variable: bVAR0 ~ bVAR127

**bVar:** variable (0 ~ +255).

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

- Method 1: use Sub\_Function code method :

```
ETM_WRITE_bVAR (s, 1, bVAR100, 100);
//write bVAR100=100 in module 1
The address of bVARn =  $128 + n$  (or  $0x80 + n$ ).
( bVAR100 =  $128 + 100 = 228 = 0xE4$  )
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 23		Sub_funciton code			
1		00 E4		bvarNo = $n + 0x80 = 0xE4$			
2		00 64		bVar (100 = $0x64$ )			

- Method 2: Set the Holding Register table :

All bVARn are defined in the holding register table. The index of bVARn is (128+n); therefore, the index, or address, of bVAR100 is 228 (= 0xE4).

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	00 E4	00 01	02
Register[]		Value (hex)		Remarks			
0		00 64		<i>Sub_funciton code</i>			

- **long** ETM03\_READ\_VAR (**SOCKET** s, **BYTE** cardNo, **long** varNo, **long\*** VARValue)

**Description:**

This function read variable VARn, it could be passed to Macro Program (MP), for detail information, please refer to Chapter 7.

**Category:**

MODBUS table; RTC.

**Parameters:**

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

bvarNo: custom variable: VAR0 ~ VAR511

VARValue: The pointer to the memory that stores VARn (-2,000,000,000 ~ +2,000,000,000).

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

All VARn are defined in the holding register table. Each VARn takes two registers space. For example, VARn takes both registers indexed by  $(300 + 2*n)$  and  $(300 + 2*n + 1)$ . To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.

The Start\_Address (index) =  $300 + 2*100 = 500 = 0x01F4$

```
long VARValue;
ETM03_READ_VAR (s, 1, VAR100, &VARValue);
//read value of VAR100 in module 1
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	01 F4	00 02

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
00 01	00 00	00 07	01	03	04	00 00	27 10

To get the ldata, use the following instructions in C language.

ldata = Register[0];

```
ldata = ((ldata <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

- **int** ETM\_WRITE\_VAR (**SOCKET** s, **BYTE** cardNo, **long** varNo, **long** IVar)

**Description:**

This function write variable VARn, it could be passed to Macro Program (MP), for detail information, please refer to Chapter 7.

**Category:**

MODBUS table, MODBUS sub\_function; RTC.

**Parameters:**

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

varNo: Custom variable: VAR0 ~ VAR511

IVar: Value of variable(-2,000,000,000 ~ +2,000,000,000).

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_WRITE_VAR (s, 1, VAR100, 10000);
//write VAR100=10,000 in module 1
```

■ **Method 1: use Sub\_Function code method**

The address of VARn = 0x7fff0000 + n.

➔ the address of VAR100 = 0x7fff0000 + 0x64

And, 10000 = 0x2710

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			
0		0A 25		Sub_funciton code			
1		7F FF		MSW of varNo (MSW of 0x7FFF0064)			
2		00 64		LSW of varNo (LSW of 0x7FFF0064)			
3		00 00		MSW of IVar (MSW Of 0x2710)			
4		2710		LSW of IVar (LSW Of 0x2710)			

■ **Method 2: Set the Holding Registers.**

All VARn are defined in the holding register table. Each VARn takes two

registers space. For example, VARn takes both registers indexed by  $(300 + 2*n)$  and  $(300 + 2*n + 1)$ . To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.

For VARn, the Start address =  $(300 + 2*n)$ .

→ The Start\_Address =  $300 + 2*100 = 500 = 0x01F4$

Suppose the value of VAR100 is 10000 (= 0x2710).

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	01 F4	00 02	04
Register[]		Value (hex)		Remarks			
0		00 00		MSW of IVar (MSW of 0x2710)			
1		27 10		LSW of IVar (LSW of 0x2710)			



## 2.19 Read/Write Data for Power outage carry-over (MD)

- **int** ETM03\_READ\_MD (**SOCKET** s, **BYTE** cardNo, **long** mdNo, **long\*** ldata, **float\*** fdata)

### Description:

This function reads the machine data.

### Category:

MODBUS table; RTC.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**mdNo:** Machine data(**long**): MD0 ~ MD1023  
Machine data (**float**): MD1024 ~ MD2047

**&ldata:** Read MD **long** (-2,147,483,648 ~ +2,147,483,647)

**&fdata:** Read MD **float** (Integer number plus decimal number giving a total of six places).

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

All MDn are defined in the holding register table. Each MDn takes two registers space. For example, MDn takes both registers indexed by (3000 + 2\*n) and (3000 + 2\*n + 1). To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.

The MD100 Start\_Address = 3000 + 2\*100 = 3200 = **0x0C80**

The MD1500 Start\_Address = 3000 + 2\*1500 = 6000 = **0x1770**

```
long ldata;  
float fdata;  
ETM03_READ_MD (s, 1, MD100, &ldata, 0);  
//read MD100 long data in module 1  
ETM03_READ_MD(s, 1, MD1500, 0, &fdata);  
//read MD1500 float data in module 1
```

ldata: the MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	0C 80	00 02

**ldata:** The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
00 01	00 00	00 07	01	03	04	12 34	56 78

**fdata:** the MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	17 70	00 02

**fdata:** The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
00 01	00 00	00 07	01	03	04	43 A0	D3 33

To get the ldata, use the following instructions in C language.

ldata = Register[0];

ldata = ((ldata <<16) & 0xffff0000) | (Register[1] & 0xffff);

- **int** ETM\_WRITE\_MD (**SOCKET** s, **BYTE** cardNo, **long** mdNo, **long** ldata, **float** fdata)

#### Description:

This function writes the machine data.

#### Category:

MODBUS table, MODBUS sub\_function; RTC.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**mdNo:** Machine data(**long**): MD0 ~ MD1023  
Machine data (**float**): MD1024 ~ MD2047

**&ldata:** Write MD **long** (-2,147,483,648 ~ +2,147,483,647)

**&fdata:** Write MD **float** (Integer number plus decimal number giving a total of six places).

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_WRITE_MD (s, CardNo, MD100, 0x12345678, 0);
//write MD100 long data in module 1
```

#### ■ Method 1: use Sub\_Function code method

The address of MDn = 0x00000000 + n.

➔ the address of MD100 = 0x00000000 + 0x64

**Write MD100 = 0x12345678**

**Write MD1500 = 321.65**

ldata: the MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 15	01	10	1F 40	00 07	0E
Register[]		Value (hex)		Remarks			
0		0A 27		Sub_funciton code			
1		00 00		MSW of mdNo (= 0)			
2		00 64		LSW of mdNo (n= 0x64)			
3		12 34		MSW of ldata			
4		56 78		LSW of ldata			

5	00 00	MSW of fdata
6	00 00	LSW of fdata

fdata: the MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 15	01	10	1F 40	00 07	0E
Register[]		Value (hex)		Remarks			
0		0A 27		Sub_funciton code			
1		00 00		MSW of mdNo (= 0)			
2		05 DC		LSW of mdNo (n= 0x05DC)			
3		00 00		MSW of ldata			
4		00 00		LSW of ldata			
5		43 A0		MSW of fdata			
6		D3 33		LSW of fdata			

## ■ Method 2: Set the Holding Registers.

All MDn are defined in the holding register table. Each MDn takes two registers space. For example, MDn takes both registers indexed by (3000 + 2\*n) and (3000 + 2\*n + 1). To read or write the two-register variable, the first register must be addressed first and the Length must be the multiple of 2.

→The Start\_Address = 3000 + 2\*100 = 3200 = 0x0C80

ldata: the MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	0C 80	00 02	04
Register[]		Value (hex)		Remarks			
0		12 34		MSW of MD100			
1		56 78		LSW of MD100			

→The Start\_Address = 3000 + 2\*1500 = 6000 = 0x1770

ldata: the MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	17 70	00 02	04
Register[]		Value (hex)		Remarks			
0		43 A0		MSW of MD1500			

1	D3 33	LSW of MD1500
---	-------	---------------

---

## 3. Reading and Register Setting

---

### 3.1 Setting and Reading Command Position

- **\*\*int** ETM\_SET\_LP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long** wdata)

Description:

This function sets the command position counter value (logical position counter, LP).

Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**wdata:** Position command  
(-2,000,000,000 ~ +2,000,000,000)

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

```
ETM_SET_LP (s, 1, AXIS_XYZU, 10000);  
//Set the LP as 0 for X, Y, Z, and U axes in module 1  
// will clear all LP counters on module 1
```

- Method 1: use sub\_function code method

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 28		Sub_funciton code			
1		00 0F		axis (F → <a href="#">AXIS_XYZU</a> )			
2		00 00		MSW of wdata			
3		27 10		LSW of wdata			

■ **Method 2: Set the holding registers.**

The Start address is 90, or 0x5A, for LP\_X. Since the LP needs 2 registers to hold the value, the start address setting for MODBUS request must begin from the MSW of the required LP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.

Method 2 allows users to set different values for 4 different axes at only one MODBUS request, which is not possible by using method 1.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 17	01	10	00 5A	00 08	10
Register[]		Value (hex)		Remarks			
0		00 00		LP_X_MSW (address = 0x5A)			
1		27 10		LP_X_LSW			
2		00 00		LP_Y_MSW (address = 0x5C)			
3		27 10		LP_Y_LSW			
4		00 00		LP_Z_MSW (address = 0x5E)			
5		27 10		LP_Z_LSW			
6		00 00		LP_U_MSW (address = 0x60)			
7		27 10		LP_U_LSW			

- **\*\*int** ETM\_GET\_LP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)
- **\*\*int** ETM03\_GET\_LP(**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long\*** LPValue)

#### Description:

This function reads the command position counter value (logical position counter, LP).

#### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis (Please refer to Table 2-1)

**LPValue:** The pointer to the memory that stores the logical position counter (-2,000,000,000 ~ +2,000,000,000).

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

- Method 1: Use Holding Registers to get the current values.

Since each LP needs 2 registers to hold the value, the start address setting for MODBUS request must begins from the MSW of the required LP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.

This method allows users to get more than one axis data at only one MODBUS request. Please refer to MODBUS holding registers definition table.

```
long X_LP;
ETM03_GET_LP (s, 1, AXIS_X, &X_LP);
//Reads the LP value of the X axis on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	00 5A	00 02

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
--------------	--------------	--------------------------	--------------	-------------	------------------------	----------------	----------------



00 01	00 00	00 07	01	03	04	00 00	27 10
-------	-------	-------	----	----	----	-------	-------

To get the X\_LP, use the following instructions in C language.

```
X_LP = Register[0];
```

```
X_LP = ((X_LP <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

- Method 2: It is used as an MP instruction.

For this case, the current LP value does not actually be returned at the moment when request is sent. The getting LP instruction is desired to be executed later when an MP is called. Users can **FC = 16** to write this command into an MP. Inside this MP, there can be an **ETM\_MP\_SET\_RVAR()** function just next to it to save the return LP value to a specified variable. This variable can be referred by other functions. Please refer to other related MP usages explanations.

```
ETM_GET_LP (s, 1, AXIS_X);
```

```
//Reads the LP value of the X axis on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 29		Sub_funciton code			
1		00 01		axis (1 → <b>AXIS_X</b> )			

## 3.2 Setting and Reading the Encoder Counter

- **\*\* int** ETM\_SET\_EP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long** wdata)

Description:

This function sets the encoder position counter value (real position counter or EP).

Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

axis: Axis or axes (Please refer to Table 2-1)

wdata: Position command  
(-2,000,000,000 ~ +2,000,000,000)

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

```
ETM_SET_EP (s, 1, AXIS_XYZU, 10000);  
//Set the EP as 0 for X, Y, Z, and U axes of module 1.  
//This command clears all EP counters on module 1.
```

### ■ Method 1: use Sub\_Function code method

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 2A		Sub_funciton code			
1		00 0F		axis (F → AXIS_XYZU)			
2		00 00		MSW of wdata			
3		27 10		LSW of wdata			

### ■ Method 2: Set the Holding Registers.

Since each EP needs 2 registers to hold the value, the start address

setting for MODBUS request must begins from the MSW of the required EP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.

This method allows users to set more than one axes data at only one MODBUS request. Please refer to MODBUS holding registers definition table.

The start address is 98, or 0x62, for EP\_X.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 17	01	10	00 62	00 08	10
Register[]		Value (hex)		Remarks			
0		00 00		LP_X_MSW (address = 0x5A)			
1		27 10		LP_X_LSW			
2		00 00		LP_Y_MSW (address = 0x5C)			
3		27 10		LP_Y_LSW			
4		00 00		LP_Z_MSW (address = 0x5E)			
5		27 10		LP_Z_LSW			
6		00 00		LP_U_MSW (address = 0x60)			
7		27 10		LP_U_LSW			

- **\*\*int** ETM\_GET\_EP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)
- **\*\*int** ETM03\_GET\_EP (**SOCKET** s, **BYTE** cardNo, **BYTE** axis , **long\*** EPValue)

**Description:**

This function reads the encoder position counter value (LP).

**Category:**

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis (Please refer to Table 2-1).  
The axis can be either X, Y, Z, or U.

**EPValue:** The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~ +2,000,000,000).

**Return:**

0: Success; Others: Fail (Please refer to Section 8.3)

**Sample:**

Suppose the EP is 10000, or 0x2710. The Start address is 98, or 0x62. Since the EP needs 2 registers to hold the value, the start address setting for MODBUS request must begin from the MSW of the required EP. Otherwise, the host will receive an exception error. This rule will be applied to other long, DWORD or float type values.

- Method 1: Use holding registers to get the current values.

```
long X_EP;
ETM03_GET_EP (s, 1, AXIS_X, &X_EP);
//reads the encoder position value (EP) of the X axis on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	00 62	00 02

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
00 01	00 00	00 07	01	03	04	00 00	27 10

To get the X\_EP, use the following instructions in C language.

```
X_EP = Register[0];
```

```
X_EP = ((X_EP <<16) & 0xffff0000) | (Register[1] & 0xffff);
```

- **Method 2:** It is used for creating the content of an MP.

For this case, users do not actually want to get the current EP values. The getting EP will be executed only when the MP is called. Therefore, **use FC = 16 to write this command inside a MP. This kind of usages often has ETM\_MP\_SET\_RVAR() followed to get the return EP value.** Please refer to MP related explanation literature.

**ETM\_GET\_EP** (s, 1, AXIS\_X);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A2B		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			

### 3.3 Reading the Current Velocity

- **Int** ETM03\_GET\_CV (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long\*** CVValue)

Description:

This function reads the current velocity value.

Category:

MODBUS table; RTC.

Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis (Please refer to Table 2-1).  
The axis can be either X, Y, Z, or U.

**EPValue:** The pointer to the memory that stores the current velocity

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

Suppose the CV is 10000, or 0x2710. The Start address is 106, or 0x6A.

**long** CVValue;

CVValue = ETM03\_GET\_CV (s, 1, AXIS\_X, &CVValue);

//reads the current velocity of the X axis on module 1

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	00 6A	00 02

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
00 01	00 00	00 07	01	03	04	00 00	27 10

To get the CV\_X, use the following instructions in C language.

CV\_X = Register[0];

CV\_X = ((CV\_X <<16) & 0xffff0000) | (Register[1] & 0xffff);

## 3.4 Reading the Current Acceleration

- **Int** ETM03\_GET\_CA (**SOCKET** s, **BYTE** cardNo, **BYTE** axis , **long\*** CAValue)

Description:

This function reads the current acceleration value PPS/Sec.

Category:

MODBUS table; RTC.

Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis (Please refer to Table 2-1).  
The axis can be either X, Y, Z, or U.

**EPValue:** The pointer to the memory that stores the current acceleration (in PPS/Sec).

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

Suppose the CA is 10000, or 0x2710. The Start address is 114, or 0x72.  
DWORD dwdata;  
dwdata = ETM03\_GET\_CA (s, 1, AXIS\_X);  
*//reads the current acceleration value of the X axis on module 1.*

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	00 72	00 02

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
00 01	00 00	00 07	01	03	04	00 00	27 10

To get the CA\_X, use the following instructions in C language.

CA\_X = Register[0];

CA\_X = ((CA\_X <<16) & 0xffff0000) | (Register[1] & 0xffff);



## 3.5 Reading the DI Status

- **\*\*int** ETM\_GET\_DI (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nType)
- **\*\*int** ETM04\_GET\_DI (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nType, **BYTE\*** DIState)

### Description:

This function reads the digital input (DI) status.

### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis (Please refer to Table 2-1).  
The axis can be either X, Y, Z, or U.

**nType:** 0 → DRIVING (Check whether the axis is driving or not)  
1 → LIMIT+ (Check whether the limit+ is engaged or not)  
2 → LIMIT- (Check whether the limit- is engaged or not)  
3 → EMERGENCY (Check whether EMG signal is on or not)  
4 → ALARM (Check the ALARM input signal)  
5 → IN1 (Check the IN1 input signal)  
6 → IN0 (Check the IN0 input signal)  
7 → IN3 (Check the IN3 input signal)  
8 → INPOS (Check the INPOS input signal)  
9 → IN2 (Check the IN2 input signal)

**DIState:** The pointer to the memory that stores the single DI Status (1: ON; 0: OFF )

### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

### Sample:

- Method 1: It is used for getting current status of DI.

Each bit corresponds to a register. Each register's value is either 0 or 1.

**BYTE DIState;**

**ETM04\_GET\_DI(s, 1, AXIS\_X, 1, &DIState);**

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
--------------	--------------	--------------------------	--------------	-------------	-------------------	------------------------

00 01	00 00	00 06	01	04	00 15	00 01
-------	-------	-------	----	----	-------	-------

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Status of LIMIT+ of AXIS_X ( Reg_0 )
00 01	00 00	00 05	01	04	02	00 00

- Method 2: It is used for creating the content of an MP.

For this case, users do not actually want to get the current DI values. The getting DI will be executed only when the MP is called. Therefore, use **FC = 16** to write this command inside a MP. This kind of usage often has **ETM\_MP\_SET\_RVAR()** followed to get the return DI value. Please refer to MP related explanation literature.

**ETM\_GET\_DI** (s, 1, AXIS\_X, 1);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 2E		Sub_funciton code			
1		00 01		axis			
2		00 01		The specified DI (LIMIT+)			

- **\*\*int** ETM\_GET\_DI\_ALL (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)
- **\*\*int** ETM04\_GET\_DI\_ALL (**SOCKET** s, **BYTE** cardNo, **BYTE** axis ,  
**WORD\*** DValue)

#### Description:

This function reads the digital input (DI) status.

#### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis (Please refer to Table 2-1).  
The axis can be either X, Y, Z, or U.

**DValue:** The pointer to the memory that stores the all DI Status  
The bits in *DValue* are defined as follows:

<i>Bit 0</i> → DRIVING	(Axis is driving or not)
<i>Bit 1</i> → LIMIT+	(Limit+ is engaged or not)
<i>Bit 2</i> → LIMIT-	(Limit- is engaged or not)
<i>Bit 3</i> → EMERGENCY	(EMG signal,)
<i>Bit 4</i> → ALARM	(ALARM signal,)
<i>Bit 5</i> → IN1	(IN1 input signal, low-active)
<i>Bit 6</i> → IN0	(IN0 input signal, low-active)
<i>Bit 7</i> → IN3	(IN3 input signal, low-active)
<i>Bit 8</i> → INPOS	(INPOS input signal)
<i>Bit 9</i> → IN2	(IN2 input signal, low-active)

#### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

#### Sample:

- Method 1: It is used for getting current status of DI.

Each bit corresponds to a register. Each register's value is either 0 or 1.  
**WORD DValue;**  
**ETM04\_GET\_DI\_ALL** (s, 1, AXIS\_X, &DValue);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
--------------	--------------	--------------------------	--------------	-------------	-------------------	------------------------

00 01	00 00	00 06	01	04	00 00	00 01
-------	-------	-------	----	----	-------	-------

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	DI Status of AXIS_X. ( Reg_0 )
00 01	00 00	00 05	01	04	02	00 00

■ Method 2: When it is used inside an MP.

For this case, users do not actually want to get the current DI values. The getting DI will be executed only when the MP is called. Therefore, use **FC = 16** to write this command inside a MP. This kind of usage often has **ETM\_MP\_SET\_RVAR()** followed to get the return DI value. Please refer to MP-related explanation literature.

**ETM\_GET\_DI\_ALL** (s, 1, AXIS\_X);  
**//ETM\_MP\_SET\_RVAR(s, 1, VAR0); //assign this DI value to VAR0**

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 31		Sub_funciton code			
1		00 01		axis (1: AXIS_X)			

## 3.6 Reading and Clearing the ERROR Status

- **\*\*int** ETM\_GET\_ERROR (**SOCKET** s, **BYTE** cardNo)
- **\*\*int** ETM04\_GET\_ERROR\_STATE (**SOCKET** s, **BYTE** cardNo, **BYTE\*** ErrState)

Description:

This function checks whether an error occurs or not.

Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

Parameters:

- s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).
- cardNo: Module number
- ErrState: The pointer to the memory that stores the error status
- 1: Error(s) occurred. Please perform ETM\_GET\_ERROR\_CODE () to get more information. If GET\_ERROR\_CODE =256, it indicates that the motion stop was due to the “STOP” command, not because of an error happened. Please refer to Section [6.5.5](#) and the following **sample** for cleaning ERROR.
- 0: No error.

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

- Method 1: It is used for getting current status of error.  
**BYTE** ErrState;  
**ETM04\_GET\_ERROR\_STATE** (s, 1, &ErrState);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	04	00 05	00 01

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	Error status ( Reg_0 )
00 01	00 00	00 05	01	04	02	00 00

- **Method 2:** It is used for creating the content of an MP.

For this case, users do not actually want to get the current error status. The getting error will be executed only when the MP is called. Therefore, use **FC = 16** to write this command inside a MP. This kind of usage often has **ETM\_MP\_SET\_RVAR()** followed to get the return value. Please refer to MP related explanation literature.

**ETM\_GET\_ERROR** (s, 1);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A 2F		<i>Sub_funciton code</i>			

- **\*\* int** ETM\_GET\_ERROR\_CODE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)
- **\*\* int** ETM04\_GET\_ERROR\_CODE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **WORD\*** ErrCode)

#### Description:

This function reads the ERROR status.

#### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis (Please refer to Table 2-1).  
The axis can be either X, Y, Z, or U.

**ErrCode:** The pointer to the memory that stores the error status  
0 → No error.  
For non-zero return value, please refer to the following table. If there are more than one error, the return value will be the sum of there error code values.

Erro Code	Cause of stop	Explanation
1	SOFT LIMIT+	Occurs when the forward software limit is asserted
2	SOFT LIMIT-	Occurs when the reverse software limit is asserted
4	LIMIT+	Occurs when the forward hardware limit is asserted
8	LIMIT-	Occurs when the reverse hardware limit is asserted
16	ALARM	Occurs when the ALARM is asserted
32	EMERGENCY	Occurs when the EMG is asserted
64	Reserved	Reserved
128	HOME	Occurs when both Z phase and HOME are asserted
256	Refer to <a href="#">6.5.4</a>	Occurs when the EMG(software) is asserted

For example, a return code **48** means that ALARM and EMGERENCY occur at the same time.

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

- **Method 1:** It is used for getting current error code.

**WORD** ErrCode;

**ETM04\_GET\_ERROR\_CODE** (s, 1, AXIS\_X, &ErrCode);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	04	00 10	00 01

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	ErrCode ( Reg_0 )
00 01	00 00	00 05	01	04	02	00 00

- Method 2: It is used for creating the content of an MP.

For this case, users do not actually want to get the current error codes. The getting error code will be executed only when the MP is called. Therefore, **use FC = 16** to write this command inside a MP. This kind of usage often has **ETM\_MP\_SET\_RVAR()** followed to get the return DI value. Please refer to MP related explanation literature.

**ETM\_GET\_ERROR\_CODE** (s, 1, AXIS\_X);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 30		Sub_funciton code			
1		00 01		axis (AXIS_X)			



---

## 4. FRnet Functions

---

### 4.1 Read FRnet DI Signals

- **\*\*int** ETM\_FRNET\_IN (**SOCKET** s, **BYTE** cardNo, **BYTE** wGroup)
- **\*\*int** ETM04\_GET\_FRNET\_GPDI (**SOCKET** s, **BYTE** cardNo, **BYTE** wGroup, **WORD\*** GPDIValue)

#### Description:

This function reads the FRnet digital input signals. One group comprises 16 bits data. Therefore, total 128 DI can be defined for one FRnet interface.

#### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**wGroup:** Group number, 8~15

**GPDIValue:** The pointer to the memory that stores the FRnet DI status.

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

Each register contains 16-bit data. Each bit corresponds to a DI of group.

- **Method 1:** It is used for getting FRnet DI values of assigned group.  
WORD GPDIValue;  
**ETM04\_GET\_FRNET\_GPDI** (s, 1, 8, &GPDIValue);  
*//Read the 16-bit DI which is on module 1 and the group number is 8.*

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	04	00 08	00 01

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	FRnet DI of Group 8 ( Reg_0 )
--------------	--------------	--------------------------	--------------	-------------	------------------------	-------------------------------------

00 01	00 00	00 05	01	04	02	00 00
-------	-------	-------	----	----	----	-------

- Method 2: It is used for creating the content of an MP.

For this case, users do not actually want to get the current DI value. The getting will be executed only when the MP is called. Therefore, use **FC = 16** to write this command inside a MP. This kind of usage often has **ETM\_MP\_SET\_RVAR()** followed to get the return DI value. Please refer to MP related explanation literature.

**ETM\_FRNET\_IN** (s, 1, 8);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 32		Sub_funciton code			
1		00 08		Group number (8~15)			

## 4.2 Write data to FRnet DO

- **\*\*int** ETM\_FRNET\_OUT (**SOCKET** s, **BYTE** cardNo, **BYTE** wGroup, **DWORD** data)

### Description:

This function writes data to the FRnet digital output. One group comprises 16 bits data. Therefore, total 128 DO can be defined for one FRnet interface.

### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**wGroup:** Group number, 0~7

**data:** 0x00000000 ~ 0x0000ffff

### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

### Sample:

- **Method 1:** It is used for setting FRnet DO values by some true values. Each register contains 16-bit data. Each bit corresponds to a DO of group. **Please note that FRnet DO values can be read back.**

```
ETM_FRNET_OUT (s, 1, 0, 0x0000FFFF);  
ETM_FRNET_OUT (s, 1, 1, 0x0000AAAA);  
ETM_FRNET_OUT (s, 1, 2, 0x00001100);  
ETM_FRNET_OUT (s, 1, 3, 0x00000011);
```

UID+FC (hex)	Start_Address (hex)	WORD COUNT (hex)
0110	0	4
Register[]	Value (hex)	Remarks
0	FFFF	DO setting (for address = 0)
1	AAAA	DO setting (for address = 1)
2	1100	DO setting (for address = 2)
3	0011	DO setting (for address = 3)

- **Method 2:** It can be used for setting FRnet DO values by true values or by variables.

The DO is defined as a DWORD because it can be a variable or a true value. **When a VARn is used instead of the true value, the MSW will not be zero.** When users desire to use a true value to set the DO, please keep the MSW to be zero, and let the LSW part contain the DO value.

```
ETM_FRNET_OUT (s, 1, 0, 0x0000FFFF);
// on module 1, set group number RA=0, output data is 0x0000ffff
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 33		Sub_funciton code			
1		00 00		Group number (0~7)			
2		00 00		MSW of DO			
3		FF FF		LSW of DO			

```
ETM_FRNET_OUT (s, 1, bVAR0, VAR1);
// on module 1, set group number RA= bVAR0, output data is VAR1
```

Note: The address of bVAR0 is 0x80 ( = 0x80+n); and the address of VAR0 is 0x7FFFF0001 (= 0x7FFFF0000 + n).

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 33		Sub_funciton code			
1		00 80		Group number (0~7)			
2		7F FF		MSW of DO			
3		00 01		LSW of DO			

## 4.3 Read back FRnet DO Signals

- **\*\*int** ETM03\_GET\_FRNET\_GPDO (**SOCKET** s, **BYTE** cardNo, **BYTE** wSA, **WORD\*** GPDOValue)

### Description:

This function reads back the FRnet digital output signals. One group comprises 16 bits data. Therefore, total 128 DO can be defined for one FRnet interface.

### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**wGroup:** Group number, 0~7  
**GPDOValue:** The pointer to the memory that stores the FRnet DO signals.

### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

### Sample:

Each register contains 16-bit data. Each bit corresponds to a DO of group. It is used for getting FRnet DO values of assigned group.

```
WORD GPDOValue;  
ETM03_GET_FRNET_GPDO (s, 1, 0, &GPDOValue);  
//Read back the 16-bit DO signal of RA0
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	03	00 00	00 01

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	FRnet DO of Group 0 ( Reg_0 )
00 01	00 00	00 05	01	03	02	FF FF

---

## 5. Auto Homing Search

---

i-8094H module provides automatic home search function, after the configuration of proper settings, it would function automatically. The main steps are as bellows:

- Near-home sensor searching (NHOME) under high-speed motion.
- Home sensor searching under low-speed motion.
- Servo motor Z-Phase searching under low-speed motion.
- Offset movement to the origin of the working area under high-speed motion.

A few steps could be skipped to adjust settings accordingly to meet customer's actual needs. This operation could be performed automatically, therefore economize on CPU resource and reduce programming efforts.

### 5.1 Set Up Homing Speed

- **\* int** ETM\_SET\_HV (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

Description:

This function sets the homing speed.

Category:

MODBUS sub\_function; RTC and MP.

Parameters:

<b>s:</b>	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<b>cardNo:</b>	Module number
<b>axis:</b>	Axis or axes (Please refer to Table 2-1)
<b>data:</b>	Homing speed (Vmin~Vmax PPS)

Return:

0: Success; Others: Fail (Please refer to Section 8.3)

Sample:

```
ETM_SET_HV (s, 1, AXIS_X, 500);  
//set the homing speed of the X axis on module 1 to be 500 PPS.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 3C		<i>Sub_funciton code</i>			
1		00 01		<i>axis</i>			
2		00 00		<i>MSW of data</i>			
3		01 F4		<i>LSW of data (500 = 0x1F4)</i>			

## 5.2 Using an Limit Switch as the Home sensor

- **\*int** ETM\_HOME\_LIMIT (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nType)

### Description:

This function sets the Limit Switch to be used as the HOME sensor.

### Category:

MODBUS sub\_function; RTC and MP.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**nType:** 0: Disable the The LIMIT SWITCH function;  
1: Use the LIMIT SWITCH as the HOME sensor.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_HOME_LIMIT (s, 1, AXIS_X, 0);  
// Do not use the Limit Switch as the HOME sensor.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 3D		Sub_funciton code			
1		00 01		axis			
2		00 00		nType			



## 5.3 Setting the Homing Mode

- **\*int** ETM\_SET\_HOME\_MODE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nStep1, **BYTE** nStep2, **BYTE** nStep3, **BYTE** nStep4 , **DWORD** data)

### Description:

This function sets the homing method and other related parameters.

### Category:

MODBUS sub\_function; RTC and MP.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**nStep1:** 0: Step 1 will not be executed.  
1: Moves in the positive direction.  
2: Moves in the negative direction.

**nStep2:** 0: Step 2 will not be executed.  
1: Moves in a positive direction.  
2: Moves in a negative direction.

**nStep3:** 0: Step 3 will not be executed.  
1: Moves in a positive direction.  
2: Moves in a negative direction.

**nStep4:** 0: Step 4 will not be executed.  
1: Moves in a positive direction.  
2: Moves in a negative direction.

**data:** Offset value (0 ~ +2,000,000,000)

The Four Steps Required for Automatic Homing

Step	Action	Speed	Sensor
1	Searching for the Near Home sensor	V	NHOME (IN0)
2	Searching for the HOME sensor	HV	HOME (IN1)
3	Searching for the encoder Z-phasesignal	HV	Z-Phase (IN2)
4	Moves to the specified position	V	

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

ETM\_SET\_HOME\_MODE (s, 1, 0x1, 2, 2, 1, 1, 3500);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 17	01	10	1F 40	00 08	10
Register[]		Value (hex)		Remarks			
0		0A 3E		Sub_funciton code			
1		00 01		axis			
2		00 02		nStep1			
3		00 02		nStep2			
4		00 01		nStep3			
5		00 01		nStep4			
6		00 00		MSW of data			
7		0D AC		LSW of data (3500 = 0xDAC)			

## 5.4 Starting the Homing Sequence

- **\*int** ETM\_HOME\_START (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

Description:

This function starts the home search of assigned axes.

Category:

MODBUS sub\_function; RTC and MP.

Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

```
ETM_HOME_START (s, 1, AXIS_X);  
//start the automatic homing sequence for the X axis on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 3F		Sub_funciton code			
1		00 01		axis (AXIS_X)			

---

## 6. Basic Motion Control

---

### 6.1 Independent Motion Control for each axis

- Multiple axes could move at the same time.
- The motion of each axis can be started independently.
- Each axis is moving independently.
- Each axis can receive commandeds to change motion, such as changing the number of pulses or the speed.
- Each axis can receive commandeds to stop slowly or suddenly to meet the specific requirements.
- Independent axis motion can work with interpolation or synchronous action to perform more complicated and versatile motion.

#### 6.1.1 Setting the Acceleration/Deceleration Mode

- **\*\* int** ETM\_NORMAL\_SPEED (**SOCKET** s, **BYTE** cardNo, **BYTE** axis , **BYTE** nMode)

##### Description:

This function sets the speed mode.

##### Category:

MODBUS sub\_function; RTC, MP and ISR.

##### Parameters:

<b>s:</b>	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<b>cardNo:</b>	Module number
<b>axis:</b>	Axis or axes (Please refer to Table 2-1)
<b>nMode:</b>	0 → Symmetric T-curve (Please set SV, V, A, and AO). 1 → Symmetric S-curve (Please set SV, V, K, and AO). 2 → Asymmetric T-curve (Please set SV, V, A, D, and AO). 3 → Asymmetric S-curve (Please set SV, V, K, L, and AO).

**Note:** Please refer the configurations of speed-related parameters.

##### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

##### Sample:

```
ETM_NORMAL_SPEED (s, 1, AXIS_XYZU, 0);
```

//use a symmetric T-curve for all axes on module 1.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 46		Sub_funciton code			
1		00 0F		axis (AXIS_XYZU)			
2		00 00		nMode			

## 6.1.2 Setting the Starting Speed

- **\*\* int** ETM\_SET\_SV (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

This function sets the initial speed for the assigned axes.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** Set up speed(Please refer to 2.5 for maxim speed)PPS.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_SV (s, 1, AXIS_X, 1000);  
//set the starting speed for the X axis on module 1 to 1000 PPS.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 47		Sub_funciton code			
1		00 01		axis (1 = AXIS_X)			
2		00 00		MSW of data			
3		03 E8		LSW of data (1000 = 0x3E8)			

### 6.1.3 Setting the Desired Speed

- **\*\*int** ETM\_SET\_V (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

#### Description:

This function sets the desired speed for the assigned axes.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** Set up speed(Please refer to 2.5 for maximum speed)PPS.

#### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

#### Sample:

```
ETM_SET_V (s, 1, AXIS_X, 120000L);  
//set the speed for the X axis on module 1 to 120000 PPS.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 48		Sub_function code			
1		00 01		axis (1 = AXIS_X)			
2		00 01		MSW of data			
3		D4 C0		LSW of data (120000 = 0x1D4C0)			

## 6.1.4 Setting the Acceleration

- **\*\* int** ETM\_SET\_A (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

This function sets the acceleration value for the assigned axes.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** The acceleration value (PPS/Sec).  
This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V() function. The maximum available acceleration value is MAX\_V \* **125**. The minimum acceleration value is MAX\_V ÷ **64**, and all other acceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_A (s, 1, AXIS_X, 100000L);  
//set the acceleration value of the X axis on module 1 to 100K PPS/Sec.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 49		Sub_funciton code			
1		00 01		axis (1 = AXIS_X)			
2		00 01		MSW of data			
3		86 A0		LSW of data (100000 = 0x186A0)			



## 6.1.5 Setting the Deceleration

- **\*\* int** ETM\_SET\_D (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

This function sets the deceleration value for the assigned axes.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** The deceleration value (PPS/Sec).  
This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V () function. The maximum available acceleration value is MAX\_V \* **125**. The minimum acceleration value is MAX\_V ÷ **64**, and all other acceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_D (s, 1, AXIS_X, 100000L);  
//set the deceleration value of the X axis on module 1 to 100K PPS/Sec.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 4A		Sub_funciton code			
1		00 01		axis (1 = AXIS_X)			
2		00 01		MSW of data			
3		86 A0		LSW of data (100000 = 0x186A0)			

## 6.1.6 Setting the Acceleration Rate

- **\*\* int** ETM\_SET\_K (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

The function sets the acceleration rate (i.e., Jerk) value for the assigned axes.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

- s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).
- cardNo:** Module number
- axis:** Axis or axes (Please refer to Table 2-1)
- data:** The acceleration rate (jerk) value(PPS/ Sec<sup>2</sup>).  
This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V () function.  
The minimum acceleration rate is MAX\_V \* **0.0119211**;  
The maximum acceleration rate value is **2,000,000,000**.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_K (s, 1, AXIS_X, 10000);  
//set the acceleration rate value of the X axis on module 1 to  
//1,000*10 (= 10,000) PPS/Sec^2.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 4B		Sub_funciton code			
1		00 01		axis (1 = AXIS_X)			
2		00 00		MSW of data			
3		27 10		LSW of data (10000 = 0x2710)			

## 6.1.7 Setting the Deceleration Rate

- **\*\* int** ETM\_SET\_L (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **DWORD** data)

### Description:

The function sets the deceleration rate (i.e., Jerk) value for the assigned axes.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** The deceleration rate (jerk) value(PPS/ Sec<sup>2</sup>).  
This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V () function.  
The minimum acceleration rate is MAX\_V \* 0.0119211;  
The maximum acceleration rate value is 2,000,000,000.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_L (s, 1, AXIS_X, 10000);  
//set the deceleration rate value of the X axis on module 1 to  
//1,000*10 (= 10,000) PPS/Sec^2.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 4C		Sub_funciton code			
1		00 01		axis (1 = AXIS_X)			
2		00 00		MSW of data			
3		27 10		LSW of data (10000 = 0x2710)			

## 6.1.8 Setting the Value of the Remaining Offset of Pulses

- **\*\* int** ETM\_SET\_AO (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long** data)

### Description:

This function sets the number of remaining offset pulses for the assigned axes. Please refer to the figure below for a definition of the remaining offset pulse value.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

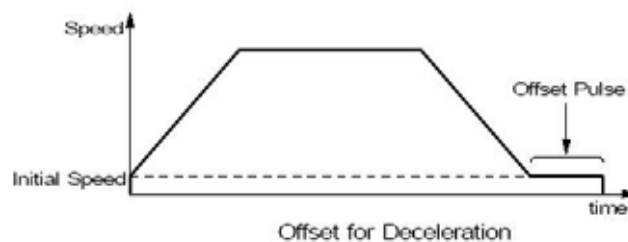
**data:** The number of remaining offset pulses. (-32,768 ~ +32,767).

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_AO (s, 1, AXIS_X, 200);  
//set the number of remaining offset pulses for the X axis on  
//module 1 to 200 pulses.
```



The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 4D		Sub_funciton code			
1		00 01		axis (1 = AXIS_X)			
2		00 00		MSW of data			

3	00 C8	<i>LSW of data (200 = 0xC8)</i>
---	-------	---------------------------------

### 6.1.9 Fixed Pulse Output

- **\*\* int** ETM\_FIXED\_MOVE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long** data)

#### Description:

Command a point-to-point motion for Fixed Position movement.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** Pulses (-2,000,000,000 ~ +2,000,000,000).

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_FIXED_MOVE (S, cardNo, AXIS_XYZU, 10000);  
// AXIS_XYZU move 10000 Pulses
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 4E		Sub_funciton code			
1		00 0F		axis (0xF = AXIS_XYZU)			
2		00 00		MSW of data			
3		27 10		LSW of data (10000 = 0x2710)			

- **\*\* int ETM\_SET\_PULSE (SOCKET s, BYTE cardNo, BYTE axis, DWORD data)**

**Description:**

This command enables to make a change of pulse during outputting fixed pulses on each axis (but not for change of the direction).

**Category:**

MODBUS sub\_function; RTC, MP and ISR.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** Pulses (0 ~ +2,000,000,000).

**Return:**

0: Success; Others: Fail (Please refer to Section 8.3)

**Sample:**

```
ETM_SET_PULSE (s, cardNo, AXIS_XYZU, 9000);
//Change the total pulses to 9000 pulses during fixed pulse moving.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 4F		Sub_function code			
1		00 0F		axis (0xF = AXIS_XYZU)			
2		00 00		MSW of data			
3		23 28		LSW of data (9000 = 0x2328)			

## 6.1.10 Continue Pulse Output

- **\*\* int** ETM\_CONTINUE\_MOVE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long** data)

### Description:

This function is continuous motion command for several independent axes.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

**data:** The specified speed (positive value for CW motion; negative value for CCW motion)

### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the errors.

### Sample:

```
ETM_CONTINUE_MOVE (s, cardNo, AXIS_XYZU, 1000);  
//continue to move at the speed of 1K PPS
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 50		Sub_function code			
1		00 0F		axis (0xF = AXIS_XYZU)			
2		00 00		MSW of data			
3		03 E8		LSW of data (1000 = 0x3E8)			



## 6.2 Interpolation Commands

### 6.2.1 Assigning the Axes for Interpolation

- **\*\* int** ETM\_AXIS\_ASSIGN (**SOCKET** s, **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **BYTE** axis3)

#### Description:

This function assigns the axes to be used for interpolation. Either two or three axes can be assigned by using this function. Interpolation commands will refer to the assigned axes to construct a working coordinate system. The X axis does not necessarily have to be the first axis. However, it is easier to use the X axis as the first axis, the Y axis as the second axis, and the Z axis as the third axis.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis1:** The first axis; can be either X, Y, Z, or U(1, 2, 4 or 8). Please refer to Table 2-1 for the axis definition.

**axis2:** The second axis; can be either X, Y, Z, or U(1, 2, 4 or 8).

**axis3:** The third axis; can be either X, Y, Z, or U(1, 2, 4, 8). Without 3rd axis the value is 0.

#### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

#### Sample:

```
ETM_AXIS_ASSIGN (s, 1, AXIS_X, AXIS_Y, 0);  
//set the X axis of module 1 as the first axis and the Y axis as the second axis.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 5A		Sub_function code			
1		00 01		axis1 (axis1 = AXIS_X)			
2		00 02		axis2 (axis1 = AXIS_Y)			

3	00 00	<i>Axis3 (not defined)</i>
---	-------	----------------------------

## 6.2.2 Setting the Speed and Acc/Dec Mode

- **\*\* int** ETM\_VECTOR\_SPEED (**SOCKET** s, **BYTE** cardNo, **BYTE** nMode)

### Description:

This function sets vector acceleration or deceleration mode.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

<b>s:</b>	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<b>cardNo:</b>	Module number
<b>nMode:</b>	<p>0 → 2-axis linear or circular motion at a constant vector speed (Set VV and VSV; and VV=VSV).</p> <p>1 → 2-axis linear motion using a symmetric T-curve velocity profile (set VSV, VV, VA, and VAO)</p> <p>2 → 2-axis linear motion using a symmetric S-curve velocity profile (set VSV, VV, VK, and VAO)</p> <p>3 → 2-axis linear motion using an asymmetric T-curve velocity profile (set VSV, VV, VA, VD, and VAO)</p> <p>4 → 2-axis linear motion using an asymmetric S-curve velocity profile (set VSV, VV, VK, VL, and VAO)</p> <p>5 → 2-axis circular motion using a symmetric T-curve velocity profile (set VSV, VV, VA, and VAO)</p> <p>6 → 2-axis circular motion using an asymmetric T-curve velocity profile (set VSV, VV, VA, VD, and VAO)</p> <p>7 → 3-axis linear motion at a constant vector speed set VV and VSV; and VV=VSV)</p> <p>8 → 3-axis linear motion at using a symmetric T-curve velocity profile (set VSV, VV, VA, and VAO)</p> <p>9 → 3-axis linear motion using a symmetric S-curve velocity profile (set VSV, VV, VK, and VAO)</p> <p>10 → 3-axis linear motion using an asymmetric T-curve velocity profile (set VSV, VV, VA, VD, and VAO)</p> <p>11 → 3-axis linear motion using an asymmetric S-curve velocity profile (set VSV, VV, VK, VL, and VAO)</p>

**Note:** Please refer the configurations of speed-related parameters.

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_VECTOR_SPEED (s, cardNo, 0);
//set the module to perform 2-axis linear or circular motion
//at a constant vector speed.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 5B		Sub_funciton code			
1		00 00		nMode			

### 6.2.3 Setting the Vector Starting Speed

- **\*\* int** ETM\_SET\_VSV (**SOCKET** s, **BYTE** cardNo, **DWORD** data)

#### Description:

This function sets the starting speed of the principle axis (axis 1) for the interpolation motion.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**data:** The vector starting speed value (in PPS) (For maximum value please refer to Table 2-5).

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_SET_VSV (s, 1, 1000);  
//set the starting speed of the axis 1 for the interpolation motion  
//on module 1 to 1000 PPS.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 5C		Sub_funciton code			
1		00 00		MSW of data			
2		03 E8		LSW of data (1000 = 0x3E8)			

## 6.2.4 Setting the Vector Speed

- **\*\* int** ETM\_SET\_VV (**SOCKET** s, **BYTE** cardNo, **DWORD** data)

### Description:

This function sets the vector speed of the interpolation motion. Users do not need to assign any axes on this function. The speed setting will take effect on the current working coordinate system which is defined by the ETM\_AXIS\_ASSIGN() function.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**data:** The vector speed value (in PPS) (For maximum value please refer to Table 2-5).

### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

### Sample:

```
ETM_SET_VV (s, 1, 120000L);  
//set the vector speed of the interpolation on module 1  
//to 120000 PPS.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 5D		Sub_function code			
1		00 01		MSW of data			
2		D4 C0		LSW of data (120000 = 0x1D4C0)			

## 6.2.5 Setting the Vector Acceleration

- **\*\* int** ETM\_SET\_VA (**SOCKET** s, **BYTE** cardNo, **DWORD** data)

### Description:

This function sets the vector acceleration of the interpolation motion. Users do not need to assign any axes on this function. The speed setting will take effect on the current working coordinate system which is defined by the ETM\_AXIS\_ASSIGN() function.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**data:** The vector acceleration value (in PPS/Sec). This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V () function. The maximum available acceleration value is MAX\_V \* **125**. The minimum acceleration value is MAX\_V ÷ **64**, and all other acceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_VA (s, 1, 100000L);  
//set the vector acceleration of the interpolation motion  
//on module 1 to 100K PPS/Sec.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 5E		Sub_funciton code			
1		00 01		MSW of data			
2		86 A0		LSW of data (100000 = 0x186A0)			

## 6.2.6 Setting the Vector Deceleration Value

- **\*\* int** ETM\_SET\_VD (**SOCKET** s, **BYTE** cardNo, **DWORD** data)

### Description:

This function sets the vector deceleration of the interpolation motion.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**data:** The vector deceleration value (in PPS/Sec). This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V () function. The maximum available acceleration value is MAX\_V \* **125**. The minimum acceleration value is MAX\_V ÷ **64**, and all other acceleration values are the integral multipliers of this value. The practical value for application depends on the capability of the motor drive and motor.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_VD (s, 1, 100000L);  
//set the vector deceleration value of interpolation motion  
//on module 1 to 100K PPS/Sec.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 5F		Sub_funciton code			
1		00 01		MSW of data			
2		86 A0		LSW of data (100000 = 0x186A0)			



## 6.2.7 Setting the Vector Acceleration Rate

- **\*\* int** ETM\_SET\_VK (**SOCKET** s, **BYTE** cardNo, **DWORD** data)

### Description:

Set the acceleration rate (jerk) value for interpolation motion.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**data:** The acceleration rate (jerk) value(PPS/ Sec<sup>2</sup>).  
This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V () function.  
The minimum acceleration rate is MAX\_V \* **0.0119211**;  
The maximum acceleration rate value is **2,000,000,000**.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_VK (s, 1, 10000);  
//set the acceleration rate of the interpolation motion on module 1  
// to 10,000 PPS/ Sec^2.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 60		Sub_funciton code			
1		00 00		MSW of data			
2		27 10		LSW of data (10000 = 0x2710)			

## 6.2.8 Setting the Vector Deceleration Rate

- **\*\* int** ETM\_SET\_VL (**SOCKET** s, **BYTE** cardNo, **DWORD** data)

### Description:

Set the deceleration rate of the interpolation motion.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**data:** The deceleration rate (jerk) value(PPS/ Sec<sup>2</sup>).  
This value is related to the maximum speed value defined by ETM\_SET\_MAX\_V () function.  
The minimum acceleration rate is MAX\_V \* **0.0119211**;  
The maximum acceleration rate value is **2,000,000,000**.

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_SET_VL (s, 1, 10000);  
//set the deceleration rate of the interpolation on module 1 to 10,000  
//PPS/Sec^2.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 61		Sub_funciton code			
1		00 00		MSW of data			
2		27 10		LSW of data (10000 = 0x2710)			

## 6.2.9 Setting the Number of Remaining Offset Pulses

- **\*\* int** ETM\_SET\_VAO (**SOCKET** s, **BYTE** cardNo, **long** data)

### Description:

Setting this value will cause the motion control chip to start deceleration earlier. The remaining offset pulses will be completed at low speed to allow the controller to stop immediately when it reaches the offset pulse value. Please refer to the figure below for more information.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or axes (Please refer to Table 2-1)

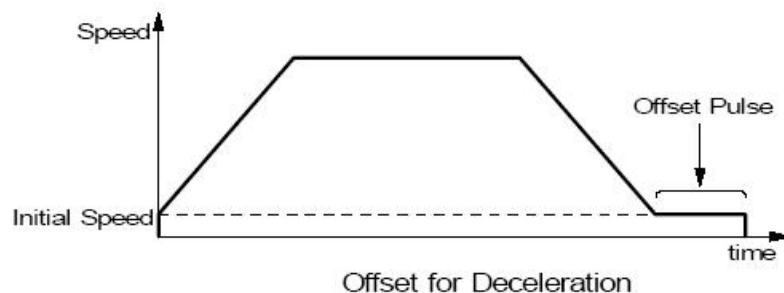
**data:** The number of remaining offset pulses. (-32,768 ~ +32,767).

### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

### Sample:

```
ETM_SET_VAO (s, 1, 200);  
//set the number of remaining offset pulse value on module 1 to 200.
```



The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A 62		Sub_funciton code			

1	00 00	<i>MSW of data</i>
2	00 C8	<i>LSW of data (200 = 0xC8)</i>

## 6.2.10 2-Axis Interpolation Motion

- **\*\* int** ETM\_LINE\_2D (**SOCKET** s, **BYTE** cardNo, **long** fp1, **long** fp2)

### Description:

This function executes a 2-axis linear interpolation motion.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**fp1:** The displacement of the axis 1 in Pulses  
(-2,000,000,000 ~ +2,000,000,000)

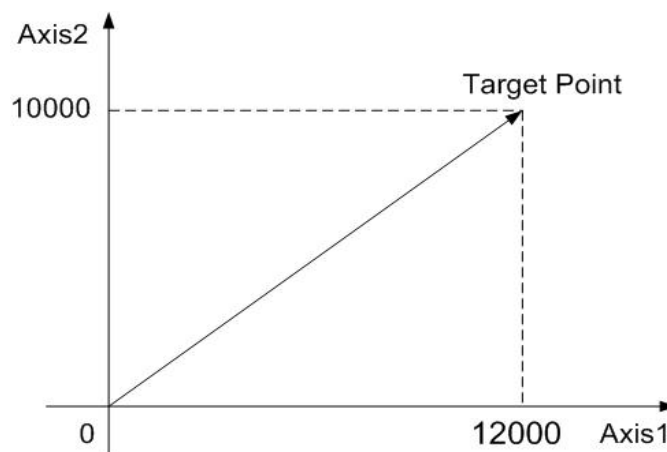
**fp2:** The displacement of the axis 2 in Pulses  
(-2,000,000,000 ~ +2,000,000,000)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_LINE_2D (s, 1, 12000, 10000);  
//execute the 2-axis linear interpolation motion on module 1.
```



2-axis linear interpolation motion

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			

0	0A 63	<i>Sub_funciton code</i>
1	00 00	<i>MSW of fp1</i>
2	2E E0	<i>LSW of fp1 (12000 = 0x2EE0)</i>
3	00 00	<i>MSW of fp2</i>
4	27 10	<i>LSW of fp2 (10000 = 0x2710)</i>

## 6.2.11 3-Axis Interpolation Motion

- **\*\* BYTE** ETM\_LINE\_3D (**SOCKET** s, **BYTE** cardNo, **long** fp1, **long** fp2, **long** fp3)

### Description:

This function executes a 3-axis linear interpolation motion.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

- s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).
- cardNo:** Module number
- fp1:** The displacement of the first axis (axis 1) in Pulses (-2,000,000,000 ~ +2,000,000,000)
- fp2:** The displacement of the second axis (axis 2) in Pulses (-2,000,000,000 ~ +2,000,000,000)
- fp3:** The displacement of the third axis (axis 3) in Pulses (-2,000,000,000 ~ +2,000,000,000)

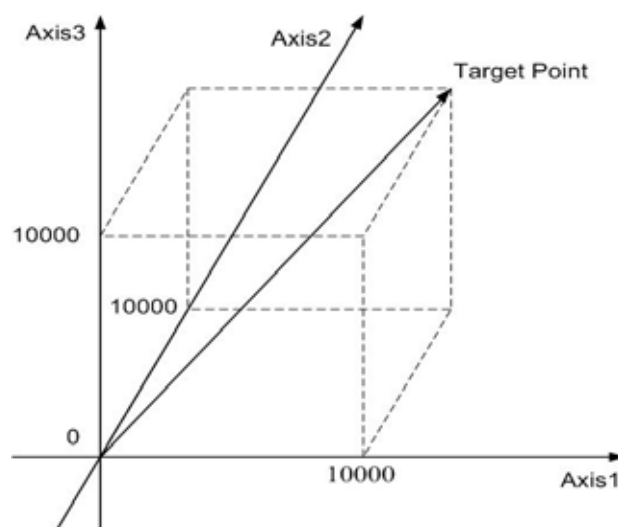
### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the errors.

### Sample:

```
ETM_LINE_3D (s, 1, 10000, 10000, 10000);  
//execute the 3-axis linear interpolation motion on module 1.
```



3-axis linear interpolation motion

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 15	01	10	1F 40	00 07	0E
Register[]		Value (hex)		Remarks			
0		0A 64		Sub_funciton code			
1		00 00		MSW of fp1			
2		27 10		LSW of fp1 (10000 = 0x2710)			
3		00 00		MSW of fp2			
4		27 10		LSW of fp2 (10000 = 0x2710)			
5		00 00		MSW of fp3			
6		27 10		LSW of fp3 (10000 = 0x2710)			



## 6.2.12 2-Axis Circular Interpolation Motion (an Arc)

- **\*\* int** ETM\_ARC\_CW (**SOCKET** *s*, **BYTE** *cardNo*, **long** *cp1*, **long** *cp2*, **long** *fp1*, **long** *fp2*)

### Description:

This function executes a 2-axis circular interpolation motion in a clockwise (CW) direction.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

<i>s</i> :	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<i>cardNo</i> :	Module number
<i>cp1</i> :	The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).
<i>cp2</i> :	The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).
<i>fp1</i> :	The displacement of the axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)
<i>fp2</i> :	The displacement of the axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)

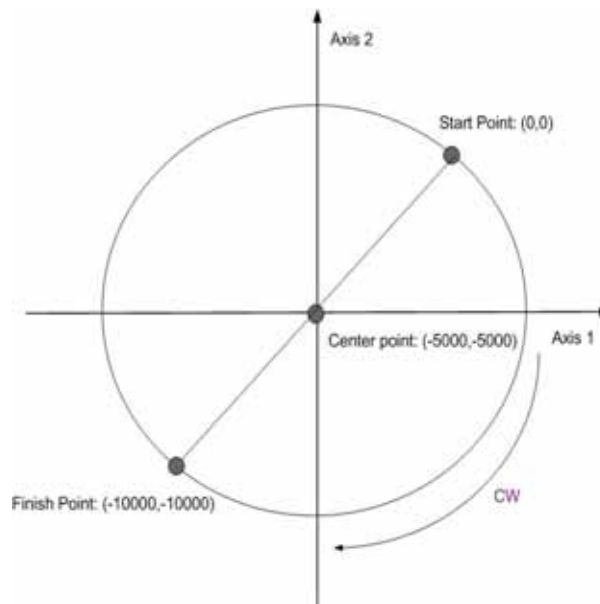
### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the errors.

### Sample:

```
ETM_ARC_CW (s, 1, -5000, -5000, -10000, -10000);  
// Issues a command to perform a circular motion (an arc)  
// in a CW direction. Please refer to the following figure.
```



2-axis circular motion in a CW direction

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 19	01	10	1F 40	00 09	12
Register[]		Value (hex)		Remarks			
0		0A 65		Sub_funciton code			
1		FF FF		MSW of cp1			
2		EC 78		LSW of cp1 (-5000 = 0xFFFFEC78)			
3		FF FF		MSW of cp2			
4		EC 78		LSW of cp2 (-5000 = 0xFFFFEC78)			
5		FF FF		MSW of fp1			
6		D8 F0		LSW of fp1 (-10000 = 0xFFFFD8F0)			
7		FF FF		MSW of fp2			
8		D8 F0		LSW of fp2 (-10000 = 0xFFFFD8F0)			

- **\*\* int ETM\_ARC\_CCW (SOCKET s, BYTE cardNo, long cp1, long cp2, long fp1, long fp2)**

**Description:**

This function executes a 2-axis circular interpolation motion in a counter-clockwise (CCW) direction.

**Category:**

MODBUS sub\_function; RTC, MP and ISR.

**Parameters:**

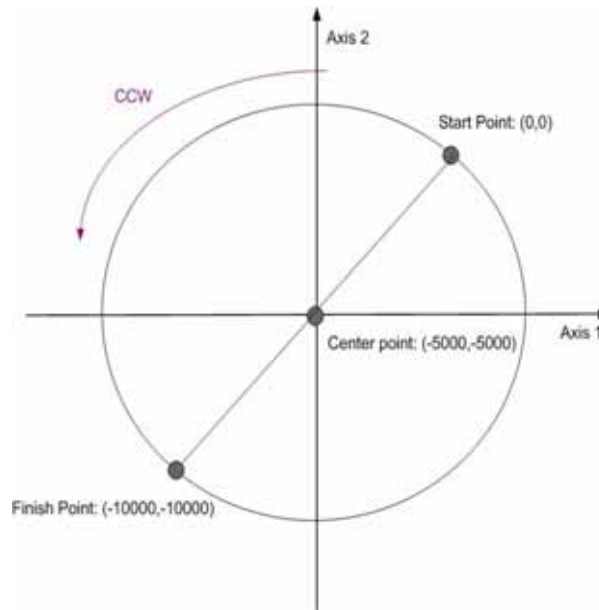
<b>s:</b>	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<b>cardNo:</b>	Module number
<b>cp1:</b>	The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).
<b>cp2:</b>	The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).
<b>fp1:</b>	The displacement of the axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)
<b>fp2:</b>	The displacement of the axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)  
Use the ETM\_GET\_ERROR\_CODE() function to identify the errors.

**Sample:**

```
ETM_ARC_CCW (s, 1, -5000, -5000, -10000, -10000);
// Issues a command to perform a circular motion (an arc)
// in a CCW direction. Please refer to the following figure.
```



2-axis circular motion in a CW direction

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 19	01	10	1F 40	00 09	12
Register[]		Value (hex)		Remarks			
0		0A 67		Sub_funciton code			
1		FF FF		MSW of cp1			
2		EC 78		LSW of cp1 (-5000 = 0xFFFFEC78)			
3		FF FF		MSW of cp2			
4		EC 78		LSW of cp2 (-5000 = 0xFFFFEC78)			
5		FF FF		MSW of fp1			
6		D8 F0		LSW of fp1 (-10000 = 0xFFFFD8F0)			
7		FF FF		MSW of fp2			
8		D8 F0		LSW of fp2 (-10000 = 0xFFFFD8F0)			

### 6.2.13 2-Axis Circular Interpolation Motion (an Complete Circle)

- **\*\* int** ETM\_CIRCLE\_CW (**SOCKET** s, **BYTE** cardNo, **long** cp1, **long** cp2)

#### Description:

This function executes a 2-axis circular interpolation motion in a clockwise (CW) direction.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**cp1:** The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).

**cp2:** The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the errors.

#### Sample:

```
ETM_CIRCLE_CW (s, 1, 0, 10000);
```

```
//execute a circular motion (a complete circle) in a CW direction on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			
0		0A 69		Sub_funciton code			
1		00 00		MSW of cp1			
2		00 00		LSW of cp1 (0 = 0x0)			
3		00 00		MSW of cp2			
4		27 10		LSW of cp2 (10000 = 0x2710)			

- **\*\* int ETM\_CIRCLE\_CCW (SOCKET s, BYTE cardNo, long cp1, long cp2)**

#### Description:

This function executes a 2-axis circular interpolation motion in a counter-clockwise (CCW) direction.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**cp1:** The relative position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).

**cp2:** The relative position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)  
Use the ETM\_GET\_ERROR\_CODE() function to identify the errors.

#### Sample:

```
ETM_CIRCLE_CCW (s, 1, 0, 10000);
// execute a circular motion (a complete circle) in a CCW direction
// on module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			
0		0A 6A		Sub_funciton code			
1		00 00		MSW of cp1			
2		00 00		LSW of cp1 (0 = 0x0)			
3		00 00		MSW of cp2			
4		27 10		LSW of cp2 (10000 = 0x2710)			

## 6.3 Synchronous Actions

### 6.3.1 Setting the Synchronous Action

- **\*\* int** ETM\_SYNC\_ACTION (**SOCKET** s,  
    **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **DWORD** nSYNC,  
    **BYTE** nDRV, **BYTE** nLATCH, **BYTE** nPRESET, **BYTE** nOUT,  
    **BYTE** nINT, **BYTE** isrNoX, **BYTE** isrNoY, **BYTE** isrNoZ, **BYTE** isrNoU)

#### Description:

This function sets the activation factors for synchronous action. (It's total-hardware-solution, will not consume any WinCon or I8000 system resources).

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

- s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).
- cardNo:** Module number
- axis1:** This is the monitored axis. It will be checked by hardware. The axis can be either X, Y, Z, or U (1 or 2 or 4 or 8). (Please refer to Table 2-1.)
- axis2:** This defined the other axes (or axis) that will take action when one of the activation factors occurs. The axes are defined in the following table.

<b>axis1</b> <b>axis2</b>	<b>X</b>	<b>Y</b>	<b>Z</b>	<b>U</b>
<b>0</b>	X	Y	Z	U
<b>1</b>	Y	Z	U	X
<b>2</b>	Z	U	X	Y
<b>3</b>	YZ	ZU	UX	XY
<b>4</b>	U	X	Y	Z
<b>5</b>	YU	ZX	UY	XZ
<b>6</b>	ZU	UX	XY	YZ
<b>7</b>	YZU	ZUX	UXY	XYZ

If axis2 is set as 0, it means the synchronous actions axis and moving axis are the same axis.

- nSYNC:** It defines the activation factors. Multiple activation factors can be defined at the same time. Available active factors are listed in the following table.

Value	Event	Explanation
0x00000000		Disable the synchronous action
0x00000001	$P \geq C+$	The logical/real position counter value exceeded the COMP+ register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2).
0x00000002	$P < C+$	The logical/real position counter value became less than the COMP+ register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2).
0x00000004	$P < C-$	The logical/real position counter value became less than the COMP- register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2).
0x00000008	$P \geq C-$	The logical/real position counter value exceeded the COMP- register value. Use the ETM_SET_COMPARE () function for selection of a logical/real position(6.3.2).
0x00000010	D-STA	Driving started.
0x00000020	D-END	Driving terminated.
0x00000040	IN3↑	The nIN3 signal rose from the Low to the High level.
0x00000080	IN3↓	The nIN3 signal fell from the High to the Low level.

**nDRV:** It defines the actions that are related with axial driving.  
Available actions are listed in the following table. Only one driving action can be chosen.

Value	Symbol	Explanation
0		Disable driving action.
1	FDRV+	Activates fixed pulse driving in the + direction. It must set the nPRESET value to be "OPSET" which indicates that ETM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, ETM_SET_PRESET(), is necessary. However, this command does not take effect if the assigned axes, axis2, are moving.
2	FDRV-	Activates fixed pulse driving in the - direction. It must set the nPRESET value to be "OPSET" which indicates that ETM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, ETM_SET_PRESET(), is necessary. However, this command does not take effect if the assigned axes, axis2, are moving.
3	CDRV+	Activates continuous pulse driving in the + direction. However, this command does not take effect if the assigned axes, axis2, are moving
4	CDRV-	Activates continuous pulse driving in the - direction. However, this command does not take effect if the assigned axes, axis2, are moving.
5	SSTOP	Stop driving in deceleration



6	ISTOP	Stop driving immediately
---	-------	--------------------------

**nLATCH:** It defines the actions that is related of latching position.  
Available actions are listed in the following table. Only one of these actions can be chosen

Value	Symbol	Explanation
0		Disable position latch function
1	LPSAV	Saves the current logical position counter value (LP) in the synchronous buffer register (BR). [LP → LATCH]
2	EPSAV	Saves the current real position counter value (EP) in the synchronous buffer register (BR). [EP → LATCH]

After the event is occurred, the ETM\_GET\_LATCH() function can be use to get the latched value.

**nPRESET:** It defines the actions that is related of latching position.  
Available actions are listed in the following table. Only one of these actions can be chosen.

Value	Symbol	Explanation
0		Disable setting function
1	LPSET	Indicates that a new value for the logical position (LP) will be set. The new value will be set by ETM_SET_PRESET() function. [LP ← PRESET]
2	EPSET	Indicates that a new value for the real position (EP) will be set. The new value will be set by ETM_SET_PRESET() function. [EP ← PRESET]
3	OPSET	Indicates that a new offset value (P) for the fixed pulse driving will be set. The new value will be set by ETM_SET_PRESET() function. [P ← PRESET] This setting is invalid to the Finish-Point of the axes in CONTINUE_MOVE.
4	VLSET	Indicates that a new speed value (V) will be set. The new value will be set by ETM_SET_PRESET() function. [V ← PRESET]

Must be used with ETM\_SET\_PRESET together, please refer to section 6.3.4.

**nOUT:** setting trigger output, as the following table:

Value	Symbol	Explanation
0		Disable trigger output
1	OUT	Enable trigger output

Must be used with ETM\_SET\_OUT together, please refer to section 6.3.5.

**nINT:** setting interrupt function, as the following table:

Value	Symbol	Explanation
0		Disable interrupt function
1	INT	Enable interrupt function

1. It will generate a interrupt for the synchronous action of axis2, please write the corresponding number: *isrNoX* 、 *isrNoY* 、 *isrNoZ* 、 *isrNoU*

2. Must be used with ETM\_ENABLE\_INT together, please refer to section 6.3.6.

*isrNoX:* ISR1 ~ ISR20 : Specify the interrupt number of **X-axis**, please refer to the section 7.2.1.

0: **disable**

*isrNoY:* ISR1 ~ ISR20 : Specify the interrupt number of **Y-axis**, please refer to the section 7.2.1.

0: **disable**

*isrNoZ:* ISR1 ~ ISR20 : Specify the interrupt number of **Z-axis**, please refer to the section 7.2.1.

0: **disable**

*isrNoU:* ISR1 ~ ISR20 : Specify the interrupt number of **U-axis**, please refer to the section 7.2.1.

0: **disable**

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

ETM\_SYNC\_ACTION (s, cardNo, AXIS\_U, 0, 0X00000040, 0, 2, 4, 0, 0, 0, 0, 0, 0);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 23	01	10	1F 40	00 0E	1C
Register[]		Value (hex)		Remarks			
0		0A 6E		<i>Sub_funciton code</i>			
1		00 08		<i>axis1 (8 = AXIS_U)</i>			
2		00 00		<i>axis2</i>			
3		00 00		<i>MSW of nSYNC</i>			
4		00 40		<i>LSW of nSYNC (0x40 → rising edge of IN3 will trigger the action)</i>			
5		00 00		<i>nDRV (0 → no driving control)</i>			
6		00 02		<i>nLATCH (2 → will latch EP value)</i>			
7		00 04		<i>nPRESET (4 → set new velocity when SYNC condition is true)</i>			
8		00 00		<i>nOUT (0 → no pulse out)</i>			
9		00 00		<i>nINT (0 → disable)</i>			
10		00 00		<i>isrNoX (It can be any value since nINT is disable.)</i>			

11	00 00	<i>isrNoY (It can be any value since nINT is disable.)</i>
12	00 00	<i>isrNoZ (It can be any value since nINT is disable.)</i>
13	00 00	<i>isrNoU (It can be any value since nINT is disable.)</i>

### 6.3.2 Setting the COMPARE Value

- **\*\* int** ETM\_SET\_COMPARE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nSELECT, **BYTE** nTYPE, **long** data)

#### Description:

This function sets the values of COMPARE registers. **However, it will disable the functions of software limits.**

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**nSELECT:** 0 → C+  
1 → C-

**nTYPE:** 0 → Position(P) = LP  
1 → Position(P) = EP

**data:** Set the COMPARE value (-2,000,000,000 ~ +2,000,000,000).

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_SET_COMPARE (s, cardNo, AXIS_U, 0, 1, 5000);  
//Set the comparison function for U-Axis.  
//Set the compared source to be EP; and the COMP+ value to 5000.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 13	01	10	1F 40	00 06	0C
Register[]		Value (hex)		Remarks			
0		0A 70		Sub_funciton code			
1		00 08		Axis (8 → AXIS_U)			
2		00 00		nSELECT			
3		00 01		nTYPE			
4		00 00		MSW of data			
5		13 88		LSW of data (5000 = 0x1388)			

### 6.3.3 Get the LATCH Value

- **\*\* int** ETM\_GET\_LATCH (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)
- **\*\*long** ETM04\_GET\_LATCH (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long\*** LatchValue)

#### Description:

This function gets the values from the LATCH register.

#### Category:

MODBUS table, MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

- s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).
- cardNo:** Module number
- axis:** The axis can be either X, Y, Z, or U (1 or 2 or 4 or 8). Please refer to Table 2-1.
- LatchValue:** The pointer to the memory that stores the value of LATCH.  
The value of LATCH : -2,000,000,000 ~ +2,000,000,000

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

- Method 1: It can get latched values directly.

The latched values are always long type values. Therefore, polling the latched values must start at the MSW of each axis's MSW.

STOP status	Address	Remarks
X_LATCH_MSW	80 (0x50)	MSW of X_LATCH
X_LATCH_LSW	81 (0x51)	LSW of X_LATCH
Y_LATCH_MSW	82 (0x52)	MSW of Y_LATCH
Y_LATCH_LSW	83 (0x53)	LSW of Y_LATCH
Z_LATCH_MSW	84 (0x54)	MSW of Z_LATCH
Z_LATCH_LSW	85 (0x55)	LSW of Z_LATCH
U_LATCH_MSW	86 (0x56)	MSW of U_LATCH
U_LATCH_LSW	87 (0x57)	LSW of U_LATCH

```
long LATCH_Y;  
ETM04_GET_LATCH (s, 1, AXIS_Y, &LATCH_Y);
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	04	00 52	00 02

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	MSW of Y_LATCH ( Reg_0 )	LSW of Y_LATCH ( Reg_1 )
00 01	00 00	00 07	01	04	04	00 00	03 E8

LATCH\_Y = Register[0];

LATCH\_Y = (long) ( ( LATCH\_Y << 16 ) & 0xffff0000 ) | ( Register[1] & 0xffff );

- Method 2: It can be used inside a MP program.

For this case, users do not actually want to get the current latched values. The getting latched values will be executed only when the MP is called. Therefore, use **FC = 16** to write this command inside a MP. This kind of usage often has **ETM\_MP\_SET\_RVAR()** followed to save the return latched value. Please refer to MP related explanation literature.

**ETM\_GET\_LATCH** (s, 1, AXIS\_Y);

//Get the latched value which is from Y-axis of card 1.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A 71		Sub_funciton code			
1		00 02		axis			

### 6.3.4 Set the PRESET data for synchronous action

- **\*\* int** ETM\_SET\_PRESET (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long** data)

#### Description:

This function sets the PRESET value for synchronous action(Each synchronous action axis could not be set individually.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** The axis must be assigned as the **axis2** parameter of ETM\_SYNC\_ACTION( ).

**data:** LP: (-2,000,000,000 ~ +2,000,000,000)  
EP: (-2,000,000,000 ~ +2,000,000,000)  
P : (-2,000,000,000 ~ +2,000,000,000)  
V : Please refer to section 2.5. If there are more than two synchronous action axes, please set ETM\_SET\_MAX\_V to be same value.

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

If the SYNC action is set to change velocity, then following statement will change the velocity of AXIS\_U to 100 PPS when the condition is true.

```
ETM_SET_PRESET (s, cardNo, AXIS_U, 100);
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A 72		Sub_funciton code			
1		00 08		Axis (8 → AXIS_U)			
2		00 00		MSW of data			
3		00 64		LSW of data (100 = 0x64)			

### 6.3.5 Set the OUT Data

- **\*\* int** ETM\_SET\_OUT (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** outEdge, **BYTE** PulseWidth)

#### Description:

This function sets the output pulse settings.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or Axes (Please refer to Table 2-1 for the axis definition.), **currently only support Axes X & Y.**

**outEdge:** trigger OUT active logic:  
0 = low active; 1 = high active

**PulseWidth:** trigger OUT output pulse width  
0 = 10 uSec  
1 = 20 uSec  
2 = 100 uSec  
3 = 200 uSec  
4 = 1,000 uSec  
5 = 2,000 uSec  
6 = 10,000 uSec  
7 = 20,000 uSec

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

If the SYNC action enables the digital OUT function, then following statement can define the waveform of digital output that includes the level and the pulse width.

```
ETM_SET_OUT (s, 1, AXIS_U, 1, 0);
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08



Register[]	Value (hex)	Remarks
0	0A 73	<i>Sub_funciton code</i>
1	00 08	<i>Axis (8 → AXIS_U)</i>
2	00 01	<i>outEdge</i>
3	00 00	<i>PulseWidth</i>

### 6.3.6 The nterrupt Function of motion module (i-8094H)

- \* **int** ETM\_ENABLE\_INT (**SOCKET** s, **BYTE** cardNo)

#### Description:

This function enables the interrupt function.

#### Category:

MODBUS sub\_function; RTC, MP.

#### Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_ENABLE_INT (s, 1);  
//Enable the interrupt function for module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A AA		Sub_funciton code			

● **\* int ETM\_DISABLE\_INT (SOCKET s, BYTE cardNo)**

**Description:**

This function disables the interrupt function.

**Category:**

MODBUS sub\_function; RTC, MP.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_DISABLE_INT (s, 1);  
//Disable the interrupt function for module 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A AB		Sub_funciton code			

- **\*\* int** ETM\_INTFACTOR\_ENABLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nINT, **BYTE** isrNo)

**Description:**

This function sets the condition factor of interrupt.

**Category:**

MODBUS sub\_function; RTC, MP, ISR.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or Axes (Please refer to Table 2-1 for the axis definition)

**nINT:** condition factors of interrupt, please refer to the following table

Number	Symbol	Description
0	PULSE	Interrupt occurs when pulse is up
1	P>=C-	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register. <b>Please refer to ETM_SET_COMPARE() (6.3.2)</b>
2	P<C-	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register <b>Please refer to ETM_SET_COMPARE() (6.3.2)</b>
3	P<C+	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register. <b>Please refer to ETM_SET_COMPARE() (6.3.2)</b>
4	P>=C+	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register. <b>Please refer to ETM_SET_COMPARE() (6.3.2)</b>
5	C-END	Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output
6	C-STA	Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output
7	D-END	Interrupt occurs when the driving is finished

**isrNo:** ISR1 ~ ISR20 : Specify the number of interrupt service, please refer to section [\(7.2.1\)](#)

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Note:** This function is conflict with ETM\_SET\_SLMT[\(2.10\)](#).

**Sample:**

This function assigns an ISRn to process an interrupt that happens at a specified axis.

ETM\_INTFACTOR\_ENABLE (s, 1, AXIS\_X, 4, ISR1);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0F	01	10	1F 40	00 04	08
Register[]		Value (hex)		Remarks			
0		0A AC		Sub_funciton code			
1		00 01		axis (1 → AXIS_X)			
2		00 04		nINT			
3		00 01		isrNo (ISR1 = 1)			

- **\*\* int** ETM\_INTFACTOR\_DISABLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** nINT)

**Description:**

This function disables the condition factor of interrupt.

**Category:**

MODBUS sub\_function; RTC, MP, ISR.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or Axes (Please refer to Table 2-1 for the axis definition).

**nINT:** condition factors of interrupt, please refer to the following table

Number	Symbol	Description
0	PULSE	Interrupt occurs when pulse is up
1	P>=C-	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register.
2	P<C-	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register
3	P<C+	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register.
4	P>=C+	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register.
5	C-END	Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output
6	C-STA	Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output
7	D-END	Interrupt occurs when the driving is finished

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

This function assigns an ISRn to process an interrupt that happens at a specified axis.

```
ETM_INTFACTOR_DISABLE (s, 1, AXIS_XYZU, 4);
// Disable the interrupt factors of Module 1
// 4 : Interrupt occurs when the value of logic / real position counter
// is larger than or equal to that of COMP+ register.
```

The MODBUS command is listed as follows:

TID	PID	Field	UID	FC	St_Addr.	Word	Byte
-----	-----	-------	-----	----	----------	------	------

ICP DAS

4

ET-M8194H User Manual – API Library  
Version 1.0 2010/02/22

(hex)	(hex)	Length (hex)	(hex)	(hex)	(hex)	Count (hex)	Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A AD		<i>Sub_funciton code</i>			
1		00 0F		<i>axis (0xF → AXIS_XYZU)</i>			
2		00 04		<i>nINT</i>			

### 6.3.7 The Interrupt of i-8094H for controller system

- **int** ETM\_ENABLE\_RINT (**SOCKET** s, **BYTE** cardNo)

#### Description:

This function enables the interrupt function of i-8094H for controller system. When this function is called via MODBUS, the ET-M8194H will clear the related status, and then wait for the input of RINT. The real input values are listed as follows:

Address	Description (MODBUS table, read with FC=4)
60	Line Scan finished, ( please refer to the section 6.4.8)
61	Macro Program finished, (please refer to section 7.2.2)
62	User defined RINT finished, ( please refer to section 7.2.8)
63	being interrupted when execute a continuous interpolation .
64	0
65	Reserved
66	Axes Error , (please use with the ETM_GET_ERROR_CODE in section 3.6)
67	Module Error , (please use when reset module, please refer to section 2.3)
4	The Bit0 ~ bit7of this register are mapped to address 60 ~ 67.

#### Category:

MODBUS sub\_function; RTC.

#### Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_ENABLE_RINT (s);  
//Enable Card-1 the interrupt function of i-8094H for controller system
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A AE		Sub_funciton code			



● **int** ETM\_DISABLE\_RINT (**SOCKET** s, **BYTE** cardNo)

**Description:**

This function disables the interrupt function of i-8094H for controller system. When this function is called via MODBUS, the ET-M8194H will clear the related status, and then terminate the RINT checking. The values in the following list will all be zero.

Address	Description (MODBUS table, read with FC=4)
60	Line Scan finished, ( please refer to the section 6.4.8)
61	Macro Program finished, (please refer to section 7.2.2)
62	User defined RINT finished, ( please refer to section 7.2.8)
63	being interrupted when execute a continuous interpolation .
64	0
65	Reserved
66	Axes Error , (please use with the ETM_GET_ERROR_CODE in section 3.6)
67	Module Error ,(please use when reset module, please refer to section 2.3)
4	The Bit0 ~ bit7of this register are mapped to address 60 ~ 67.

**Category:**

MODBUS sub\_function; RTC.

**Parameters:**

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_DISABLE_RINT (s);
//Disable the interrupt function of i-8094H on Card-1
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A AF		Sub_funciton code			

- **int** ETM\_GET\_RINT\_STATE\_ALL (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE\*** RINTState)

**Description:**

This function gets for the returning interrupt status.

**Category:**

MODBUS sub\_function;.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**RINTState:** The pointer to the memory that stores RINT\_STATE\_ALL state. The contents of address 60 ~ 67 are mapped into Bit0 ~ bit7 of the RINT\_STATE\_ALL. Please refer to the following table:

STOP status	Address	Remarks
Line_Scan_Completed	60 (0x3C)	1 → When line scan has completed
MP_Completed	61 (0x3D)	1 → When an MP has completed
User_Defined_RINT	62 (0x3E)	1 → When user_defined RINT is received
Continue_Inp_Interrupted	63 (0x3F)	1 → When the continuous interpolation is interrupted
... Reserved	64 (0x40)	0
... Reserved	65 (0x41)	0
Axes_Error	66 (0x42)	1 → When any axis error happens
Module_Error	67 (0x43)	1 → When other error happened
RINT_STATE_ALL	4	The contents of address 60 ~ 67 are mapped into Bit0 ~ bit7 of the associated register.

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
00 01	00 00	00 06	01	04	00 04	00 01

The response via MODBUS may be:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	Byte Count (hex)	RINT_STATE _ALL (hex)
--------------	--------------	--------------------------	--------------	-------------	------------------------	-----------------------------

00 01	00 00	00 05	01	04	02	00 00
-------	-------	-------	----	----	----	-------

User could get these states by polling and decide what to do in following program.

In addition, several ET-M8194H APIs help to get the RINT\_STATE:

```
// Read the Line_Scan_Completed state
int ETM04_GET_LINE_SCAN_DONE (SOCKET s, BYTE cardNo,
                                BYTE* LScanState);

// Read the MP_Completed state
int ETM04_GET_MP_DONE (SOCKET s, BYTE cardNo, BYTE* MPState);

// Read User_Defined_RINT state
int ETM04_GET_USER_RINT (SOCKET s, BYTE cardNo,
                           BYTE* URINTState);
```

## 6.4 Continuous Interpolation

### 6.4.1 2-Axis Rectangular Motion

- **\* int** ETM\_RECTANGLE (**SOCKET** s, **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **BYTE** nAcc, **BYTE** Sp, **BYTE** nDir, **long** Lp, **long** Wp, **long** Rp, **DWORD** RSV, **DWORD** RV, **DWORD** RA, **DWORD** RD)

#### Description:

Continuous interpolation will be performed to create a rectangular motion, which is formed by 4 lines and 4 arcs. The length of each side can be changed. The radius of each arc is the same and it can also be changed. The deceleration point will be calculated automatically. This is a command macro command that appears in various motion applications. **(It will not consume any system resources of WinCon or I8000).**

#### Category:

MODBUS sub\_function; RTC and MP.

#### Parameters:

<b>s:</b>	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<b>cardNo:</b>	Module number
<b>axis1:</b>	The first axis (axis 1). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、 2、 4、 8).
<b>axis2:</b>	The second (axis 2). Please refer to Table 2-1. The first axis and It can be either X, Y, Z, or U(1、 2、 4、 8).
<b>nAcc:</b>	0 → constant vector speed interpolation mode 1 → symmetric T-curve Acc/Dec interpolation mod
<b>Sp:</b>	Start point 0 ~ 7. (Sp0 ~ Sp7 are defined in the following.
<b>nDir:</b>	Direction of movement 0: CCW; 1: CW
<b>Lp:</b>	Length in Pulses (1 ~ 2,000,000,00)
<b>Wp:</b>	Width in Pulses (1 ~ 2,000,000,00)
<b>Rp:</b>	Radius of each in pulses (1 ~ 2,000,000,00)
<b>RSV:</b>	Starting speed (in PPS)
<b>RV:</b>	Vector speed (in PPS)
<b>RA:</b>	Acceleration (PPS/Sec)
<b>RD:</b>	Deceleration of the last segment (in PPS/Sec)

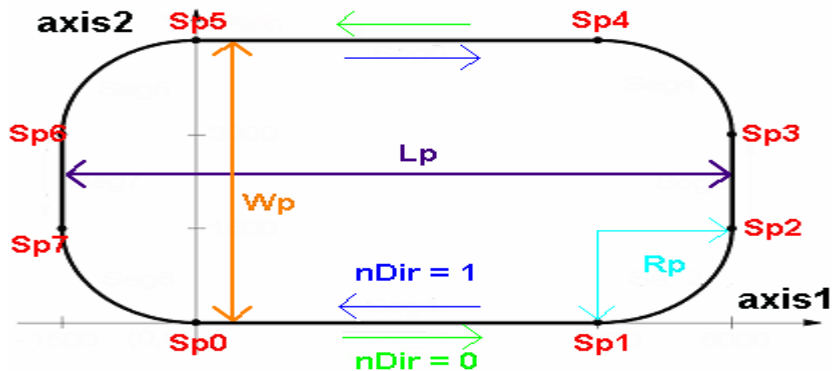
#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the error.

Sample:

```
ETM_RECTANGLE (
s, 1, AXIS_X, AXIS_Y, 1, 0, 0, 20000, 10000, 1000, 1000, 10000, 5000, 5000);
//execute a rectangular motion on XY plane, will auto-calculate
//deceleration point.
```



The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 2F	01	10	1F 40	00 14	28
Register[]		Value (hex)		Remarks			
0		0A 78		Sub_funciton code			
1		00 01		axis1 (1 → AXIS_X)			
2		00 02		axis2 (2 → AXIS_Y)			
3		00 01		nAcc (velocity profile)			
4		00 00		Sp (choose the start point from 0~7 )			
5		00 00		nDir (set the direction to be CCW)			
6		00 00		MSW of Lp (length of the rectangle)			
7		4E 20		LSW of Lp (20000 = 0x4E20)			
8		00 00		MSW of Wp (width of the rectangle)			
9		27 10		LSW of Wp (10000 = 0x2710)			
10		00 00		MSW of Rp (set the corner radius value)			
11		03 E8		LSW of Rp (1000 = 0x3E8)			
12		00 00		MSW of RSV (define start velocity)			
13		03 E8		LSW of RSV (1000 = 0x3E8)			
14		00 00		MSW of RV (define velocity)			
15		27 10		LSW of RV (10000 = 0x2710)			
16		00 00		MSW of RA (define acc value)			

<b>17</b>	<b>13 88</b>	<i>LSW of RA (5000 = 1388)</i>
<b>18</b>	<b>00 00</b>	<i>MSW of RD (define dec value)</i>
<b>19</b>	<b>13 88</b>	<i>LSW of RD (5000 = 1388)</i>

## 6.4.2 2-Axis Continuous Linear Interpolation

- **\* int** ETM\_LINE\_2D\_INITIAL (**SOCKET** s, **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **DWORD** VSV, **DWORD** VV, **DWORD** VA)

### Description:

This function sets the necessary parameters for a 2-axis continuous linear interpolation using symmetric T-curve speed profile.

### Category:

MODBUS sub\_function; RTC and MP.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis1:** The first axis (axis 1). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、2、4、8).

**axis2:** The second axis (axis 2). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、2、4、8).

**VSV:** Starting speed (in PPS)

**VV:** Vector speed (in PPS)

**VA:** Vector acceleration (PPS/Sec)

### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the error.

### Sample:

```
ETM_LINE_2D_INITIAL (s, 1, AXIS_X, AXIS_Y, 300, 18000, 500000);  
//This function should be defined before the ETM_LINE_2D_CONTINUE()  
//function is used. Please refer to the example of this function.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 19	01	10	1F 40	00 09	12
Register[]		Value (hex)		Remarks			
0		0A 7C		Sub_function code			
1		00 01		axis1 (1 → AXIS_X)			
2		00 02		axis2 (2 → AXIS_Y)			
3		00 00		MSW of VSV			
4		01 2C		LSW of VSV (300 = 0x12C)			

5	00 00	<i>MSW of VV</i>
6	46 50	<i>LSW of VV (18000 = 0x4650)</i>
7	00 07	<i>MSW of VA</i>
8	A1 20	<i>LSW of VA (500000 = 0x0007A120)</i>



- **\* int** ETM\_LINE\_2D\_CONTINUE (**SOCKET** s, **BYTE** cardNo, **BYTE** nType, **long** fp1, **long** fp2)

**Description:**

This function executes a 2-axis continuous linear interpolation. **(It will not consume any system resources of WinCon or I8000).**

**Category:**

MODBUS sub\_function; RTC and MP.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**nType:** 0 → 2-axis linear continuous interpolation  
1 → end of 2-axis linear continuous interpolation

**fp1:** The assigned number of pulses for the axis 1 (in Pulses)  
(-2,000,000,000 ~ +2,000,000,000)

**fp2:** The assigned number of pulses for the axis 2 (in Pulses)  
(-2,000,000,000 ~ +2,000,000,000)

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the error.

**Sample:**

ETM\_LINE\_2D\_CONTINUE (s, 1, 0, 100, 100);  
//execute X, Y 2-axis linear continuous interpolation on module 1

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 13	01	10	1F 40	00 06	0C
Register[]		Value (hex)		Remarks			
0		0A 7E		Sub_funciton code			
1		00 00		nType			
2		00 00		MSW of fp1			
3		00 64		LSW of fp1 (100 = 0x64)			
4		00 00		MSW of fp2			
5		00 64		LSW of fp2			

### 6.4.3 3-Axis Continuous Linear Interpolation

- **\* int** ETM\_LINE\_3D\_INITIAL (**SOCKET** s, **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **BYTE** axis3, **DWORD** VSV, **DWORD** VV, **DWORD** VA)

#### Description:

This function sets the necessary parameters for a 3-axis continuous linear interpolation using symmetric T-curve speed profile.

#### Category:

MODBUS sub\_function; RTC and MP.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis1:** The first axis (axis 1). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、 2、 4、 8).

**axis2:** The second axis (axis 2). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、 2、 4、 8).

**axis3:** The third axis (axis 3). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、 2、 4、 8).

**VSV:** Starting speed (in PPS)

**VV:** Vector speed (in PPS)

**VA:** Vector acceleration (PPS/Sec)

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_LINE_3D_INITIAL (s, 1, AXIS_X, AXIS_Y, AXIS_Z, 300, 18000, 500000);  
//This function should be defined before the ETM_LINE_3D_CONTINUE()  
//function is used. Please refer to the example of this function.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 1B	01	10	1F 40	00 0A	14
Register[]		Value (hex)		Remarks			
0		0A 7F		Sub_funciton code			
1		00 01		axis1 (1 ➔ AXIS_X)			
2		00 02		axis2 (2 ➔ AXIS_Y)			

3	00 04	<i>axis2 (4 → AXIS_Z)</i>
4	00 00	<i>MSW of VSV</i>
5	01 2C	<i>LSW of VSV (300 = 0x12C)</i>
6	00 00	<i>MSW of VV</i>
7	46 50	<i>LSW of VV (18000 = 0x4650)</i>
8	00 07	<i>MSW of VA</i>
9	A1 20	<i>LSW of VA (500000 = 0x7A120)</i>

- **\* int** ETM\_LINE\_3D\_CONTINUE (**SOCKET** s, **BYTE** cardNo, **BYTE** nType, **long** fp1, **long** fp2, **long** fp3)

**Description:**

This function executes a 3-axis continuous linear interpolation. (It will not consume any system resources of WinCon or I8000).

**Category:**

MODBUS sub\_function; RTC and MP.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**nType:** 0 → 2-axis linear continuous interpolation  
1 → end of 2-axis linear continuous interpolation

**fp1:** The assigned number of pulses for the axis 1 (in Pulses)  
(-2,000,000,000 ~ +2,000,000,000)

**fp2:** The assigned number of pulses for the axis 2 (in Pulses)  
(-2,000,000,000 ~ +2,000,000,000)

**fp3:** The assigned number of pulses for the axis 3 (in Pulses)  
(-2,000,000,000 ~ +2,000,000,000)

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the error.

**Sample:**

ETM\_LINE\_3D\_CONTINUE (s, 1, 0, 100, 100, 100);

//execute X, Y, Z 3-axis linear continuous interpolation on module 1

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 17	01	10	1F 40	00 08	10
Register[]		Value (hex)		Remarks			
0		0A 81		Sub_funciton code			
1		00 00		nType			
2		00 00		MSW of fp1			
3		00 64		LSW of fp1 (100 = 0x64)			
4		00 00		MSW of fp2			
5		00 64		LSW of fp2			

<b>6</b>	<b>00 00</b>	<b><i>MSW of fp3</i></b>
<b>7</b>	<b>00 64</b>	<b><i>LSW of fp3</i></b>

## 6.4.4 Mixed Linear and Circular 2-axis motions in Continuous Interpolation

- \* **int** ETM\_MIX\_2D\_INITIAL (**SOCKET** s, **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **BYTE** nAcc, **DWORD** VSV, **DWORD** VV, **DWORD** VA)

### Description:

This function does the initial settings for mixed linear and circular 2-axis motions in continuous interpolation.

### Category:

MODBUS sub\_function; RTC and MP.

### Parameters:

s:	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
cardNo:	Module number
axis1:	The first axis (axis 1). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、2、4、8).
axis2:	The second axis (axis 2). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、2、4、8).
nAcc:	0 → constant speed (VV) 1 → symmetric T-curve Acc/Dec (VSV、VV、VA)
VSV:	Starting speed (in PPS)
VV:	Vector speed (in PPS)
VA:	Vector acceleration (PPS/Sec)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_MIX_2D_INITIAL (s, 1, 1, 2, 0, 300, 18000, 500000);  
//This function should be defined before the ETM_MIX_2D_CONTINUE()  
//function is used. Please refer to the example of this function.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 1B	01	10	1F 40	00 0A	14
Register[]		Value (hex)		Remarks			
0		0A 82		Sub_functon code			
1		00 01		axis1 (1 → AXIS_X)			
2		00 02		axis2 (2 → AXIS_Y)			

3	00 00	<i>nAcc</i>
4	00 00	<i>MSW of VSV</i>
5	01 2C	<i>LSW of VSV (300 = 0x12C)</i>
6	00 00	<i>MSW of VV</i>
7	46 50	<i>LSW of VV (18000 = 0x4650)</i>
8	00 07	<i>MSW of VA</i>
9	A1 20	<i>LSW of VA (500000 = 0x7A120)</i>

- **\* int** ETM\_MIX\_2D\_CONTINUE (**SOCKET** s, **BYTE** cardNo, **BYTE** nAcc, **BYTE** nType, **long** cp1, **long** cp2, **long** fp1, **long** fp2)

#### Description:

This function executes mixed linear and circular 2-axis motion in continuous interpolation. **(It will not consume any system resources of WinCon or I8000).**

#### Category:

MODBUS sub\_function; RTC and MP.

#### Parameters:

<b>s:</b>	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<b>cardNo:</b>	Module number
<b>nAcc:</b>	0 → continuous interpolation. 1 → it is the last command of this continuous interpolation. In Acc/Dec mode, it will perform a deceleration stop. In constant speed mode, it will directly stop rather than decelerate.
<b>nType:</b>	1 → ETM_LINE_2D( <b>BYTE</b> cardNo, <b>long</b> fp1, <b>long</b> fp2) 2 → ETM_ARC_CW( <b>BYTE</b> cardNo, <b>long</b> cp1, <b>long</b> cp2, <b>long</b> fp1, <b>long</b> fp2) 3 → ETM_ARC_CCW( <b>BYTE</b> cardNo, <b>long</b> cp1, <b>long</b> cp2, <b>long</b> fp1, <b>long</b> fp2) 4 → ETM_CIRCLE_CW( <b>BYTE</b> cardNo, <b>long</b> cp1, <b>long</b> cp2) 5 → ETM_CIRCLE_CCW( <b>BYTE</b> cardNo, <b>long</b> cp1, <b>long</b> cp2)
<b>cp1:</b>	It assigns the center point data at axis 1 (-2,000,000,000 ~ +2,000,000,000)
<b>cp2:</b>	It assigns the center point data at axis 2 (-2,000,000,000 ~ +2,000,000,000)
<b>fp1:</b>	It assigned number of pulses for axis 1 (-2,000,000,000 ~ +2,000,000,000)
<b>fp2:</b>	It assigned number of pulses for axis 2 (-2,000,000,000 ~ +2,000,000,000)

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the error.

#### Sample:

```
ETM_MIX_2D_CONTINUE (s, 1, 0, 1, 0, 0, 100, 100);
//execute X, Y, 2-axis motions in continuous interpolation on module 1
```

The MODBUS command is listed as follows:



TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 1D	01	10	1F 40	00 0B	16
Register[]		Value (hex)		Remarks			
0		0A 84		<i>Sub_funciton code</i>			
1		00 00		<i>nAcc</i>			
2		00 01		<i>nType</i>			
3		00 00		<i>MSW of cp1</i>			
4		00 00		<i>LSW of cp1 (for linear motion, set 0)</i>			
5		00 00		<i>MSW of cp2</i>			
6		00 00		<i>LSW of cp2 (for linear motion, set 0)</i>			
7		00 00		<i>MSW of fp1</i>			
8		00 64		<i>LSW of fp1 (100 = 0x64)</i>			
9		00 00		<i>MSW of fp2</i>			
10		00 64		<i>LSW of fp2 (100 = 0x64)</i>			

### 6.4.5 3-Axis Helical Motion

- **\* int** ETM\_HELIX\_3D (**SOCKET** s, **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **BYTE** axis3, **BYTE** nDir, **DWORD** VV, **long** cp1, **long** cp2, **long** cycle, **long** pitch)

#### Description:

This function performs a 3-axis helical motion. However, it is a software macro-function; therefore, it requires CPU resource to run this function. **(It will not consume any system resources of WinCon or I8000).**

#### Category:

MODBUS sub\_function; RTC and MP.

#### Parameters:

<b>s:</b>	The SOCKET handle, please refer to the <a href="#">ETM_Connect</a> function in Section <a href="#">8.2.1</a> .
<b>cardNo:</b>	Module number
<b>axis1:</b>	The first axis (axis 1). Please refer to Table 2-1. It can be either X, Y, Z, or U(1、2、4、8).
<b>axis2:</b>	The second axis (axis 2). Can be either X, Y, Z, or U axis.
<b>axis3:</b>	The third axis (axis 3). Can be either X, Y, Z, or U axis.
<b>nDir:</b>	0 → Move in a CW direction. 1 → Move in a CCW direction.
<b>VV:</b>	Vector speed (in PPS)
<b>cp1:</b>	The value of center at axis 1 (-2,000,000,000 ~ +2,000,000,000)
<b>cp2:</b>	The value of center at axis 2 (-2,000,000,000 ~ +2,000,000,000)
<b>cycle:</b>	Number of cycle (-2,000,000,000 ~ +2,000,000,000)
<b>pitch:</b>	Pitch per revolution (the advanced distance for each revolution) (-2,000,000,000 ~ +2,000,000,000)

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the error.

#### Sample:

```
ETM_HELIX_3D (s, 1, AXIS_Y, AXIS_Z, AXIS_U, 1, 50000, 0, 25000, 50, 3600);  
//the circular motion is on YZ plane, and the linear motion is along the U axis.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 25	01	10	1F 40	00 0F	1E
Register[]		Value (hex)		Remarks			
0		0A 88		Sub_funciton code			
1		00 02		axis1 (2 → AXIS_Y)			
2		00 04		axis2 (4 → AXIS_Z)			
3		00 08		axis3 (8 → AXIS_U)			
4		00 01		nDir			
5		00 00		MSW of VV			
6		C3 50		LSW of VV (50000 = 0xC350)			
7		00 00		MSW of cp1			
8		00 00		LSW of cp1 (0 = 0x0)			
9		00 00		MSW of cp2			
10		61 A8		LSW of cp2 (25000 = 0x61A8)			
11		00 00		MSW of cycle			
12		00 32		LSW of cycle (50 → 0x32)			
13		00 00		MSW of pitch			
14		0E 10		LSW of pitch (3600 → 0xE10)			

## 6.4.6 2-Axis Ration Motion

- \* **int** ETM\_RATIO\_INITIAL (**SOCKET** s, **BYTE** cardNo, **BYTE** axis1, **BYTE** axis2, **DWORD** SV, **DWORD** V, **DWORD** A, **BYTE** CMX, **BYTE** CDV)

### Description:

This function sets the Initial values for ratio motion (motion in ratio) using a symmetric T-curve speed profile.

### Category:

MODBUS sub\_function; RTC and MP.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis1:** The first axis (axis 1). Can be either X, Y, Z, or U axis (1,2,4,8). Please refer to Table 2-1 for the axis definition.

**axis2:** The second axis (axis 2). Can be either X, Y, Z, or U axis(1,2,4,8).

**SV:** Set the value for the starting speed ( in PPS)

**V:** Set the value for the vector speed (in PPS)

**A:** Set the acceleration value (in PPS/Sec)

**CMX:** Set the molecule value between the two assigned axes(1 ~+127)

**CDV:** Set the denominator value between the two assigned axes(1 ~ +127)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

ETM\_RATIO\_INITIAL (s, 1, AXIS\_U, AXIS\_X, 300, 18000, 500000, 9, 25);  
//Initial setting forETM\_RATIO\_2D(...) function.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 1D	01	10	1F 40	00 0B	16
Register[]		Value (hex)		Remarks			
0		0A 8B		Sub_funciton code			
1		00 08		axis1 (2 → AXIS_U)			
2		00 01		axis2 (4 → AXIS_X)			

3	00 00	MSW of SV
4	01 2C	LSW of SV (300 = 0x12C)
5	00 00	MSW of V
6	46 50	LSW of V (18000 = 0x4650)
7	00 07	MSW of A
8	A1 20	LSW of A (500000 = 0x7A120)
9	00 09	CMX
10	00 19	CDV (25 = 0x19)

- **\* int** ETM\_RATIO\_2D (**SOCKET** s, **BYTE** cardNo, **BYTE** nType, **long** data, **BYTE** nDir)

**Description:**

This function performs a two-axis ratio motion. **(It will not consume any system resources of WinCon or I8000).**

**Category:**

MODBUS sub\_function; RTC and MP.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**nType:** 0 → Perform the ratio motion.  
1 → Declare the end of ratio motion.

**data:** The pulse number of axis1. (-2,000,000,000 ~ +2,000,000,000)

**nDir:** Direction of the second axis.  
0: CW; 1: CCW

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

Use the ETM\_GET\_ERROR\_CODE() function to identify the error.

**Sample:**

```
ETM_RATIO_2D (s, 1, 0, 3600, 0);
//perform the ratio motion in the CW direction.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			
0		0A 8D		Sub_funciton code			
1		00 00		nType			
2		00 00		MSW of data			
3		0E 10		LSW of data (3600 = 0xE10)			
4		00 00		nDir			

## 6.4.7 Synchronous Line Scan Motion

- **int** ETM\_LINE\_SCAN (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** Type, **BYTE** outEdge, **BYTE** PulseWidth, **long** Pitch)

### Description:

Equal distance Line Scan trigger out : Max speed < 100KHz (Pulse Width 10uS). Unequal distance Line Scan trigger out : Max speed < 18KHz.

### Category:

MODBUS sub\_function; RTC.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis number: equal distance X or Y (1,2) unequal distance X(1).

**Type:** 0 = equal distance motion (please refer to C+ , LP)  
1 = unequal distance motion (please refer to C+ , LP)  
2 = equal distance motion (please refer to C+ , EP)  
3 = unequal distance motion (please refer to C+ , EP)

**outEdge:** trigger OUT active logic:  
0 = low active; 1 = high active

**PulseWidth:** trigger OUT output pulse width  
0 = 10 uSec  
1 = 20 uSec  
2 = 100 uSec  
3 = 200 uSec  
4 = 1,000 uSec  
5 = 2,000 uSec  
6 = 10,000 uSec  
7 = 20,000 uSec

**pitch:** Specify interval pulses (-2,000,000,000 ~ +2,000,000,000)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_LINE_SCAN (s, 1, AXIS_X, 0, 0, 0, -39);  
//Sets the equal distance Line Scan trigger on axis-X.  
//One trigger will be sent out every 40 (=39+1) pulses.
```

The MODBUS command is listed as follows:

TID	PID	Field	UID	FC	St_Addr.	Word	Byte
-----	-----	-------	-----	----	----------	------	------

(hex)	(hex)	Length (hex)	(hex)	(hex)	(hex)	Count (hex)	Count (hex)
00 01	00 00	00 15	01	10	1F 40	00 07	0E
Register[]		Value (hex)		Remarks			
0		0A 90		Sub_funciton code			
1		00 01		axis (1 → <i>AXIS_X</i> )			
2		00 00		Type (even distance trigger)			
3		00 00		outEdge (high level pulse)			
4		00 00		PulseWidth (0→ 10 uSec)			
5		FF FF		MSW of Pitch			
6		FF D9		LSW of Pitch (-39 = 0xFFFFFD9)			



- **int** ETM\_LINE\_SCAN\_START (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** Type, **long** Position)

**Description:**

Enable Line Scan trigger out motion.

**When performing unequal distance motion, please note (there is no restriction for equal distance motion):**

- All ISR will be stopped.
- Do not execute the following commands:

ETM\_READ\_bVAR  
ETM\_READ\_VAR  
ETM\_READ\_FLONG  
ETM\_GET\_LP  
ETM\_GET\_EP  
ETM\_GET\_CV  
ETM\_GET\_CA  
ETM\_GET\_DI  
ETM\_GET\_ERROR  
ETM\_GET\_ERROR\_CODE  
ETM\_FRNET\_IN  
ETM\_GET\_LATCH  
i8094H\_STOP\_WAIT  
ETM\_CLEAR\_STOP

**Category:**

MODBUS sub\_function; RTC.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis number: equal distance X or Y (1,2) unequal distance X(1).

**Type:** 0 = equal distance motion (please refer to LP)  
1 = unequal distance motion (please refer to LP, it require to accordingly adjust **ETM\_LINE\_SCAN\_OFFSET2**)  
2 = equal distance motion (please refer to EP)  
3 = unequal distance motion (please refer to EP, it require to accordingly adjust **ETM\_LINE\_SCAN\_OFFSET2**)

**Position:** Total pulses, movement distance  
(-2,000,000,000 ~ +2,000,000,000)

**Return:**

ICP DAS

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

ETM\_LINE\_SCAN\_START (s, 1, AXIS\_XY, 0, -4000000);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			
0		0A 91		Sub_funciton code			
1		00 03		axis (3 → AXIS_XY)			
2		00 00		Type			
3		FF C2		MSW of Position			
4		F7 00		LSW of Position (-4000000 = 0xFFC2F700)			

- **int** ETM\_LINE\_SCAN\_OFFSET2 (**SOCKET** s, **BYTE** cardNo, **DWORD** Start, **WORD** Param\_Len, **char** Offset[])

#### Description:

Set unequal distance interval offset pulses. The array, *Offset[ ]*, contains the offset for the trigger-out. (the actual trigger = equal-distance + offset). This function helps to change the contents of *Offset[ ]*. The parameter, *Start*, defines the beginning of offset-elements that will be changed.

The total length of the all unequal distance interval offsets is increased when this function is called. The maximum length of the unequal distance interval offsets is 7000.

#### Category:

MODBUS sub\_function.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**Start:** Axis number: equal distance X or Y (1,2) unequal distance X(1).  
**Param\_Len:** The number of element in *Offset[ ]*.  
**Offset[ ]:** The unequal distance interval offset.

#### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

#### Sample:

```
char Offset[10];

Offset[0] = 1;
Offset[1] = -2;
Offset[2] = 1;
Offset[3] = 2;
Offset[4] = -1;
Offset[5] = 1;
Offset[6] = 0;
Offset[7] = 1;
Offset[8] = 0;
Offset[9] = 0;
ETM_LINE_SCAN_OFFSET2 (s, 1, 0, sizeof(Offset), Offset);
```

The MODBUS command is listed as follows:

TID	PID	Field	UID	FC	St_Addr.	Word	Byte
-----	-----	-------	-----	----	----------	------	------

ICP DAS

4

ET-M8194H User Manual – API Library  
Version 1.0 2010/02/22

(hex)	(hex)	Length (hex)	(hex)	(hex)	(hex)	Count (hex)	Count (hex)
00 01	00 00	00 21	01	10	1F 40	00 0D	1A
Register[]		Value (hex)		Remarks			
0		0A 93		Sub_funciton code			
1		00 00		MSW 0F Start			
2		00 00		LSW 0F Start			
3		00 01		Offset[0]			
4		00 FE		Offset[1] (-2 = 0xFE )			
5		00 01		Offset[2]			
6		00 02		Offset[3]			
7		00 FF		Offset[4] (-1 = 0xFF )			
8		00 01		Offset[5]			
9		00 00		Offset[6]			
10		00 01		Offset[7]			
11		00 00		Offset[8]			
12		00 00		Offset[9]			

## 6.5 Other Functions

### 6.5.1 Holding theDriving Command

- **\*\* int** ETM\_DRV\_HOLD (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

#### Description:

This command is usually used when users desire to start multi-axis driving simultaneously. When this command is issued, users may write other driving commands to the control card. All the driving commands will be held after ETM\_DRV\_HOLD() is issued, and these commands will be started once the ETM\_DRV\_START() is issued. However, if in driving, this command will not cause the driving to be stopped. But the next command will be held.

#### Category:

MODBUS sub\_function; RTC, MP and ISR.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**axis:** Axis or Axes (Please refer to Table 2-1 for the axis definition.)

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_DRV_HOLD (s, 1, AXIS_XYU);  
//hold the driving command to XYU
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A B4		Sub_funciton code			
1		00 0B		axis (B → AXIS_XYU)			

## 6.5.2 Release the Holding Status and Start the Driving

- **\*\* int** ETM\_DRV\_START (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

### Description:

This command releases the holding status, and start the driving of the assigned axes immediately.

### Category:

MODBUS sub\_function; RTC, MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**axis:** Axis or Axes (Please refer to Table 2-1 for the axis definition.)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_DRV_START (s, 1, AXIS_XYU);  
//release the holding status. X,Y , and U axes will start to move simultaneously.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A B5		Sub_funciton code			
1		00 0B		axis (B → AXIS_XYU)			

### 6.5.3 Stopping the Axes

- **\* int** ETM\_STOP\_SLOWLY (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

#### Description:

This function commands to decelerate and finally stops the assigned axes slowly.

#### Category:

MODBUS sub\_function; RTC, and MP

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**axis:** Axis or Axes (Please refer to Table 2-1)

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

```
ETM_STOP_SLOWLY (s, 1, AXIS_XY);  
//decelerate and stop the X and Y axes
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A B7		Sub_funciton code			
1		00 03		axis (3 → AXIS_XY)			

- **\* int** ETM\_STOP\_SUDDENLY (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

**Description:**

This function commands to immediately stop the assigned axes.

**Category:**

MODBUS sub\_function; RTC, and MP

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**axis:** Axis or Axes (Please refer to Table 2-1)

**Return:**

0: Success; Others: Fail (Please refer to Section 8.3)

**Sample:**

ETM\_STOP\_SUDDENLY (s, 1, AXIS\_ZU);

//immediately stop the Z and U axes.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A B8		Sub_function code			
1		00 0C		axis (C → AXIS_ZU)			



- **\* int** ETM\_VSTOP\_SLOWLY (**SOCKET** s, **BYTE** cardNo)

**Description:**

This function commands to stop interpolation motion of the assigned module in a decelerating way.

**Category:**

MODBUS sub\_function; RTC, and MP

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_VSTOP_SLOWLY (s, 1);
//stop the interpolation of card 1 in a decelerating way.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A B9		<i>Sub_funciton code</i>			

- **\* int** ETM\_VSTOP\_SUDDENLY (**SOCKET** s, **BYTE** cardNo)

**Description:**

This function commands to stop interpolation motion of the assigned module immediately.

**Category:**

MODBUS sub\_function; RTC, and MP

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_VSTOP_SUDDENLY (s, 1);
// stop the interpolation of card 1 immediately.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A BA		<i>Sub_funciton code</i>			

## 6.5.4 Clear the Stop Status

- \* **int** ETM\_CLEAR\_STOP (**SOCKET** s, **BYTE** cardNo)

### Description:

After using anyone of the stop functions mentioned in section 6.5.3, please solve the malfunction, and then issue this function to clear the stop status.

### Category:

MODBUS sub\_function; RTC, and MP

### Parameters:

**s:** The SOCKET handle, please refer to the **ETM\_Connect** function in Section 8.2.1.

**cardNo:** Module number

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_CLEAR_STOP (s, 1);  
//clear the error status of card 1.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A BB		Sub_funciton code			

### 6.5.5 End of Interpolation

- **\* int** ETM\_INTP\_END (**SOCKET** s, **BYTE** cardNo, **BYTE** type)

Description:

- If the current motion status is running an interpolation motion and you would like to issue a single axis motion or change the coordinate definition, you should call this function before the new command is issued.
- You can redefine the **MAX\_V** for each axis. In this way, you do not have to execute ETM\_INTP\_END() function.

Category:

MODBUS sub\_function; RTC, and MP

Parameters:

- s:** The SOCKET handle, please refer to the **ETM\_Connect** function in Section [8.2.1](#).
- cardNo:** Module number
- type:** 0 → 2-axis interpolation  
1 → 3-axis interpolation

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

ETM\_INTP\_END (s, 1, 0); //declare the end of a 2-axis interpolation on card 1.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A BC		Sub_funciton code			
1		00 00		type (0 → 2-axis interpolation)			

---

## 7. Additional Functions Supported by i8094H

---

- **Default Settings for i-8094H when system boots:**

```
i8094H_SET_PULSE_MODE(cardNo, AXIS_XYZU, 0);  
i8094H_SET_MAX_V(cardNo, AXIS_XYZU, 2000000L);  
i8094H_SET_HLMT(cardNo, AXIS_XYZU, 0, 0);  
i8094H_LIMITSTOP_MODE(cardNo, AXIS_XYZU, 0);  
i8094H_SET_NHOME(cardNo, AXIS_XYZU, 0);  
i8094H_SET_HOME_EDGE(cardNo, AXIS_XYZU, 0);  
i8094H_CLEAR_SLMT(cardNo, AXIS_XYZU);  
i8094H_SET_ENCODER(cardNo, AXIS_XYZU, 0, 0, 0);  
i8094H_SERVO_OFF(cardNo, AXIS_XYZU);  
i8094H_SET_ALARM(cardNo, AXIS_XYZU, 0, 0);  
i8094H_SET_INPOS(cardNo, AXIS_XYZU, 0, 0);  
i8094H_SET_FILTER(cardNo, AXIS_XYZU, 0, 0);  
i8094H_VRING_DISABLE(cardNo, AXIS_XYZU);  
i8094H_AVTRI_DISABLE(cardNo, AXIS_XYZU);  
i8094H_EXD_DISABLE(cardNo, AXIS_XYZU);
```

Functions could be used in downloading parameter table (please refer to 7.1)

- **Functions could be used in downloading parameter table(IT, please refer to Section 7.1):**

```
ETM_SET_PULSE_MODE  
ETM_SET_MAX_V  
ETM_SET_HLMT  
ETM_LIMITSTOP_MODE  
ETM_SET_NHOME  
ETM_SET_HOME_EDGE  
ETM_SET_SLMT (ETM_CLEAR_SLMT)  
ETM_SET_ENCODER  
ETM_SERVO_ON (ETM_SERVO_OFF)  
ETM_SET_ALARM  
ETM_SET_INPOS  
ETM_SET_FILTER  
ETM_VRING_ENABLE (ETM_VRING_DISABLE)  
ETM_AVTRI_ENABLE (ETM_AVTRI_DISABLE)
```

<b>Notice: If the settings is loaded with ETM_LOAD_INITIAL(), please pre-set the related parameters. No default setting for ETM_LOAD_INITIAL().</b>
---

- **Set the return value of functions (please refer to Section 7.2.3) to variables:**

ETM\_GET\_LP

ETM\_GET\_EP

ETM\_GET\_DI

ETM\_GET\_ERROR

ETM\_GET\_ERROR\_CODE

ETM\_FRNET\_IN

ETM\_HOME\_WAIT

ETM\_GET\_LATCH

ETM\_GET\_DI\_ALL

*The associated function: ETM\_MP\_SET\_RVAR().*

## 7.1 Initial Parameter Table

- **int** ETM\_LOAD\_INITIAL (**SOCKET** s, **BYTE** cardNo)

Description:

Write the settings into i8094H parameter table (please refer to table 2-1a) by functions; and then load the initial setting values into i8094H by ETM\_LOAD\_INITIAL.

Category:

MODBUS sub\_function; RTC

Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

```
ETM_LOAD_INITIAL (s, 1);  
// load the initial setting values of the parameter table into i8094H
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A C8		Sub_funciton code			

## 7.2 Create Macro Program (MP)

### 7.2.1 Create Macro Program Codes

- **int** ETM\_MP\_CREATE(**SOCKET** s, **BYTE** cardNo, **BYTE** mp\_isr\_No)

Description:

Write in Macro Program codes into i8094H, and execute the program by ETM\_MP\_CALL.

1. All Macro Programs that were write inti i8094H could power outage carry-over.
2. It could not be applied to Macro Program.

Category:

MODBUS sub\_function; RTC

Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**mpNo:** Macro Program number (MP1 ~ MP157)

**isrNo:** Interrupt Service Program number (ISR1 ~ ISR20)

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

The true value of MPn = 20 + n; therefore, MP21 = 41 = 0x29.

However, the true value of ISRn = n;

```
ETM_MP_CREATE (s, 1, MP21);
```

```
//Write Macro Program into i8094H.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A C9		Sub_funciton code			
1		00 29		mpNo (MP21 = 41 = 0x29)			



- **\* int** ETM\_MP\_CLOSE (**SOCKET** s, **BYTE** cardNo)

**Description:**

Finish writing Macro Program into i8094H, this function will take effect only when working with ETM\_MP\_CREATE.

**Category:**

MODBUS sub\_function; MP

**Parameters:**

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

```
ETM_MP_CLOSE (s, 1);
// module 1 finish, write Macro Program into i8094H
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A CA		Sub_funciton code			

## 7.2.2 Execute Maro Program (MP)

- **\* int** ETM\_MP\_CALL (**SOCKET** s, **BYTE** cardNo, **BYTE** mpNo)

Description:

Execute a specific Macro Program on i8094H. **For MP, the maximum layers could be up to 7. For ISR, only one layer.**

Category:

MODBUS table, MODBUS sub\_function; RTC and MP.

Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**mpNo:** Macro Program number (MP1 ~ MP157)

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

The true value of MPn = 20 + n; therefore, MP21 = 41 = 0x29.

```
ETM_MP_CALL (s, 1, MP21);    // Execute the MP21 on i8094H
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A CB		Sub_function code			
1		00 29		mpNo (MP21 = 41 = 0x29)			

The other method to call MPn is writing the MPn to the pre-defined holding register 8.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	00 08	00 01	02
Register[]		Value (hex)		Remarks			
0		29		mpNo (MP21 = 41 = 0x29)			

--	--	--

### 7.2.3 User Defined Variables

- **\*\* int** ETM\_MP\_SET\_VAR (**SOCKET** s, **BYTE** cardNo, **long** varNo, **long** varNo1)
- **\*\* int** ETM\_MP\_SET\_VAR (**SOCKET** s, **BYTE** cardNo, **long** varNo, **long** IVar1)

#### Description:

Set variable VARn = VARn

Set variable VARn = Value of variable

#### Category:

MODBUS table, MODBUS sub\_function; MP and ISR.

#### Parameters:

- s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).
- cardNo: Module number
- varNo: Variables to be used in Macro Program: VAR0 ~ VAR511
- varNo1: Variables: VAR0 ~ VAR511
- IVar1: Value of variable (-2,000,000,000 ~ +2,000,000,000)

#### Return:

0: Success; Others: Fail (Please refer to Section 8.3)

#### Sample:

The address of VARn = 0x7FFF0000 + n; therefore, the address of VAR1 = 0x7FFF0001.

ETM\_MP\_SET\_VAR(s, 1, VAR1, 100); // VAR1 = 100

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 11	01	10	1F 40	00 05	0A
Register[]		Value (hex)		Remarks			
0		0A D2		Sub_function code			
1		7F FF		MSW address of varNo			
2		00 01		LSW address of varNo			
3		00 00		MSW of IVar1			
4		00 64		LSW of IVar1 (100 = 0x64)			

- **\*\* int** ETM\_MP\_SET\_RVAR (**SOCKET** s, **BYTE** cardNo, **long** varNo)

**Description:**

Set variable VARn=return value of the function (the value will be transferred automatically to (**long**))

**Category:**

MODBUS sub\_function; MP and ISR.

**Parameters:**

**s:** The SOCKET handle, please refer to the **ETM\_Connect** function in Section [8.2.1](#).

**cardNo:** Module number

**varNo:** Variables to be used in Macro Program: VAR0 ~ VAR511

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

The address of VARn = 0x7FFF0000 + n; therefore, the address of VAR5 = 0x7FFF0005.

ETM\_MP\_SET\_RVAR (s, 1, VAR5);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A D3		Sub_funciton code			
1		7F FF		MSW address of varNo			
2		00 05		LSW address of varNo			

- **\*\* int** ETM\_MP\_VAR\_CALCULATE (**SOCKET** s, **BYTE** cardNo, **long** varNo, **BYTE** Operator, **long** varNo1, **long** varNo2)
- **\*\* int** ETM\_MP\_VAR\_CALCULATE (**SOCKET** s, **BYTE** cardNo, **long** varNo, **BYTE** Operator, **long** IVar1, **long** varNo2)

**Description:**

Variable calculation, *varNo (+-\*/)* *varNo1 = varNo2*

Variable calculation, *varNo (+-\*/)* *value of valuable = varNo2*

**Category:**

MODBUS sub\_function; MP and ISR.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**varNo:** Variables to be used in Macro Program: VAR0 ~ VAR511

**Operator:** ' + ' addition  
' - ' subtraction  
' \* ' multiplication  
' / ' division(rounding down to the nearest integer)

**varNo1:** Instance variable for calculation: VAR0 ~ VAR511

**varNo2:** Variable for storing calculation result: VAR0 ~ VAR511

**IVar1:** Value of variable(-2,000,000,000 ~ +2,000,000,000)

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

The address of VARn = 0x7FFF0000 + n; for example, the address of VAR2 = 0x7FFF0002.

The ASCII code of operator:

Operator	ASCII (dec)	ASCII (hex)
' + '	43	2B
' - '	45	2D
' * '	42	2A
' / '	47	2F

ETM\_MP\_VAR\_CALCULATE (s, 1, VAR1, '+', VAR2, VAR3);

//VAR1 + VAR2 = VAR3

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 17	01	10	1F 40	00 08	10
Register[]		Value (hex)		Remarks			
0		0A D4		<i>Sub_funciton code</i>			
1		7F FF		<i>MSW address of varNo</i>			
2		00 01		<i>LSW address of varNo</i>			
3		00 2B		<i>Operator : ASCII code of '+'</i>			
4		7F FF		<i>MSW address of varNo1</i>			
5		00 02		<i>LSW address of varNo1</i>			
6		7F FF		<i>MSW address of varNo2</i>			
7		00 03		<i>LSW address of varNo2</i>			

## 7.2.4 Command Loop (FOR~NEXT)

- **\* int** ETM\_MP\_FOR (**SOCKET** s, **BYTE** cardNo, **long** varNo)
- **\* int** ETM\_MP\_FOR (**SOCKET** s, **BYTE** cardNo, **long** IVar)

### Description:

This function performs command loop, it could work on as many as 7 levels.

### Category:

MODBUS sub\_function; MP.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**varNo:** Variables to be used in Macro Program: VAR0 ~ VAR511

**IVar:** Value of variable(-2,000,000,000 ~ +2,000,000,000)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

The address of VARn = 0x7FFF0000 + n; for example, the address of VAR1 = 0x7FFF0001.

ETM\_MP\_FOR (s, 1, VAR1);

// enable axis X to move back and forward 1,000 Pulse, loop for 100 times.

// or input the instant operating value ETM\_MP\_FOR(1, 100)

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A D5		Sub_funciton code			
1		7F FF		MSW address of varNo			
2		00 01		LSW address of varNo			



- **\* int** ETM\_MP\_NEXT (**SOCKET** s, **BYTE** cardNo)

**Description:**

This function has to work together with ETM\_MP\_FOR.

**Category:**

MODBUS sub\_function; MP.

**Parameters:**

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**Sample:**

ETM\_MP\_NEXT (s, 1);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A D6		<i>Sub_funciton code</i>			

## 7.2.5 Condition Command (IF~ELSE)

- **\*\* int** ETM\_MP\_IF (**SOCKET** s, **BYTE** cardNo, **long** varNo, **WORD** Operator, **long** varNo1)
- **\*\* int** ETM\_MP\_IF (**SOCKET** s, **BYTE** cardNo, **long** varNo, **WORD** Operator, **long** IVar1)

### Description:

Conditions to be described under condition command: For MP, it could work on as many as 7 levels. For ISR, it could work on 1 level at most.

### Category:

MODBUS sub\_function; MP and ISR.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**varNo:** Variable to be calculated: VAR0 ~ VAR511

**Operator:** '<' is less than (ASCII code = 0x003C)  
'>' is greater than (ASCII code = 0x3D3C)  
'<=' is less than or equal to (ASCII code = 0x003E)  
'>=' is greater than or equal to (ASCII code = 0x3D3E)  
'==' is equal to (ASCII code = 0x3D3D)  
'!=' is not equal to (ASCII code = 0x3D21)  
'!' is not equal to (ASCII code = 0x0021)  
**(Operator: the single quotes(' ') represent a single or complex operators.)**

**varNo1:** Condition variable: VAR0 ~ VAR511

**IVar1:** Value of variable (-2,000,000,000 ~ +2,000,000,000)

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

The ASCII code of operator.

Operator	ASCII (dec)	ASCII (hex)
<	60	003C
<=	15676	3D3C
>	62	003E

<b>&gt;=</b>	<b>15678</b>	<b>3D3E</b>
<b>==</b>	<b>15677</b>	<b>3D3D</b>
<b>=</b>	<b>61</b>	<b>003D</b>
<b>!=</b>	<b>15649</b>	<b>3D21</b>
<b>!</b>	<b>33</b>	<b>0021</b>

ETM\_MP\_IF (s, 1, VAR1, '<=', VAR2);

//The single quotes ( ' ') is required for *Operator* parameter.

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
<b>00 01</b>	<b>00 00</b>	<b>00 13</b>	<b>01</b>	<b>10</b>	<b>1F 40</b>	<b>00 06</b>	<b>0C</b>
Register[]		Value (hex)		Remarks			
<b>0</b>		<b>0A D7</b>		<i>Sub_funciton code</i>			
<b>1</b>		<b>7F FF</b>		<i>MSW address of varNo</i>			
<b>2</b>		<b>00 01</b>		<i>LSW address of varNo</i>			
<b>3</b>		<b>3C 3D</b>		<i>Operator (0x3C3D → '&lt;=')</i>			
<b>4</b>		<b>7F FF</b>		<i>MSW address of varNo1</i>			
<b>5</b>		<b>00 02</b>		<i>LSW address of varNo1</i>			

- **\*\* int** ETM\_MP\_ELSE (**SOCKET** s, **BYTE** cardNo)

**Description:**

This function has to work together with ETM\_MP\_IF.

**Category:**

MODBUS sub\_function; MP and ISR.

**Parameters:**

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**Return:**

0: Success; Others: Fail (Please refer to Sectin 8.3)

**MODBUS 範例:**

ETM\_MP\_ELSE (s, 1);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A D8		<i>Sub_funciton code</i>			

● **\*\* int** ETM\_MP\_IF\_END (**SOCKET** s, **BYTE** cardNo)

Description:

Close ETM\_MP\_IF function.

Category:

MODBUS sub\_function; MP and ISR.

Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

cardNo: Module number

Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

Sample:

ETM\_MP\_IF\_END (s, 1);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A D9		Sub_funciton code			

## 7.2.6 TIMER

- **\* int** ETM\_MP\_TIMER (**SOCKET** s, **BYTE** cardNo, **long** varNo)
- **\* int** ETM\_MP\_TIMER (**SOCKET** s, **BYTE** cardNo, **long** ms)

### Description:

Delay the execution of the function.

### Category:

MODBUS sub\_function; MP.

### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).

**cardNo:** Module number

**varNo:** Variables to be used in Macro Program: VAR0 ~ VAR511

**ms:** Value of variable(0 ~ +2,000,000,000) ms

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_MP_TIMER (s, 1, 200);  
//delay the execution of the function for 200ms.
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0D	01	10	1F 40	00 03	06
Register[]		Value (hex)		Remarks			
0		0A DA		Sub_funciton code			
1		00 00		MSW of ms			
2		00 C8		LSW of ms (200 = 0xC8)			

### 7.2.7 Wait Motion Stop(For MP Only)

- \* **int** ETM\_MP\_STOP\_WAIT (**SOCKET** s, **BYTE** cardNo, **BYTE** axis)

#### Description:

The MP function could be executed only after the axes completely stop.

#### Category:

MODBUS sub\_function; MP.

#### Parameters:

**s:** The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
**cardNo:** Module number  
**axis:** Axis or axes (Please refer to Table 2-1)

#### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

#### Sample:

ETM\_MP\_STOP\_WAIT (s, 1, AXIS\_Y);

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 0B	01	10	1F 40	00 02	04
Register[]		Value (hex)		Remarks			
0		0A DB		Sub_funciton code			
1		00 02		axis (2 → AXIS_Y)			

## 7.2.8 User-defined RINT

- **\*\* int** ETM\_MP\_SET\_RINT (**SOCKET** s, **BYTE** cardNo)

### Description:

This function configures user-defined RINT settings. When this function is executed in MP, the module will send out **0x04** interrupt to the controller. This function should be activated with the ETM\_ENABLE\_RINT( ).

### Category:

MODBUS sub\_function; MP ans ISR.

### Parameters:

s: The SOCKET handle, please refer to the [ETM\\_Connect](#) function in Section [8.2.1](#).  
cardNo: Module number

### Return:

0: Success; Others: Fail (Please refer to Sectin 8.3)

### Sample:

```
ETM_MP_SET_RINT (s, 1);
```

The MODBUS command is listed as follows:

TID (hex)	PID (hex)	Field Length (hex)	UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
00 01	00 00	00 09	01	10	1F 40	00 01	02
Register[]		Value (hex)		Remarks			
0		0A DC		Sub_funciton code			



---

## 8. ET-M8194H API

---

### 8.1 API Function Types:

The ET-M8194H API Library Functions are divided into 9 types. These functions help user to edit the Macro Codes on i-8094H, access the I-8094H via MODBUS/TCP. With these functions, multiple ET-M8194H can be controlled by single Host. The types of these functions are listed as follows:

- (1) Communication Functions
- (2) MODBUS Function Code 16
- (3) MODBUS Function Code 04
- (4) MODBUS Function Code 03
- (5) MODBUS Function Code 01
- (6) MODBUS Function Code 02
- (7) MODBUS Function Code 05
- (8) MODBUS Function Code 15
- (9) Other Functions

## 8.2 API Function Descriptions:

This section gives the descriptions of the ET-M8194H API.

### 8.2.1 Communication Functions:

The functions help to establish the Ethernet connection between the PC and ET-M8194H. With these functions, the MODBUS command can be sent out manually. The MODBUS command on this connection can be monitored, as well.

#### 1. **SOCKET ETM\_Connect** (char\* ip)

Func.: Establish the Ethernet connection between PC and ET-M8194H. [The connection is required before sending any MODBUS command out.](#)  
The ET-M8194H allows 29 Clients to connect to simultaneously. [連](#)The returned SOCKET stands for the connection to every ET-M8194H, and this handle is needed to control the ET-M8194H.

Parm.: ip: The IP address of specific ET-M8194H

Return: >0:Success; -1:Fail (Timeout = 5 second)

Sample: [VC++]

```
#include <winsock2.h>
#include "ET_M8194H_API.h"
SOCKET Skt[10];
BYTE Rtn;
Skt[1] = ETM_Connect("192.168.0.1");
//The connection to 1st ET-M8194H
Skt[2] = ETM_Connect("192.168.0.2");
//The connection to 2nd ET-M8194H
Rtn = ETM_RESET_CARD(Skt[1], 1); //Reset the 1st ET-M8194H
Rtn = ETM_RESET_CARD(Skt[2], 1); // Reset the 2nd ET-M8194H
```

[VB] – The Module, ET\_M8194H\_Lib.bas, must be added into VB Project.

```
Dim Skt[10] As Long
Skt[1] = ETM_Connect("192.168.0.1")
//The connection to 1st ET-M8194H
Skt[2] = ETM_Connect("192.168.0.2")
//The connection to 2nd ET-M8194H
Call ETM_RESET_CARD(Skt[1], 1) //Reset the 1st ET-M8194H
Call ETM_RESET_CARD(Skt[2], 1) // Reset the 2nd ET-M8194H
```

2. **int ETM\_DisConnect (SOCKET s)**

Func.: Disconnect the established Ethernet connection.

Param.: s:The SOCKET handle to the target ET-M8194H.

Return: 0: Success ; -1:Fail

3. **int ETM\_Connection\_State (SOCKET s)**

Func.: Shows the status of Ethernet connection.

Param.: s:The SOCKET handle to the target ET-M8194H.

Return: 1:Connected; 0:Disconnected

4. **int ETM\_Set\_DebugMode (SOCKET s, BYTE DebugMode)**

Func.: This function enables/disable to display the MODBUS command. This feature helps user to debug the functions based on MODBUS Function Code 03, 04 and 16.

Param.: s:The SOCKET handle to the target ET-M8194H.

DebugMode : 1: Enable Debug Message, 0: Disable Debug Message

Return: 0: Success ; -1:Fail

5. **int ETM\_MODBUS\_CMD (SOCKET s, char\* Request, int Req\_Len, char\* Response, int\* Res\_Len)**

Func.: Send out the MODBUS command without related message.

Param.: s:The SOCKET handle to the target ET-M8194H.

Request: The sent-out MODBUS command

Req\_Len: The length of sent-out MODBUS command

Response: The MODBUS response from ET-M8194H

Res\_Len: The length of MODBUS response from ET-M8194H

Return: 0: Success; Others:Fail ( Timeout = 5 second )

( Please refer to Section 8.3 )

6. **int ETM\_MODBUS\_CMD\_DEBUG (SOCKET s, char\* Request, int Req\_Len, char\* Response, int\* Res\_Len) :**

Func.: Send out the MODBUS command with the related message.

Param.: s:The SOCKET handle to the target ET-M8194H.

Request : The sent-out MODBUS command

Req\_Len: The length of sent-out MODBUS command

Response: The MODBUS response from ET-M8194H

Res\_Len: The length of MODBUS response from ET-M8194H

Return: 0: Success; Others:Fail ( Timeout = 5 second )

( Please refer to Section 8.3 )

## 8.2.2 MODBUS Function Code 16

These functions implement the features in [Appendix C](#) ( [Sub Function Code mapping table](#) ). These functions help user to configure i-8094H with their programs. The functions based on FC16 are listed as follows :  
(for detailed, please refer to Chapter 2 ~ Chapter 7)

### [1]. Basic Setting Function :

```
int ETM_RESET_CARD (SOCKET s, BYTE cardNo);
int ETM_CLEAR_CARD_BUFFER (SOCKET s, BYTE cardNo);
int ETM_SET_PULSE_MODE (SOCKET s, BYTE cardNo, BYTE axis,
                        BYTE nMode);
int ETM_SET_MAX_V (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);
int ETM_SET_HLMT (SOCKET s, BYTE cardNo, BYTE axis, BYTE nFLEdge,
                 BYTE nRLEdge);
int ETM_LIMITSTOP_MODE (SOCKET s, BYTE cardNo, BYTE axis,
                       BYTE nMode);
int ETM_SET_NHOME (SOCKET s, BYTE cardNo, BYTE axis,
                  BYTE nNHedge);
int ETM_SET_HOME_EDGE (SOCKET s, BYTE cardNo, BYTE axis,
                      BYTE nHedge);
int ETM_SET_SLMT (SOCKET s, BYTE cardNo, BYTE axis, long dwFL,
                 long dwRL, BYTE nType);
int ETM_CLEAR_SLMT (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_SET_ENCODER (SOCKET s, BYTE cardNo, BYTE axis,
                    BYTE nMode, BYTE nDivision, BYTE nZEdge);
int ETM_SERVO_ON (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_SERVO_OFF (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_SET_ALARM (SOCKET s, BYTE cardNo, BYTE axis, BYTE nMode,
                  BYTE nAEdge);
int ETM_SET_INPOS (SOCKET s, BYTE cardNo, BYTE axis, BYTE nMode,
                  BYTE nIEdge);
int ETM_SET_FILTER (SOCKET s, BYTE cardNo, BYTE axis, BYTE FEn,
                   BYTE FLn);
int ETM_VRING_ENABLE (SOCKET s, BYTE cardNo, BYTE axis,
                     DWORD nVRing);
int ETM_VRING_DISABLE (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_AVTRI_ENABLE (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_AVTRI_DISABLE (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_EXD_MP (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);
int ETM_EXD_FP (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);
int ETM_EXD_CP (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);
int ETM_EXD_DISABLE (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_WRITE_bVAR (SOCKET s, BYTE cardNo, BYTE bvarNo, BYTE bVar);
int ETM_WRITE_VAR (SOCKET s, BYTE cardNo, long varNo, long lVar);
int ETM_WRITE_MD (SOCKET s, BYTE cardNo, long mdNo, long ldata,
                 float fdata);
int ETM_Lock_IP (SOCKET s, char* IPStart, char* IPEnd); (Please refer to
Appendix F)
int ETM_UnLock_IP (SOCKET s);
```

```
int ETM_SetNET (SOCKET s, char* ServerIP, char* ServerMask,  
                char* ServerGateway);
```

**[2]. Status Function :**

```
int ETM_SET_LP (SOCKET s, BYTE cardNo, BYTE axis, long wdata);  
int ETM_GET_LP (SOCKET s, BYTE cardNo, BYTE axis);  
int ETM_SET_EP (SOCKET s, BYTE cardNo, BYTE axis, long wdata);  
int ETM_GET_EP (SOCKET s, BYTE cardNo, BYTE axis);  
int ETM_GET_DI (SOCKET s, BYTE cardNo, BYTE axis, BYTE nType);  
int ETM_GET_DI_ALL (SOCKET s, BYTE cardNo, BYTE axis);  
int ETM_GET_ERROR (SOCKET s, BYTE cardNo);  
int ETM_GET_ERROR_CODE (SOCKET s, BYTE cardNo, BYTE axis);
```

**[3]. FRNET DIO Function :**

```
int ETM_FRNET_IN (SOCKET s, BYTE cardNo, BYTE wGroup);  
int ETM_FRNET_OUT (SOCKET s, BYTE cardNo, BYTE wGroup,  
                  DWORD data);
```

**[4]. Auto Homing Function :**

```
int ETM_SET_HV (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_HOME_LIMIT (SOCKET s, BYTE cardNo, BYTE axis, BYTE nType);  
int ETM_SET_HOME_MODE (SOCKET s, BYTE cardNo, BYTE axis,  
                       BYTE nStep1, BYTE nStep2, BYTE nStep3,  
                       BYTE nStep4, DWORD data);  
int ETM_HOME_START (SOCKET s, BYTE cardNo, BYTE axis);
```

**[5]. Single Axis Motion Function :**

```
int ETM_NORMAL_SPEED (SOCKET s, BYTE cardNo,  
                      BYTE axis, BYTE nMode);  
int ETM_SET_SV (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_SET_V (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_SET_A (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_SET_D (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_SET_K (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_SET_L (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_SET_AO (SOCKET s, BYTE cardNo, BYTE axis, long data);  
int ETM_FIXED_MOVE (SOCKET s, BYTE cardNo, BYTE axis, long data);  
int ETM_SET_PULSE (SOCKET s, BYTE cardNo, BYTE axis, DWORD data);  
int ETM_CONTINUE_MOVE (SOCKET s, BYTE cardNo, BYTE axis, long data);
```

**[6]. Interpolation Function :**

```
int ETM_AXIS_ASSIGN (SOCKET s, BYTE cardNo, BYTE axis1,  
                    BYTE axis2, BYTE axis3);  
int ETM_VECTOR_SPEED (SOCKET s, BYTE cardNo, BYTE nMode);  
int ETM_SET_VSV (SOCKET s, BYTE cardNo, DWORD data);  
int ETM_SET_VV (SOCKET s, BYTE cardNo, DWORD data);  
int ETM_SET_VA (SOCKET s, BYTE cardNo, DWORD data);  
int ETM_SET_VD (SOCKET s, BYTE cardNo, DWORD data);  
int ETM_SET_VK (SOCKET s, BYTE cardNo, DWORD data);  
int ETM_SET_VL (SOCKET s, BYTE cardNo, DWORD data);  
int ETM_SET_VAO (SOCKET s, BYTE cardNo, long data);
```

```

int ETM_LINE_2D (SOCKET s, BYTE cardNo, long fp1, long fp2);
int ETM_LINE_3D (SOCKET s, BYTE cardNo, long fp1, long fp2, long fp3);
int ETM_ARC_CW (SOCKET s, BYTE cardNo, long cp1, long cp2,
               long fp1, long fp2);
int ETM_ARC_CCW (SOCKET s, BYTE cardNo, long cp1, long cp2,
               long fp1, long fp2);
int ETM_CIRCLE_CW (SOCKET s, BYTE cardNo, long cp1, long cp2);
int ETM_CIRCLE_CCW (SOCKET s, BYTE cardNo, long cp1, long cp2);

```

#### [7]. Synchronous Action Function :

```

int ETM_SYNC_ACTION (SOCKET s, BYTE cardNo, BYTE axis1, BYTE axis2,
                    DWORD nSYNC, BYTE nDRV, BYTE nLATCH,
                    BYTE nPRESET, BYTE nOUT, BYTE nINT,
                    BYTE isrNoX, BYTE isrNoY, BYTE isrNoZ,
                    BYTE isrNoU);
int ETM_SET_COMPARE (SOCKET s, BYTE cardNo, BYTE axis,
                    BYTE nSELECT, BYTE nTYPE, long data);
int ETM_GET_LATCH (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_SET_PRESET (SOCKET s, BYTE cardNo, BYTE axis, long data);
int ETM_SET_OUT (SOCKET s, BYTE cardNo, BYTE axis, BYTE outEdge,
                BYTE PulseWidth);

```

#### [8]. Interrupt Control Function :

```

int ETM_ENABLE_INT (SOCKET s, BYTE cardNo);
int ETM_DISABLE_INT (SOCKET s, BYTE cardNo);
int ETM_INTFACTOR_ENABLE (SOCKET s, BYTE cardNo, BYTE axis,
                        BYTE nINT, BYTE isrNo);
int ETM_INTFACTOR_DISABLE (SOCKET s, BYTE cardNo, BYTE axis,
                        BYTE nINT);
int ETM_ENABLE_RINT (SOCKET s, BYTE cardNo);
int ETM_DISABLE_RINT (SOCKET s, BYTE cardNo);

```

#### [9]. Continuous Interpolation Function :

```

int ETM_RECTANGLE (SOCKET s, BYTE cardNo, BYTE axis1, BYTE axis2,
                  BYTE nAcc, BYTE Sp, BYTE nDir, long Lp, long Wp,
                  long Rp, DWORD RSV, DWORD RV, DWORD RA,
                  DWORD RD);
int ETM_LINE_2D_INITIAL (SOCKET s, BYTE cardNo, BYTE axis1,
                       BYTE axis2, DWORD VSV, DWORD VV, DWORD VA);
int ETM_LINE_2D_CONTINUE (SOCKET s, BYTE cardNo, BYTE nType,
                        long fp1, long fp2);
int ETM_LINE_3D_INITIAL (BYTE cardNo, BYTE axis1, BYTE axis2,
                       BYTE axis3, DWORD VSV, DWORD VV, DWORD VA);
int ETM_LINE_3D_CONTINUE (SOCKET s, BYTE cardNo, BYTE nType,
                        long fp1, long fp2, long fp3);
int ETM_MIX_2D_INITIAL (SOCKET s, BYTE cardNo, BYTE axis1, BYTE axis2,
                      BYTE nAcc, DWORD VSV, DWORD VV, DWORD VA);
int ETM_MIX_2D_CONTINUE (SOCKET s, BYTE cardNo, BYTE nAcc,
                       BYTE nType, long cp1, long cp2, long fp1, long fp2);
int ETM_CONTINUE_INTP (SOCKET s, BYTE cardNo, BYTE axis1, BYTE axis2,
                     BYTE axis3, BYTE nAcc, DWORD VSV,

```

```

        DWORD VV, DWORD VA, DWORD VD);
int ETM_HELIX_3D (SOCKET s, BYTE cardNo, BYTE axis1, BYTE axis2,
        BYTE axis3, BYTE nDir, DWORD VV, long cp1,
        long cp2, long cycle, long pitch);
int ETM_RATIO_INITIAL (SOCKET s, BYTE cardNo, BYTE axis1, BYTE axis2,
        DWORD SV, DWORD V, DWORD A, BYTE CMX,
        BYTE CDV);
int ETM_RATIO_2D (SOCKET s, BYTE cardNo, BYTE nType, long data,
        BYTE nDir);
int ETM_LINE_SCAN (SOCKET s, BYTE cardNo, BYTE axis, BYTE Type,
        BYTE outEdge, BYTE PulseWidth, long Pitch);
int ETM_LINE_SCAN_START (SOCKET s, BYTE cardNo, BYTE axis,
        BYTE Type, long Position);
int ETM_LINE_SCAN_OFFSET2 (SOCKET s, BYTE cardNo, DWORD Start,
        WORD Param_Len, char Offset[]);

```

#### [10]. Other Function :

```

int ETM_DRV_HOLD (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_DRV_START (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_STOP_SLOWLY (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_STOP_SUDDENLY (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_VSTOP_SLOWLY (SOCKET s, BYTE cardNo);
int ETM_VSTOP_SUDDENLY (SOCKET s, BYTE cardNo);
int ETM_CLEAR_STOP (SOCKET s, BYTE cardNo);
int ETM_INTP_END (SOCKET s, BYTE cardNo, BYTE type);

```

#### [11]. Macro Program Function :

```

int ETM_LOAD_INITIAL (SOCKET s, BYTE cardNo);
int ETM_MP_CREATE (SOCKET s, BYTE cardNo, BYTE mp_isr_No);
int ETM_MP_CLOSE (SOCKET s, BYTE cardNo);
int ETM_MP_CALL (SOCKET s, BYTE cardNo, BYTE mpNo);
int ETM_MP_SET_VAR (SOCKET s, BYTE cardNo, long varNo, long varNo1);
int ETM_MP_SET_RVAR (SOCKET s, BYTE cardNo, long varNo);
int ETM_MP_VAR_CALCULATE (SOCKET s, BYTE cardNo, long varNo,
        BYTE Operator, long varNo1, long varNo2);
int ETM_MP_FOR (SOCKET s, BYTE cardNo, long varNo);
int ETM_MP_NEXT (SOCKET s, BYTE cardNo);
int ETM_MP_IF (SOCKET s, BYTE cardNo, long varNo, WORD Operator,
        long varNo1);
int ETM_MP_ELSE (SOCKET s, BYTE cardNo);
int ETM_MP_IF_END (SOCKET s, BYTE cardNo);
int ETM_MP_TIMER (SOCKET s, BYTE cardNo, long varNo);
int ETM_MP_STOP_WAIT (SOCKET s, BYTE cardNo, BYTE axis);
int ETM_MP_SET_RINT (SOCKET s, BYTE cardNo);

```

return value of FC16: 0:Success, Others: Fail (Please refer to Section 8.3 )



### 8.2.3 MODBUS Functoin Code 04

These functions implement the features in [Appendix A](#) ( [Sinput Registers](#) ). These functions help user to configure i-8094H with their programs. The functions based on FC04 are listed as follows :  
(for detailed, please refer to [Appendix A](#))

1. **int ETM04\_GET\_DI\_ALL** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **WORD\*** DIValue)

DIValue: The all DI status of specific axis (DI\_0 ~ DI\_9).

2. **int ETM04\_GET\_RINT\_STATE\_ALL** (**SOCKET** s, **BYTE** cardNo, **BYTE\*** RINTState)

RINTState: The all RINT status(Bit0 ~ Bit7).

3. **int ETM04\_GET\_ERROR\_STATE** (**SOCKET** s, **BYTE** cardNo, **BYTE\*** ErrState)

ErrState: The error state of specific module.  
(1: error occurred, 0:no error)

4. **int ETM04\_GET\_FREE\_BUFFER** (**SOCKET** s, **BYTE** cardNo, **BYTE\*** FreeBufNum)

FreeBufNum: The available block numbers of Buffer, the maximum block number is 30, and always keeps the maximum block normally.

5. **int ETM04\_GET\_FRNET\_GPDI** (**SOCKET** s, **BYTE** cardNo, **BYTE** wSA, **WORD\*** GPDIValue)

GPDIValue: The FRnet DI input of specific DI group(8 ~ 15).

6. **int ETM04\_GET\_ERROR\_CODE**(**SOCKET** s, **BYTE** cardNo, **BYTE** axis , **WORD\*** ErrCode)

ErrCode: The panic error of specific axis.

7. **int ETM04\_GET\_DI** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE** DIIn, **BYTE\*** DIState)

DIState: The single DI input of specific axis(one of DI\_0 ~ DI\_9).

8. **int ETM04\_GET\_LINE\_SCAN\_DONE** (**SOCKET** s, **BYTE** cardNo, **BYTE\*** LScanState)

LScanState: The state of Line\_Scan\_Completed ( Bit0 of RINT State ),  
1: Completed, 0: Not completed.

9. **int ETM04\_GET\_MP\_DONE** (**SOCKET** s, **BYTE** cardNo, **BYTE\*** MPState)

MPState: The state of MP\_Completed ( Bit1 of RINT State ),  
1: Completed, 0: Not completed.

10. **int ETM04\_GET\_USER\_RINT** (**SOCKET** s, **BYTE** cardNo, **BYTE\*** URINTState)

URINTState: The state of User-Defined\_RINT( Bit2 of RINT State ),  
1: Completed, 0:Not Completed.

11. **int ETM04\_GET\_MODULE\_ID** (**SOCKET** s, **BYTE** cardNo, **WORD\*** ModID)

ModID: Module ID ( [i8094H](#): 0x44 ; [i8094A](#): 0x55 ).

12. **int ETM04\_GET\_FIRMWARE\_VERSION** (**SOCKET** s, **WORD\*** FWVer)

FWVer: The Firmware version of ET-M8194H (for instance, 0x0100 is Version 1.00).



13. **int ETM04\_GET\_STOP\_STATUS** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **BYTE\*** StopState)

StopState: The motion state of specific axis (1:Stopped, 0: Running)/

14. **int ETM04\_GET\_LATCH** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long\*** LatchValue)

LatchValue: The latched data of specific axis.

15. **int ETM04\_GET\_TCN** (**SOCKET** s, **BYTE\*** TConnNum)

TConnNum: The number of MODBUS/TCP connections.

return value of FC04: 0:Success, Others: Fail (Please refer to Section 8.3 )

## 8.2.4 MODBUS Functoin Code 03

These functions implement the features in [Appendix B](#) ( [Sinput Registers](#) ). These functions help user to configure i-8094H with their programs. The functions based on FC03 are listed as follows :  
(for detailed, please refer to [Appendix B](#))

1. **int ETM03\_GET\_FRNET\_GPDO** (**SOCKET** s, **BYTE** cardNo, **BYTE** wSA, **WORD\*** GPDOValue)

GPDOValue: The FRnet DO output of specific DO group(0 ~ 7).

2. **int ETM03\_GET\_LP** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis, **long\*** LPValue)

LPValue: The LP value of specific axis.

3. **int ETM03\_GET\_EP** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis , **long\*** EPValue)

EPValue: The EP value of specific axis.

4. **int ETM03\_GET\_CV** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis , **long\*** CVValue)

CVValue: The CV value of specific axis.

5. **int ETM03\_GET\_CA** (**SOCKET** s, **BYTE** cardNo, **BYTE** axis , **long\*** CAValue)

CAValue: The CA value of specific axis.

6. **int ETM03\_READ\_bVAR** (**SOCKET** s, **BYTE** cardNo, **BYTE** bvarNo, **BYTE\*** bVARValue)

bVARValue: The bVAR( bVAR0 ~ bVAR127) value (in byte).

7. **int ETM03\_READ\_VAR** (**SOCKET** s, **BYTE** cardNo, **long** varNo, **long\*** VARValue)

VARValue: The VAR( VAR0 ~ VAR511) value (in long)

8. **int ETM03\_READ\_MD** (**SOCKET** s, **BYTE** cardNo, **long** mdNo, **long\*** ldata, **float\*** fdata)

mdNo : The mdNo to be read back (MD0 ~ MD2047).

ldata : The DWORD value of specific mdNo.

fdata : The FLOAT value of specific mdNo.

9. **int ETM03\_Read\_LockIP** (**SOCKET** s, **char\*** IPStart, **char\*** IPEnd)

IP Param.: The ragne of Lock\_IP. The IPStart is the beginning of the allowed IP address; and the IPEnd is the final allowed IP address. If these two parameters are both zero, no IP restriction is set.

10. **int ETM03\_GetNET** (**SOCKET** s, **char\*** ServerIP, **char\*** ServerMask, **char\*** ServerGateway)

Server Param.: The IP, Mask and Gateway settings of ET-M8194H.

return value of FC03: 0:Success, Others: Fail (Please refer to Section 8.3 )

## 8.2.5 MODBUS Function Code 01

The function implements the MODBUS FC01 features, and helps user to read the multiple FRnet DO on i-8094H. The functions based on FC01 are listed as follows :

(for detailed, please refer to [Appendix D](#))

1. **int ETM01\_READ\_FRNET\_MultiDO** (**SOCKET** s, **BYTE** cardNo, **BYTE** St\_Addr, **BYTE** Bit\_Len, **char\*** Read\_Data, **int\*** Read\_Len)

Param.: St\_Addr : The starting address of needed DO outputs(0 ~ 127)

Bit\_Len : The length of needed DO outputs(in bit ).

Read\_Data : The read data.(Low Byte first, and then High Byte)

Read\_Len : The length of read data(in byte).

return value of FC01: 0:Success, Others: Fail (Please refer to Section 8.3 )

## 8.2.6 MODBUS Function Code 02

These functions implement the MODBUS FC02 features, and help user to read the single/multiple FRnet DI on i-8094H. The functions based on FC02 are listed as follows :

(for detailed, please refer to [Appendix D](#))

1. **int ETM02\_READ\_FRNET\_MultiDI** (**SOCKET** s, **BYTE** cardNo, **BYTE** St\_Addr, **BYTE** Bit\_Len, **char\*** Read\_Data, **int\*** Read\_Len)

Param.: St\_Addr : The starting address of needed DI inputs(0 ~ 127)

Bit\_Len : The length of needed DI inputs(in bit ).

Read\_Data : The read data.(Low Byte first, and then High Byte)

Read\_Len : The length of read data(in byte).

2. **int ETM02\_READ\_FRNET\_SingleDI** (**SOCKET** s, **BYTE** cardNo, **BYTE** St\_Addr, **BYTE\*** Bit\_Val)

Param.: St\_Addr : The address of needed DI input (0 ~ 127 ).

Bit\_Val : The read Data\_Bit ( 1:ON, 0:OFF).

return value of FC02: 0:Success, Others: Fail (Please refer to Section 8.3 )

### 8.2.7 MODBUS Functoin Code 05

These function implements the MODBUS FC05 features, and help user to write the single FRnet DO on i-8094H. The functions based on FC05 are listed as follows :

(for detailed, please refer to [Appendix D](#))

1. **int ETM05\_WRITE\_FRNET\_SingleDO** (**SOCKET** s, **BYTE** cardNo,  
**BYTE** St\_Addr, **BYTE** Bit\_Val)

Param.: St\_Addr : The addrees to be written the Data\_Bit( 0 ~ 127 ).

Bit\_Val : The value to be written ( 1:ON, 0:OFF )

return value of FC05: 0:Success, Others: Fail (Please refer to Section 8.3 )

### 8.2.8 MODBUS Functoin Code 15

These function implements the MODBUS FC15 features, and help user to write the multiple FRnet DO on i-8094H. The functions based on FC15 are listed as follows :

(for detailed, please refer to [Appendix D](#))

1. **int ETM15\_WRITE\_FRNET\_MultiDO** (**SOCKET** s, **BYTE** cardNo, **BYTE** St\_Addr, **BYTE** Bit\_Len, **char\*** Write\_Data)

Param.: St\_Addr : The starting address of DO outputs.

Bit\_Len : The length of DO outputs( in bit ).

Write\_Data : The array that contains each DO output data.  
(in Byte, Low-Byte first and then High-Byte)

return value of FC15: 0:Success, Others: Fail (Please refer to Section 8.3 )

## 8.2.9 Other Functions

These helpful functions to simplify the programming.

### 1. **BYTE ETM\_DoEvents** (void)

Param.: None

Func.: Release the CPU in loop-interactions.

Sample:

```
ETM_FIXED_MOVE(CardNo, AXIS_X, -20000);  
while (ETM04_GET_STOP_STATUS(CardNo, axis, &StopState) == 0){  
    ETM_DoEvents();  
}
```

Return: 0:Success, 1:Fail

### 2. **BYTE ETM\_Strtok** (char\* DecStr, char\* DivChar, BYTE\* DecData)

Param.: DecStr : The string to be processed( for instance, "192.168.0.16" ).

DivChar : The specific token( for instance, "." ).

DecData : The byte array for processed string.

( for instance, DecData[0] = 192, DecData[1] = 168, DecData[2] = 0 and DecData[3] = 16 )

Func.: Help to separate the string with the specific token. The processed number in 0~255.

Return: 0:Success, 1:Fail

### 3. **WORD ETM\_GET\_DLL\_VERSION** (void)

Return: The version of ET-M8194H API Library

(for instance, the returned value, 0x0100, means the version 1.00).

**The following functions protect the writing commands (FC05, 15, 16)**

### 4. **BYTE ETM\_Set\_PWD** (char\* Oldpwd, char\* Newpwd)

Param.: Oldpwd : The active password.

Newpwd : The new password.

Func.: Set the password for writing-protection.

Return: 0:Success, 1:Fail

### 5. **BYTE ETM\_Reset\_PWD** (char\* pwd)

Param.: pwd : The access password.

Func.: Disable the writing protection, and the password will be reset.

The ET-M8194H will accept all write commands.

Return: 0:Success, 1:Fail

### 6. **BYTE ETM\_LogIn** (char\* pwd)

Param.: pwd : The login password.

Func.: When the writing-protection is enabled, the ETM\_LogIn is required for FC05, 15 and 16 writing commands.

Return: 0:Success, 1:Fail

### 7. **BYTE ETM\_LogOut** (void)

Param.: None.

Func.: Logout.

**Return: 0:Success, 1:Fail**

**8. BYTE ETM\_Access\_State (void)**

**Param.: None.**

**Func.: Check the privilege of writing.**

**Return: 0:No ; 1:Yes**

## 8.3 ErrorCode

- (1) **SUCCESSFUL\_FUNCTION\_CALL** **0x00**  
=> Successful communication via MODBUS.
- (2) **ILLEGAL\_FUNCTION\_CODE** **0x01**  
=> Error in the Function\_Code of MODBUS.
- (3) **ILLEGAL\_DATA\_ADDRESS** **0x02**  
=> Invalid starting-address of MODBUS command.  
for instance, the two Registers are to be accessed ( LP\_X, LP\_Y ... ), the address of the Low\_Word should be the correct one.
- (4) **ILLEGAL\_DATA\_VALUE** **0x03**  
=> Invalid data of MODBUS command, such as the number of parameters, the target-axis, the addresses of MD, VAR bVAR.
- (5) **SLAVE\_DEVICE\_FAILURE** **0x04**  
=> Indicates that no i-809H is resident on ET-M8194H.
- (6) **SLAVE\_DEVICE\_BUSY** **0x06**  
=> Indicates the DPRAM-Buffer on i-8094H is occupied; no more command can be forwarded through DPRAM-Buffer.
- (7) **WRONG\_CARD\_NO** **0x0B**  
=> Invalid UID ( the CardNo parameter ).
- (8) **SOCKET\_ERR** **-1**  
=> Error in communication, please check the Ethernet connection.
- (9) **WRITE\_DENY** **-2**  
=> Indicates the writing command is denied after enabling the writing-protection feature.



---

## Appendix

---

### A. Input Registers

Please use FC=4 (i.e., 0x10) to read the related states (from the base-address 0)

Variable Name	Address	Type	Comment
DI_ALL_X	0	R	The all DI status of AXIS-X, please refer to the definition of <b>ETM_GET_DI_ALL()</b> . If the individual DI is needed, please read X_DI_0 ~ X_DI_9.
DI_ALL_Y	1	R	The all DI status of AXIS-Y, please refer to the definition of <b>ETM_GET_DI_ALL()</b> . If the individual DI is needed, please read Y_DI_0 ~ Y_DI_9.
DI_ALL_Z	2	R	The all DI status of AXIS-Z, please refer to the definition of <b>ETM_GET_DI_ALL()</b> . If the individual DI is needed, please read Z_DI_0 ~ Z_DI_9.
DI_ALL_U	3	R	The all DI status of AXIS-Z, please refer to the definition of <b>ETM_GET_DI_ALL()</b> . If the individual DI is needed, please read U_DI_0 ~ U_DI_9.
RINT_STATE_ALL	4	R	Read the RINT state. For detailed, please refer to the descriptions of <b>ETM_RINT_WAIT()</b> and <b>ETM_RINT_ENABLE()</b> .
ERROR_STATE	5	R	Check the the Error status (1: error occurred; 0: no error) If error occurs, please call <b>ETM_ERROR_CODE()</b> for detailed error message.
FREE_BUFFER_SIZE	6	R	Get the available block-numbers in Buffer. The maximum block is 30.

Variable Name	Address	Type	Comment
.....Undefined	7	--	0
FRnet_DI_8G	8	R	The value of FRnet DI group 8 (16-bit)
FRnet_DI_9G	9	R	The value of FRnet DI group 9 (16-bit)
FRnet_DI_10G	10	R	The value of FRnet DI group 10 (16-bit)
FRnet_DI_11G	11	R	The value of FRnet DI group 11 (16-bit)
FRnet_DI_12G	12	R	The value of FRnet DI group 12 (16-bit)
FRnet_DI_13G	13	R	The value of FRnet DI group 13 (16-bit)
FRnet_DI_14G	14	R	The value of FRnet DI group 14 (16-bit)
FRnet_DI_15G	15	R	The value of FRnet DI group 15 (16-bit)
ERROR_CODE_X	16	R	The error occurred on AXIS-X. Please refer to ETM_GET_ERROR_CODE() for detailed information.
ERROR_CODE_Y	17	R	The error occurred on AXIS-Y. Please refer to ETM_GET_ERROR_CODE() for detailed information.
ERROR_CODE_Z	18	R	The error occurred on AXIS-Z. Please refer to ETM_GET_ERROR_CODE() for detailed information.
ERROR_CODE_U	19	R	The error occurred on AXIS-U. Please refer to ETM_GET_ERROR_CODE() for detailed information.
X_DI_0	20	R	The driving state of AXIS-X: 1: driving, 0: stop
X_DI_1	21	R	The LMT+ state of AXIS-X: 0: off, 1: on
X_DI_2	22	R	The LMT- state of AXIS-X: 0: off, 1: on
X_DI_3	23	R	The EMG state of AXIS-X: 0: off, 1: on
X_DI_4	24	R	The ALARM state of AXIS-X: (please enable Alarm with ETM_SET_ALARM()) 0: off, 1: on
X_DI_5	25	R	The HOME/IN1 state of AXIS-X: 0: on, 1: off

Variable Name	Address	Type	Comment
X_DI_6	26	R	The NHOME/IN0 state of AXIS-X: 0: on, 1: off
X_DI_7	27	R	The IN3 state of AXIS-X: 0: on, 1: off
X_DI_8	28	R	The INPOS or Servo Ready state of AXIS-X: 0: on, 1: off
X_DI_9	29	R	The Z-phase/IN2 state of AXIS-X: 0: on, 1: off
Y_DI_0	30	R	The driving state of AXIS-Y: 1: driving, 0: stop
Y_DI_1	31	R	The LMT+ state of AXIS-Y: 0: off, 1: on
Y_DI_2	32	R	The LMT- state of AXIS-Y: 0: off, 1: on
Y_DI_3	33	R	The EMG state of AXIS-Y: 0: off, 1: on
Y_DI_4	34	R	The ALARM state of AXIS-Y: (please enable Alarm with ETM_SET_ALARM()) 0: off, 1: on
Y_DI_5	35	R	The HOME/IN1 state of AXIS-Y: 0: on, 1: off
Y_DI_6	36	R	The NHOME/IN0 state of AXIS-Y: 0: on, 1: off
Y_DI_7	37	R	The IN3 state of AXIS-Y: 0: on, 1: off
Y_DI_8	38	R	The INPOS or Servo Ready state of AXIS-Y: 0: on, 1: off
Y_DI_9	39	R	The Z-phase/IN2 state of AXIS-Y: 0: on, 1: off
Z_DI_0	40	R	The driving state of AXIS-Z: 1: driving, 0: stop
Z_DI_1	41	R	The LMT+ state of AXIS-Z: 0: off, 1: on
Z_DI_2	42	R	The LMT- state of AXIS-Z: 0: off, 1: on

Variable Name	Address	Type	Comment
Z_DI_3	43	R	The EMG state of AXIS-Z: 0: off, 1: on
Z_DI_4	44	R	The ALARM state of AXIS-Z: (please enable Alarm with ETM_SET_ALARM()) 0: off, 1: on
Z_DI_5	45	R	The HOME/IN1 state of AXIS-Z: 0: on, 1: off
Z_DI_6	46	R	The NHOME/IN0 state of AXIS-Z: 0: on, 1: off
Z_DI_7	47	R	The IN3 state of AXIS-Z: 0: on, 1: off
Z_DI_8	48	R	The INPOS or Servo Ready state of AXIS-Z: 0: on, 1: off
Z_DI_9	49	R	The Z-phase/IN2 state of AXIS-Z: 0: on, 1: off
U_DI_0	50	R	The driving state of AXIS-U: 1: driving, 0: stop
U_DI_1	51	R	The LMT+ state of AXIS-U: 0: off, 1: on
U_DI_2	52	R	The LMT- state of AXIS-U: 0: off, 1: on
U_DI_3	53	R	The EMG state of AXIS-U: 0: off, 1: on
U_DI_4	54	R	The ALARM state of AXIS-U: (please enable Alarm with ETM_SET_ALARM()) 0: off, 1: on
U_DI_5	55	R	The HOME/IN1 state of AXIS-U: 0: on, 1: off
U_DI_6	56	R	The NHOME/IN0 state of AXIS-U: 0: on, 1: off
U_DI_7	57	R	The IN3 state of AXIS-U: 0: on, 1: off
U_DI_8	58	R	The INPOS or Servo Ready state of AXIS-U: 0: on, 1: off

Variable Name	Address	Type	Comment
U_DI_9	59	R	The Z-phase/IN2 state of AXIS-U: 0: on, 1: off
Line_Scan_Completed	60	R	Bit0 of RINT_STATE
MP_Completed	61	R	Bit1 of RINT_STATE
User-Defined_RINT	62	R	Bit2 of RINT_STATE
Continuous_Inp_Interrupt	63	R	Bit3 of RINT_STATE
.....Undefined	64	--	0
.....Undefined	65	--	0
Axes_Error	66	R	Bit6 of RINT_STATE
Module_Error	67	R	Bit7 of RINT_STATE
Module_ID	68	R	i8094H : 0x <b>44</b> i8094A : 0x <b>55</b>
Firmware_Version	69	R	0x0100 means Ver : 1.00
X_STOP_STAUTS	70	R	ETM04_GET_STOP_STATUS() of X-axis (1:stop, 0: moving)
Y_STOP_STAUTS	71	R	ETM04_GET_STOP_STATUS() of Y-axis (1:stop, 0: moving)
Z_STOP_STAUTS	72	R	ETM04_GET_STOP_STATUS() of Z-axis (1:stop, 0: moving)
U_STOP_STAUTS	73	R	ETM04_GET_STOP_STATUS() of U-axis (1:stop, 0: moving)
X_LATCH	80	R	Value in X-axis latch register long type (MSW is at address 80) Ref : ETM_GET_LATCH()
Y_LATCH	82	R	Value in Y-axis latch register long type (MSW is at address 82) Ref : ETM_GET_LATCH()
Z_LATCH	84	R	Value in Z-axis latch register long type (MSW is at address 84) Ref : ETM_GET_LATCH()
U_LATCH	86	R	Value in U-axis latch register long type (MSW is at address 86) Ref : ETM_GET_LATCH()
TCN	90	R	Total Modbus Client Connection Number

## B. Holding Registers (the base address is starting from 0)

Please use FC=16 (i.e., 0x10) to write registers; and use FC=03 to read the status of registers.

Variable Name	Address	Type	Comment
FRnet_DO_0G	0	R/W	Set/Read the FRnet DO group 0
FRnet_DO_1G	1	R/W	Set/Read the FRnet DO group 1
FRnet_DO_2G	2	R/W	Set/Read the FRnet DO group 2
FRnet_DO_3G	3	R/W	Set/Read the FRnet DO group 3
FRnet_DO_4G	4	R/W	Set/Read the FRnet DO group 4
FRnet_DO_5G	5	R/W	Set/Read the FRnet DO group 5
FRnet_DO_6G	6	R/W	Set/Read the FRnet DO group 6
FRnet_DO_7G	7	R/W	Set/Read the FRnet DO group 7
CALL_MPn	8	R/W	This is a simple method to call MP. Write the MPn at this address, the corresponding MPn will be called. MPn= n+20, where n>=1
.....Undefined	9	--	0
.....Undefined	10	--	0
WDT_MP	11	R/W	If WDT event happens, system will call this MP.
WDT_VALUE	12	R/W	Timeout value for the user-defined WDT function. Unit: 1s. Default: 32767
WDT_Enable	13	R/W	1: enable; 0: disable
Lock_IP	20 ~ 23	R/W	Lock/Read_Locked TCP Client IP
UnLock_IP	24	W	Unlock TCP Client IP
ET-M8194H_IP	25~26	R/W	Set ET-M8194H IP
ET-M8194H_Mask	27~28	R/W	Set ET-M8194H Mask
ET-M8194H_Gateway	29~30	R/W	Set ET-M8194H Gateway
LP_X	90	R/W	Logical position of X-axis. It takes two registers. The MSW is located at address 90.
LP_Y	92	R/W	Logical position of Y-axis. It takes two registers. The MSW is located at address 92.

Variable Name	Address	Type	Comment
LP_Z	94	R/W	Logical position of Z-axis. It takes two registers. The MSW is located at address 94.
LP_U	96	R/W	Logical position of U-axis. It takes two registers. The MSW is located at address 96.
EP_X	98	R/W	Encoder feedback position of X-axis. It takes two registers. The MSW is located at address 98.
EP_Y	100	R/W	Encoder feedback position of Y-axis. It takes two registers. The MSW is located at address 100.
EP_Z	102	R/W	Encoder feedback position of Z-axis. It takes two registers. The MSW is located at address 102.
EP_U	104	R/W	Encoder feedback position of U-axis. It takes two registers. The MSW is located at address 104.
CV_X	106	R	Current velocity of X-axis. It takes two registers. The MSW is located at address 106.
CV_Y	108	R	Current velocity of Y-axis. It takes two registers. The MSW is located at address 108.
CV_Z	110	R	Current velocity of Z-axis. It takes two registers. The MSW is located at address 110.
CV_U	112	R	Current velocity of U-axis. It takes two registers. The MSW is located at address 112.
CA_X	114	R	Current acceleration of X-axis. It takes two registers. The MSW is located at address 106.
CA_Y	116	R	Current acceleration of Y-axis. It takes two registers. The MSW is located at address 108.
CA_Z	118	R	Current acceleration of Z-axis. It takes two registers. The MSW is located at address 110.

Variable Name	Address	Type	Comment
CA_U	120	R	Current acceleration of U-axis. It takes two registers. The MSW is located at address 112.
bVAR0	128	R/W	Value of bVAR0. Only low-byte is effective.
bVAR1	129	R/W	Value of bVAR1. Only low-byte is effective.
			bVARn = 128+n
bVAR126	254	R/W	Value of bVAR126. Only low-byte is effective.
bVAR127	255	R/W	Value of bVAR127. Only low-byte is effective.
VAR0	300	R/W	Value of VAR0. It takes two registers. The MSW is located at address 300.
VAR1	302	R/W	Value of VAR1. It takes two registers. The MSW is located at address 302.
			MSW of VARn = 300+2*n LSW of VARn = 300+2*n+1
VAR510	1320	R/W	Value of VAR510. It takes two registers. The MSW is located at address 1320.
VAR511	1322	R/W	Value of VAR511. It takes two registers. The MSW is located at address 1322.
MD0	3000	R/W	Value of MD0. It takes two registers as a <b>long</b> value. The MSW is located at address 3000.
MD1	3002	R/W	Value of MD1. It takes two registers as a <b>long</b> value. The MSW is located at address 3002.
			MSW of MDn = 3000+2*n LSW of MDn = 3000+2*n+1



Variable Name	Address	Type	Comment
MD1022	5044	R/W	Value of MD1022. It takes two registers as a <b>long</b> value. The MSW is located at address 5044.
MD1023	5046	R/W	Value of MD1023. It takes two registers as a <b>long</b> value. The MSW is located at address 5046.
MD1024	5048	R/W	Value of MD0. It takes two registers as a <b>float</b> value. The MSW is located at address 5048.
MD1025	5050	R/W	Value of MD1. It takes two registers as a <b>float</b> value. The MSW is located at address 5050.
			MSW of MDn = $3000+2*n$ LSW of MDn = $3000+2*n+1$
MD2046	7092	R/W	Value of MD2046. It takes two registers as a <b>float</b> value. The MSW is located at address 7092.
MD2047	7094	R/W	Value of MD2047. It takes two registers as a <b>float</b> value. The MSW is located at address 7094.
Sub_Function_Code	8000	W	The mapping of sub function code can refer to table in <b>Appendix C</b> .
Reg1	8001	W	The definitions of parameters depend on the sub functions. Please refer to the table in <b>Appendix C</b> .
Reg2	8002	W	Users can use the sub function call method to implement most of the motion functions, especially for setting functions.
Reg3	8003	W	For getting real-time information from modules, these tables in <b>Appendix A and B</b> are valuable.
Reg4	8004	W	Users can refer to examples in the functions description above to know how to set the parameters.

Variable Name	Address	Type	Comment
			<p>The basic rules of parameter definitions are :</p> <p>BYTE → one register (low-byte)</p> <p>WORD → one register</p> <p>DWORD → two registers</p> <p>long → two registers</p> <p>float → two registers</p>
Reg100	8100	W	

## C. Sub Function Code mapping table

Please use the FC=16 to call the sub-functions; the related starting address is 8000. The following table lists the reference-section, Sub\_Function name and required-registers of the related sub-function-code.

Some code-fields are filled with xxxxxx; this means no sub-function-code is involved. Most of these functions are belong to the Getting command; the FC=3 or FC=4 will be used to get the needed information. Please refer to the MODBUS table in the Appendix A and B.

Section	Sub_Function	code (H)	code (D)	No. of registers
		reg(0)	reg(0)	
<a href="#">2.2</a>	i8094H_REGISTRATION	xxxxxxx	xxxxxxx	
<a href="#">2.2</a>	i8094H_GET_VERSION	xxxxxxx	xxxxxxx	
<a href="#">2.3</a>	ETM_RESET_CARD	0x0A0A	2570	1
<a href="#">2.3</a>	ETM_CLEAR_CARD_BUFFER	0x0A0B	2571	1
<a href="#">2.4</a>	ETM_SET_PULSE_MODE	0x0A0C	2572	3
<a href="#">2.5</a>	ETM_SET_MAX_V	0x0A0D	2573	4
<a href="#">2.6</a>	ETM_SET_HLMT	0x0A0E	2574	4
<a href="#">2.7</a>	ETM_LIMITSTOP_MODE	0x0A0F	2575	3
<a href="#">2.8</a>	ETM_SET_NHOME	0x0A10	2576	3
<a href="#">2.9</a>	ETM_SET_HOME_EDGE	0x0A11	2577	3
<a href="#">2.10</a>	ETM_SET_SLMT	0x0A12	2578	7
<a href="#">2.10</a>	ETM_CLEAR_SLMT	0x0A13	2579	2
<a href="#">2.11</a>	ETM_SET_ENCODER	0x0A14	2580	5
<a href="#">2.12</a>	ETM_SERVO_ON	0x0A15	2581	2
<a href="#">2.12</a>	ETM_SERVO_OFF	0x0A16	2582	2
<a href="#">2.13</a>	ETM_SET_ALARM	0x0A17	2583	4
<a href="#">2.14</a>	ETM_SET_INPOS	0x0A18	2584	4
<a href="#">2.15</a>	ETM_SET_FILTER	0x0A19	2585	4
<a href="#">2.16</a>	ETM_VRING_ENABLE	0x0A1A	2586	4
<a href="#">2.16</a>	ETM_VRING_DISABLE	0x0A1B	2587	2
<a href="#">2.17</a>	ETM_AVTRI_ENABLE	0x0A1C	2588	2
<a href="#">2.17</a>	ETM_AVTRI_DISABLE	0x0A1D	2589	2
<a href="#">2.18</a>	ETM_EXD_MP	0x0A1E	2590	4
<a href="#">2.18</a>	ETM_EXD_FP	0x0A1F	2591	4
<a href="#">2.18</a>	ETM_EXD_CP	0x0A20	2592	4
<a href="#">2.18</a>	ETM_EXD_DISABLE	0x0A21	2593	2
<a href="#">2.19</a>	ETM03_READ_bVAR	xxxxxxx	xxxxxxx	

Section	Sub_Function	code (H)	code (D)	No. of registers
		reg(0)	reg(0)	
<a href="#">2.19</a>	ETM_WRITE_bVAR	0x0A23	2595	3
<a href="#">2.19</a>	ETM03_READ_VAR	xxxxxx	xxxxxx	
<a href="#">2.19</a>	ETM_WRITE_VAR	0x0A25	2597	5
<a href="#">2.20</a>	ETM03_READ_MD	xxxxxx	xxxxxx	
<a href="#">2.20</a>	ETM_WRITE_MD	0x0A27	2599	5
<a href="#">3.1</a>	ETM_SET_LP	0x0A28	2600	4
<a href="#">3.1</a>	ETM_GET_LP	0x0A29	2601	2
<a href="#">3.2</a>	ETM_SET_EP	0x0A2A	2602	4
<a href="#">3.2</a>	ETM_GET_EP	0x0A2B	2603	2
<a href="#">3.3</a>	ETM03_GET_CV	xxxxxx	xxxxxx	
<a href="#">3.4</a>	ETM03_GET_CA	xxxxxx	xxxxxx	
<a href="#">3.5</a>	ETM_GET_DI	0x0A2E	2606	3
<a href="#">3.5</a>	ETM_GET_DI_ALL	0x0A31	2609	2
<a href="#">3.6</a>	ETM_GET_ERROR	0x0A2F	2607	1
<a href="#">3.6</a>	ETM_GET_ERROR_CODE	0x0A30	2608	2
<a href="#">4.1</a>	ETM_FRNET_IN	0x0A32	2610	2
<a href="#">4.2</a>	ETM_FRNET_OUT	0x0A33	2611	4
<a href="#">5.1</a>	ETM_SET_HV	0x0A3C	2620	4
<a href="#">5.2</a>	ETM_HOME_LIMIT	0x0A3D	2621	3
<a href="#">5.3</a>	ETM_SET_HOME_MODE	0x0A3E	2622	8
<a href="#">5.4</a>	ETM_HOME_START	0x0A3F	2623	2
<a href="#">5.5</a>	i8094H_HOME_WAIT	xxxxxx	xxxxxx	
<a href="#">6.1.1</a>	ETM_NORMAL_SPEED	0x0A46	2630	3
<a href="#">6.1.2</a>	ETM_SET_SV	0x0A47	2631	4
<a href="#">6.1.3</a>	ETM_SET_V	0x0A48	2632	4
<a href="#">6.1.4</a>	ETM_SET_A	0x0A49	2633	4
<a href="#">6.1.5</a>	ETM_SET_D	0x0A4A	2634	4
<a href="#">6.1.6</a>	ETM_SET_K	0x0A4B	2635	4
<a href="#">6.1.7</a>	ETM_SET_L	0x0A4C	2636	4
<a href="#">6.1.8</a>	ETM_SET_AO	0x0A4D	2637	4
<a href="#">6.1.9</a>	ETM_FIXED_MOVE	0x0A4E	2638	4
<a href="#">6.1.9</a>	ETM_SET_PULSE	0x0A4F	2639	4

Section	Sub_Function	code (H)	code (D)	No. of registers
		reg(0)	reg(0)	
<a href="#">6.1.10</a>	ETM_CONTINUE_MOVE	0x0A50	2640	4
<a href="#">6.2.1</a>	ETM_AXIS_ASSIGN	0x0A5A	2650	4
<a href="#">6.2.2</a>	ETM_VECTOR_SPEED	0x0A5B	2651	2
<a href="#">6.2.3</a>	ETM_SET_VSV	0x0A5C	2652	3
<a href="#">6.2.4</a>	ETM_SET_VV	0x0A5D	2653	3
<a href="#">6.2.5</a>	ETM_SET_VA	0x0A5E	2654	3
<a href="#">6.2.6</a>	ETM_SET_VD	0x0A5F	2655	3
<a href="#">6.2.7</a>	ETM_SET_VK	0x0A60	2656	3
<a href="#">6.2.8</a>	ETM_SET_VL	0x0A61	2657	3
<a href="#">6.2.9</a>	ETM_SET_VAO	0x0A62	2658	3
<a href="#">6.2.10</a>	ETM_LINE_2D	0x0A63	2659	5
<a href="#">6.2.11</a>	ETM_LINE_3D	0x0A64	2660	7
<a href="#">6.2.12</a>	ETM_ARC_CW	0x0A65	2661	9
<a href="#">6.2.12</a>	ETM_ARC_CCW	0x0A67	2663	9
<a href="#">6.2.13</a>	ETM_CIRCLE_CW	0x0A69	2665	5
<a href="#">6.2.13</a>	ETM_CIRCLE_CCW	0x0A6A	2666	5
<a href="#">6.3.1</a>	ETM_SYNC_ACTION	0x0A6E	2670	14
<a href="#">6.3.2</a>	ETM_SET_COMPARE	0x0A70	2672	6
<a href="#">6.3.3</a>	ETM_GET_LATCH	0x0A71	2673	2
<a href="#">6.3.4</a>	ETM_SET_PRESET	0x0A72	2674	4
<a href="#">6.3.5</a>	ETM_SET_OUT	0x0A73	2675	4
<a href="#">6.3.6</a>	ETM_ENABLE_INT	0x0AAA	2730	1
<a href="#">6.3.6</a>	ETM_DISABLE_INT	0x0AAB	2731	1
<a href="#">6.3.6</a>	ETM_INTFACTOR_ENABLE	0x0AAC	2732	4
<a href="#">6.3.6</a>	ETM_INTFACTOR_DISABLE	0x0AAD	2733	3
<a href="#">6.3.7</a>	ETM_ENABLE_RINT	0x0AAE	2734	1
<a href="#">6.3.7</a>	ETM_DISABLE_RINT	0x0AAF	2735	1
<a href="#">6.3.7</a>	i8094H_RINT_WAIT	xxxxxx	xxxxxx	
<a href="#">6.4.1</a>	i8094H_SET_BYTE_ARRAY	0x0A76	2678	<=101
<a href="#">6.4.1</a>	i8094H_SET_long_ARRAY	0x0A77	2679	<=101
<a href="#">6.4.1</a>	ETM_RECTANGLE	0x0A78	2680	20
<a href="#">6.4.2</a>	ETM_LINE_2D_INITIAL	0x0A7C	2684	9
<a href="#">6.4.2</a>	ETM_LINE_2D_CONTINUE	0x0A7E	2686	6

Section	Sub_Function	code (H)	code (D)	No. of registers
		reg(0)	reg(0)	
<a href="#">6.4.3</a>	ETM_LINE_3D_INITIAL	0x0A7F	2687	10
<a href="#">6.4.3</a>	ETM_LINE_3D_CONTINUE	0x0A81	2689	8
<a href="#">6.4.4</a>	ETM_MIX_2D_INITIAL	0x0A82	2690	10
<a href="#">6.4.4</a>	ETM_MIX_2D_CONTINUE	0x0A84	2692	11
<a href="#">6.4.5</a>	ETM_CONTINUE_INTP	0x0A86	2694	13
<a href="#">6.4.5</a>	I8094H_CONTINUE_INTP_ARRAY	0x0A87	2695	1
<a href="#">6.4.6</a>	ETM_HELIX_3D	0x0A88	2696	15
<a href="#">6.4.7</a>	ETM_RATIO_INITIAL	0x0A8B	2699	11
<a href="#">6.4.7</a>	ETM_RATIO_2D	0x0A8D	2701	5
<a href="#">6.4.8</a>	ETM_LINE_SCAN	0x0A90	2704	7
<a href="#">6.4.8</a>	ETM_LINE_SCAN_START	0x0A91	2705	5
<a href="#">6.4.8</a>	ETM_LINE_SCAN_OFFSET2	0x0A92	2706	3
<a href="#">6.4.9</a>	i8094H_ARC_3D	XXXXXX	XXXXXX	
<a href="#">6.4.10</a>	i8094H_MIX_3D_INITIAL	XXXXXX	XXXXXX	
<a href="#">6.4.10</a>	i8094H_MIX_3D_CONTINUE	XXXXXX	XXXXXX	
<a href="#">6.5.1</a>	ETM_DRV_HOLD	0x0AB4	2740	2
<a href="#">6.5.2</a>	ETM_DRV_START	0x0AB5	2741	2
<a href="#">6.5.3</a>	i8094H_STOP_WAIT	XXXXXX	XXXXXX	
<a href="#">6.5.4</a>	ETM_STOP_SLOWLY	0x0AB7	2743	2
<a href="#">6.5.4</a>	ETM_STOP_SUDDENLY	0x0AB8	2744	2
<a href="#">6.5.4</a>	ETM_VSTOP_SLOWLY	0x0AB9	2745	1
<a href="#">6.5.4</a>	ETM_VSTOP_SUDDENLY	0x0ABA	2746	1
<a href="#">6.5.5</a>	ETM_CLEAR_STOP	0x0ABB	2747	1
<a href="#">6.5.6</a>	ETM_INTP_END	0x0ABC	2748	2
<a href="#">7.1</a>	ETM_LOAD_INITIAL	0x0AC8	2760	1
<a href="#">7.2.1</a>	ETM_MP_CREATE	0x0AC9	2761	2
<a href="#">7.2.1</a>	ETM_MP_CLOSE	0x0ACA	2762	1
<a href="#">7.2.2</a>	ETM_MP_CALL	0x0ACB	2763	2
<a href="#">7.2.3</a>	ETM_MP_SET_VAR	0x0AD2	2770	5
<a href="#">7.2.3</a>	ETM_MP_SET_RVAR	0x0AD3	2771	3
<a href="#">7.2.3</a>	ETM_MP_VAR_CALCULATE	0x0AD4	2772	8
<a href="#">7.2.4</a>	ETM_MP_FOR	0x0AD5	2773	3
<a href="#">7.2.4</a>	ETM_MP_NEXT	0x0AD6	2774	1
<a href="#">7.2.5</a>	ETM_MP_IF	0x0AD7	2775	6

Section	Sub_Function	code (H)	code (D)	No. of registers
		reg(0)	reg(0)	
<a href="#">7.2.5</a>	ETM_MP_ELSE	0x0AD8	2776	1
<a href="#">7.2.5</a>	ETM_MP_IF_END	0x0AD9	2777	1
<a href="#">7.2.6</a>	ETM_MP_TIMER	0x0ADA	2778	3
<a href="#">7.2.7</a>	ETM_MP_STOP_WAIT	0x0ADB	2779	2
<a href="#">7.2.8</a>	ETM_MP_SET_RINT	0x0ADC	2780	1

## D. FRnet DI/O mapping MODBUS Address (FC01, 02, 05, 15)

The FRnet DI/DO of Group 0~7 (8 Group, 128 DI/DO ) can be mapped into contiguous addresses ( total 128 point).

Please use FC= 01, 02 to read multiple FRnet DI/O; with the starting address 0. (Using FC=01 to read FRnet DO; and FC=02 to get FRnet DI).

Please refer to the functions in Section 8.2.5 and 8.2.6.

Variable Name	Address	Type	Comment
FRnet_DO_0G FRnet_DI_8G	0 ~ 15	R	The value of FRnet DO group 0(16-bit) The value of FRnet DI group 8 (16-bit)
FRnet_DO_1G FRnet_DI_9G	16 ~ 31	R	The value of FRnet DO group 1 (16-bit) The value of FRnet DI group 9 (16-bit)
FRnet_DO_2G FRnet_DI_10G	32 ~ 47	R	The value of FRnet DO group 2(16-bit) The value of FRnet DI group 10 (16-bit)
FRnet_DO_3G FRnet_DI_11G	48 ~ 63	R	The value of FRnet DO group 3(16-bit) The value of FRnet DI group 11 (16-bit)
FRnet_DO_4G FRnet_DI_12G	64 ~ 79	R	The value of FRnet DO group 4(16-bit) The value of FRnet DI group 12 (16-bit)
FRnet_DO_5G FRnet_DI_13G	80 ~ 95	R	The value of FRnet DO group 5(16-bit) The value of FRnet DI group 13 (16-bit)
FRnet_DO_6G FRnet_DI_14G	96 ~ 111	R	The value of FRnet DO group 6(16-bit) The value of FRnet DI group 14 (16-bit)
FRnet_DO_7G FRnet_DI_15G	112 ~ 127	R	The value of FRnet DO group 7(16-bit) The value of FRnet DI group 15 (16-bit)

Please use FC=05, 15 to write the FRnet DO; with starting address 0:

(1) Write single FRnet DO with FC=05 ( any point within 0 ~ 127 )

(2) Write multiple FRnet DO with FC=15. Please refer to the functions in Section 8.2.7 and 8.2.8.

Variable Name	Addresses	Type	Comment
FRnet_DO_0G	0 ~ 15	W	The value of FRnet DO group 0 (16-bit)
FRnet_DO_1G	16 ~ 31	W	The value of FRnet DO group 1 (16-bit)
FRnet_DO_2G	32 ~ 47	W	The value of FRnet DO group 2 (16-bit)
FRnet_DO_3G	48 ~ 63	W	The value of FRnet DO group 3 (16-bit)
FRnet_DO_4G	64 ~ 79	W	The value of FRnet DO group 4 (16-bit)
FRnet_DO_5G	80 ~ 95	W	The value of FRnet DO group 5 (16-bit)
FRnet_DO_6G	96 ~ 111	W	The value of FRnet DO group 6 (16-bit)
FRnet_DO_7G	112 ~ 127	W	The value of FRnet DO group 7 (16-bit)



## E. ET-M8194H LED Description



LED description:

LED	Status	Description
Sys	On	Device is switched on and firmware is running.
	Flashing	Device is switched on and firmware is not running.
	Off	Device is switched off.
Tx	Flashing	Data is transmitted by the ET-M8194H via RS-232.
	Off	No data is sent by the ET-M8194H via RS-232.
Rx	Flashing	The device is receiving data via RS-232.
	Off	No data is being received.
NET	On	Device is connected to Ethernet.
	Flashing	Data is being transmitted via Ethernet.
	Off	Device is not connected to the Ethernet.
MOD	On	Module i-8094H is plugged into ET-M8194H device.
	Flashing	A module different than i-8094H is plugged into ET-M8194H device.
	Off	No module is plugged into the ET-M8194H device.

The LED on the i-8094H module is described in the Quick – Start manual for the i-8094H.

## F. Lock\_IP Setting

The connection can be managed with Lock\_IP setting. Three restrictions can be applied to ET-M8194H with Lock\_IP settings:

1. [ **Unique IP** ] :

If only 192.168.0.100 is allowed, assign the identical value to both IPStart and IPEnd.

```
char IPStart[20] = "192.168.0.100";  
char IPEnd[20] = "192.168.0.100";  
ETM_Lock_IP(IPStart, IPEnd);
```

2. [ **Contiguous IP (less than 256 IPs)** ] :

If the connections from 192.168.0.100 to 192.168.0.200 are allowed, assign the IPStart and IPEnd parameters as follows:

```
char IPStart[20] = "192.168.0.100";  
char IPEnd[20] = "192.168.0.200";  
ETM_Lock_IP(IPStart, IPEnd);
```

3. [ **IP Segment** ] :

(A) If the connections from 192.168.0.0 to 192.168.0.255 are allowed (i.e. connections from 192.168.0.xxx are allowed), assign the IPStart and IPEnd parameters as follows:

```
char IPStart[20] = "192.168.0.1";  
char IPEnd[20] = "255.255.255.0";    // MASK  
ETM_Lock_IP(IPStart, IPEnd);
```

(B) If the connections from 192.168.0.xxx to 192.168.1.xxx are allowed, assign the IPStart and IPEnd parameters as follows:

```
char IPStart[20] = "192.168.0.100";  
char IPEnd[20] = "255.255.254.0";    // MASK  
ETM_Lock_IP(IPStart, IPEnd);
```

If the IPEnd is set to "255.255.255.255", only the IP that assigned to IPStart is allowed.

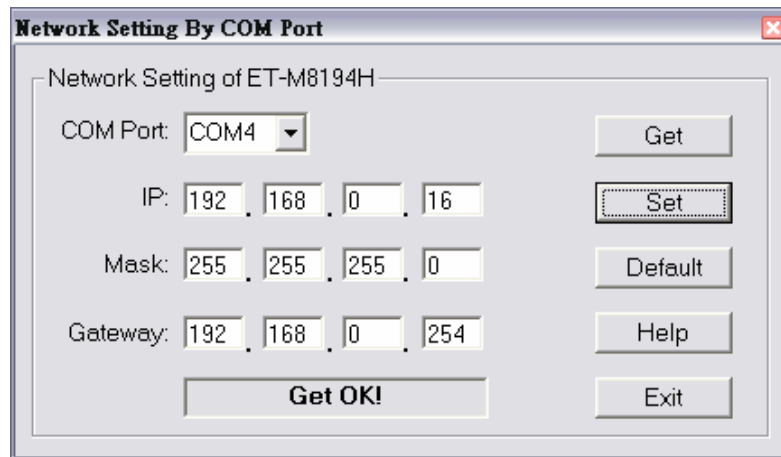
[ Note ]

After rebooting ET-M8194H, the configuration of "Lock IP" will be reset, too. Configure with ETM\_Lock\_IP() again to enable Lock\_IP feature.

## G. IP Configuration

### Method 1 – Setting via RS-232

Execute the EzMove Utility and start the “Network Settings By COM Port” dialog. ([Setting] - [ET-M8194H Setting] – [By COM Port] – [Network])

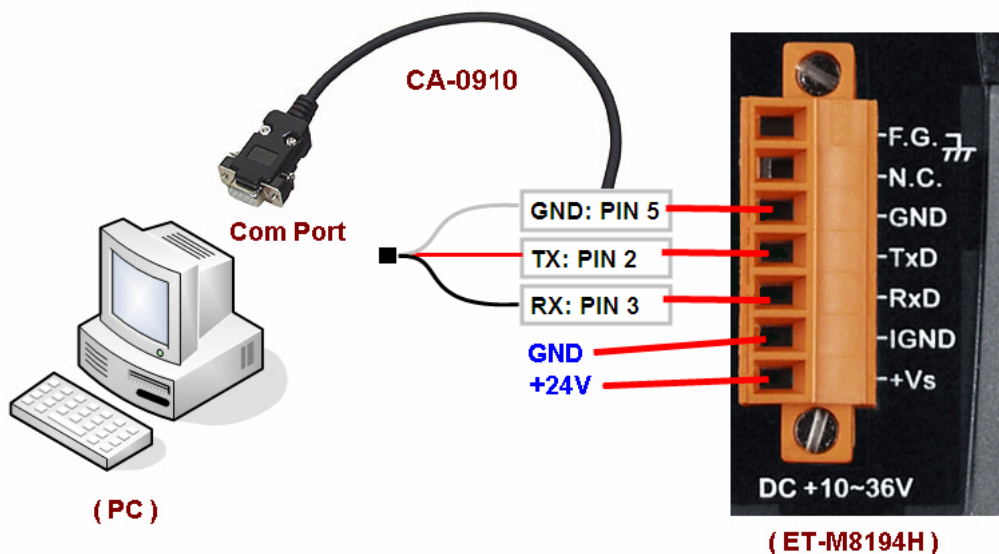


Click **Default** to show the default IP configuration of ET-M8194H:

IP: 192.168.0.16  
Mask: 255.255.255.0  
Gateway: 192.168.0.254

The procedures to read/modify the IP configuration are:

1. Shut down the ET-M8194H.
2. Connect to ET-M8194H with the associated RS-232 cable (CA-0910). The Tx, Rx and GND pins of CA-0910 are connected to the Rx, Tx and GND ports of ET-M8194H. Please connect the other end (9-pin, D-sub connector) to the normal COM port of desktop/laptop.



3. Set the DIP-switch to “Init”.



4. Choose the COM Port, and then click the **Get** button to read back the IP configuration.

5. Fill the settings in IP, Mask and Gateway fields, and then click the **Set** button to change the IP configuration.

6. Click the **Default** button and then click the **Set** button to restore the default IP configuration

7. Shut down the ET-M8194H, and set the DIP-switch to “Run”.



8. Power up the ET-M8194H.

## Method 2 – Setting via Ethernet

Execute the EzMove Utility and connect to ET-M8194H with Ethernet. Start the “ET-M8194H Setting by Ethernet” dialog. ([Setting] – [ET-M8194H Setting] – [By Ethernet] – [Network Tab])

**Get Setting** : read the current IP configuration from ET-M8194H.

**Set** : Fill the new settings in IP, Mask and Gateway fields. Click this button to start IP configuration. The configuration will be completed in 10 seconds, and the EzMove Utility will re-connect to ET-M814H with updated IP address.

**ET-M8194H Setting by Ethernet**

**Network** IP Security

Network Setting

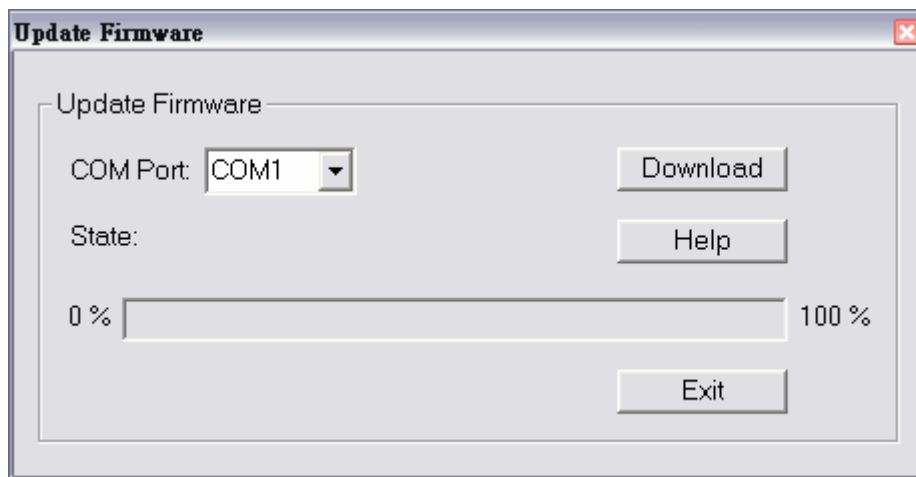
IP: 192 . 168 . 0 . 16

Mask: 255 . 255 . 255 . 0

Gateway: 192 . 168 . 0 . 254

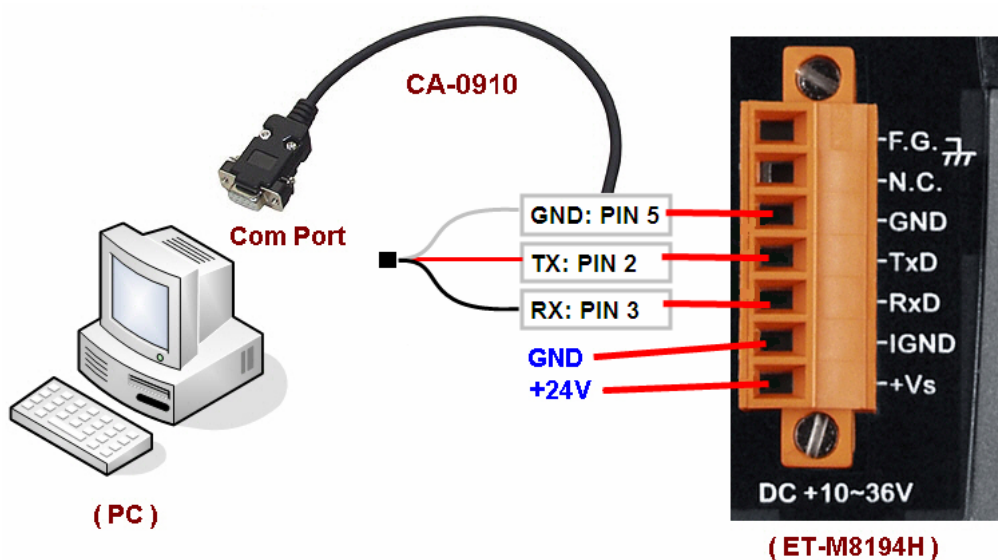
## H. Update Firmware

Execute the *EzMove\_Utility.exe* and open the “Update Firmware” dialog from the menu-bar: [Setting] – [ET-M8194H Setting] – [By COM Port] – [Update Firmware].



Please follow the following procedures to update firmware:

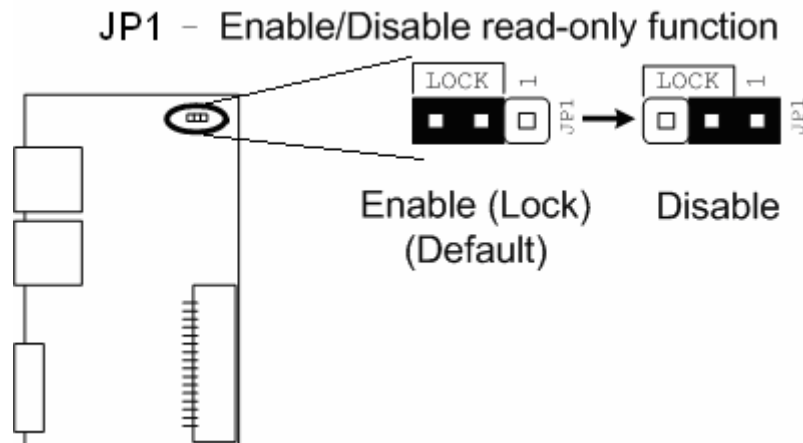
1. Shut down the ET-M8194H.
2. Connect to ET-M8194H with the associated RS-232 cable (CA-0910). The Tx, Rx and GND pins of CA-0910 are connected to the Rx, Tx and GND ports of ET-M8194H. Please connect the other end (9-pin, D-sub connector) to the normal COM port of desktop/laptop.



3. Set the DIP-switch to “Init”.



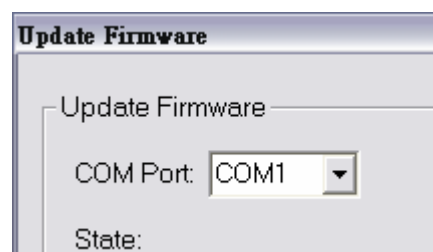
4. Change the setting of JP1 to *disable* ead-only function.



5. Power up the ET-M8194H.

6. Delete the autoexec.bat and EM94H\_10.EXE in the installed folder of EzMove (\\ICPDAS\\ET\_M8194H\\EzMove\_Utility typically).  
Copy the new autoexec.bat and the EM94H\_XX.EXE into the same folder.

7. Choose the COM Port that is connected to ET-M8194H (through CA-0903 cable).



8. Click **Download** button to start firmware-updating procedure.

9. After finishing firmware-updating procedure, shut down the ET-M8194H.

10. Restore the setting of JP1 to *enable* ead-only function.

**11. Set the DIP-switch to “Run”.**



**12. Power up the ET-M8194H.**



## I. History of Versions

Version	Author	Date	Description of changes
v1.0	Edward	13-JAN-2008	The First Version