RS-M8194H
Motion Control Module
User Manual
(Version 3.1)

API Library

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

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.

1 PREFACE....9
1.1 Introduction....9
1.2 Function Categories....10
1.3 API Command Modbus Setup....19
2 COMMUNICATION FUNCTIONS....20
2.1 Serial Communication Functions....20
2.1.1 Open COM Port....20
2.1.2 Close COM Port....22
2.1.3 Get Connection State....23
2.1.4 Set Modbus Response Timeout....24
2.1.5 Get Modbus Response Timeout....25
2.1.6 Set Modbus Request Delay Time....26
2.1.7 Get Modbus Request Delay Time....27
2.1.8 Set Modbus Register Order....28
2.1.9 Get Modbus Register Order....29
2.2 Communication Return Code....30
3 BASIC SETTINGS....33
3.1 Axes Code Definition....33
3.2 Motion Module Reset....34
3.2.1 Restore the Default Settings....34
3.2.2 Clear Command Buffer....35
3.3 Pules Output Mode Setting....36
3.4 Maximum Speed Setting....38
3.5 Trigger Level Setting of Hardware Limit Switch....40
3.6 Motion Stop Mode Setting of Limit Switch....42
3.7 Trigger Level Setting of Near Home Sensor....44
3.8 Trigger Level Setting of Home Sensor....46
3.9 Software Limit Setting....48
3.9.1 Set the Software Limit Value....48
3.9.2 Clear the Software Limit Value....50
3.10 Encoder Related Parameter Settings....51
3.11 Servo Driver (ON/OFF) Setting....54
3.11.1 Servo On....54
3.11.2 Servo Off....56
3.12 Servo Alarm Setting....57
3.13 Active Level Setting of In-Position Signals....59
3.14 Digital Filter Setting....61
3.15 Position Counter Variable Ring Setting....63
3.15.1 Enable the Ring Counter Function....63

3.15.2 Disable the Ring Counter Function....65
3.16 Triangle Velocity Profile Prevention Setting....66
3.16.1 Enable the Triangle Prevention Function....66
3.16.2 Disable the Triangle Prevention Function....68
3.17 External Pulse Input....70
3.17.1 Handwheel (Manual Pulse Generator) Driving....70
3.17.2 Fixed Pulse Driving Mode....72
3.17.3 Continuous Pulse Driving Mode....74
3.17.4 External Signal Input Setting....76
3.18 Read/Write User-Defined Variables (VAR and bVAR)....77
3.18.1 Read Byte Variable....77
3.18.2 Write Byte Variable....79
3.18.3 Read Long Variable....81
3.18.4 Write Long Variable....83
3.19 Read/Write Data for Power Outage Carry-Over (MD)....85
3.19.1 Read the Machine Data....85
3.19.2 Write the Machine Data....87
4 MOTION CONTROLLER STATUS....90
4.1 Set and Read Commanded Position....90
4.1.1 Set the Logical Position Counter....90
4.1.2 Read the Logical Position Counter....92
4.2 Set and Read the Encoder Counter....95
4.2.1 Set the Encoder Position Counter....95
4.2.2 Read the Encoder Position Counter....97
4.3 Set and Read Absolute Position....99
4.3.1 Set the Absolute Position Counter....99
4.3.2 Read the Absolute Position Counter....101
4.4 Get Current Velocity....104
4.5 Get Current Acceleration....106
4.6 Get DI Triggered Motion Status....108
4.6.1 Get Single DI Triggered Motion Status....108
4.6.2 Get All DI Triggered Motion Status....110
4.7 Get DI Signal....112
4.7.1 Get Single DI Signal Status....112
4.7.2 Get All DI Signal Status....114
4.8 Get Home Status....116
4.9 Get and Clear Error Status....119
4.9.1 Check Error Status....119
4.9.2 Get Error Code....121
4.10 Get Buffer Status....124
4.11 Get Stop Status....125
4.12 Get Software Emergency Status....127
4.13 Get Driving Axis....128
4.14 Get Interpolation Ready Flag....129

4.15 Get Triggered Interrupt Factors....130
4.16 Get RS-M8194H State....132
4.16.1 Command Download and Execution Procedure....132
4.16.2 Get RS-M8194H Status....134
5 FRNET FUNCTIONS....138
5.1 Get FRnet DIO Signals....138
5.1.1 Read FRnet Group Data (MP)....138
5.1.2 Read FRnet Channel Data (MP)....140
5.1.3 Read FRnet Channel Data (RTC)....141
5.1.4 Read FRnet Group Data (RTC)....142
5.1.5 Read FRnet Multi-group Data (RTC)....143
5.2 Set FRnet DO....144
5.2.1 Set FRnet DO Group Data (MP)....144
5.2.2 Set FRnet DO Channel Data (MP)....147
5.2.3 Set FRnet DO Channel Data (RTC)....148
5.2.4 Set FRnet DO Group Data (RTC)....149
5.2.5 Set FRnet Multi-group DO Data (RTC)....150
5.3 FRnet Wait....152
5.4 FRnet DI Trigger Event Setting....153
5.5 Get FRnet DI Trigger Event Setting....154
6 AUTO HOMING SEARCH....155
6.1 Set Home Search Speed....156
6.2 Limit Switch as the Home Sensor Setting....157
6.3 Home Search Mode Setting....158
6.4 Start Automatic Home Search Execution....160
7 MOTION CONTROL COMMANDS....161
7.1 Single Axis Motion Control....161
7.1.1 Acceleration/Deceleration Mode Setting....162
7.1.2 Initial Speed Setting....165
7.1.3 Drive Speed Setting....167
7.1.4 Acceleration Setting....168
7.1.5 Deceleration Setting....170
7.1.6 Acceleration Rate Setting....172
7.1.7 Deceleration Rate Setting....174
7.1.8 Offset Setting....176
7.1.9 Relative Distance Motion....178
7.1.10 Change Target Position on the Fly....180
7.1.11 Move to Absoulte Position....182
7.1.12 Continue Pulse Driving Output....184
7.2 Interpolation Commands....186
7.2.1 Interpolation Axes Assignement....186
7.2.2 Speed and Acc/Dec Mode Setting....188

7.2.3 Vector Initial Speed Setting 194
7.2.4 Vector Speed Setting 195
7.2.5 Vector Acceleration Setting 196
7.2.6 Vector Deceleration Setting 198
7.2.7 Vector Acceleration Rate Setting 200
7.2.8 Vector Deceleration Rate Setting 202
7.2.9 Vector Offset Pulses Setting 204
7.2.10 2-Axis Interpolation Relative Distance Motion 206
7.2.11 2-Axis Interpolation Absoult Position Motion 208
7.2.12 3-Axis Interpolation Relative Distance Motion 210
7.2.13 3-Axis Interpolation Absolute Position Motion 212
7.2.14 2-Axis Circular Interpolation Relative Distance Motion (an CW Arc) 214
7.2.15 2-Axis Circular Interpolation Relative Distance Motion (an CCW Arc) 216
7.2.16 2-Axis Circular Interpolation Absoult Position Motion (an CW Arc) 218
7.2.17 2-Axis Circular Interpolation Absoult Position Motion (an CCW Arc) 220
7.2.18 2-Axis Circular Interpolation Motion (Complete CW Circle) 222
7.2.19 2-Axis Circular Interpolation Motion (Complete CCW Circle) 224
7.3 Synchronous Actions 226
7.3.1 Setting the Activation Factors 226
7.3.2 Clear Synchronization Condition 235
7.3.3 Set the Synchronization Condition 236
7.3.4 Set the Synchronization Axis 238
7.3.5 Set the Synchronization Action 239
7.3.6 COMPARE Value Setting 244
7.3.7 Get LATCH Value 246
7.3.8 PRESET Data for Synchronous Action Setting 249
7.3.9 OUT Data Setting 251
7.3.10 Enable Interrupt Functions of i-8094H 253
7.3.11 Disable Interrupt Functions of i-8094H 254
7.3.12 Set Interrupt Factor of i-8094H 255
7.3.13 Clear Interrupt Factor of i-8094H 259
7.4 Continuous Interpolation 261
7.4.1 2-Axis Rectangular Motion 261
7.4.2 Set the Speed of 2-Axis Continuous Linear Interpolation 264
7.4.3 Execute a 2-Axis Continuous Linear Interpolation (Relative Distance) 266
7.4.4 Execute a 2-Axis Continuous Linear Interpolation (Absoult Position) 268
7.4.5 Set the Speed of 3-Axis Continuous Linear Interpolation 270
7.4.6 Execute a 3-Axis Continuous Linear Interpolation (Relative Distance) 272
7.4.7 Execute a 3-Axis Continuous Linear Interpolation (Absoult Position) 274
7.4.8 Set the Speed of Mixed 2-axis motions in Continuous Interpolation 276
7.4.9 Execute a Mixed 2-axis motions in Continuous Interpolation (Relative Distance) 278
7.4.10 Execute a Mixed 2-axis motions in Continuous Interpolation (Absoult Position) 280
7.4.11 3-Axis Helical Motion 282
7.4.12 Set Synchronous Line Scan Motion 284
7.4.13 Start Synchronous Line Scan Motion 286
7.4.14 Get Synchronous Line Scan Motion Status 288
7.5 Stop and Hold Functions 289
7.5.1 Driving Command Hold 289

7.5.2 Release Holding Status and Start Driving 291
7.5.3 Decelerate and Finally Stop Axes 293
7.5.4 Immediately Stop Axes 294
7.5.5 Decelerate and Finally Stop Interpolation Motion....295
7.5.6 Immediately Stop Interpolation Motion....296
7.5.7 Clear Axes Stop Status 297
7.5.8 Clear Interpolation Motion Stop Status....298
7.5.9 End of Interpolation 299
7.5.10 Emergency Stop....300
7.5.11 Clear Emergency Stop 301
8 INITIAL PARAMETER TABLE 302
8.1 Calling the Initial Table 304
9 MACRO PROGRAMMING....306
Introduction 306
9.1 Create MP Macro Program Codes 307
9.1.1 Start a MP Macro Program Codes Programming 307
9.1.2 End a MP Macro Program Codes Programming 309
9.1.3 MP Macro Program Execution....310
9.2 Create ISR Macro Program Codes 313
9.2.1 Start a ISR Macro Program Codes Programming 313
9.2.2 End a ISR Macro Program Codes Programming....314
9.2.3 ISR Maro Program (ISR) Execution....315
9.3 User Defined Variables 316
9.3.1 Assign Macro variable a value 316
9.3.2 Get Command Return Value 318
9.4 Simple Calculations....320
9.5 Command Loop (FOR~NEXT) 323
9.5.1 Start of FOR Loop Block....323
9.5.2 End of FOR Loop Block....326
9.5.3 FOR Loop Block Exit....327
9.6 Conditional Command (IF~ELSE) 328
9.6.1 IF Condition....328
9.6.2 ELSE Statement....332
9.6.3 End of IF Statement....333
9.7 Jump Commands (GOTO-LABEL)....334
9.7.1 GOTO Statement 334
9.7.2 LABEL Statement....336
9.8 Timer 337
9.9 Wait Until Motion Command has been Executed 338
9.10 Exit Macro Program....339
9.11 Terminate all Macro Executions....341
9.12 Get Macro Download Status 343
9.13 Change Velocity on the Fly....346

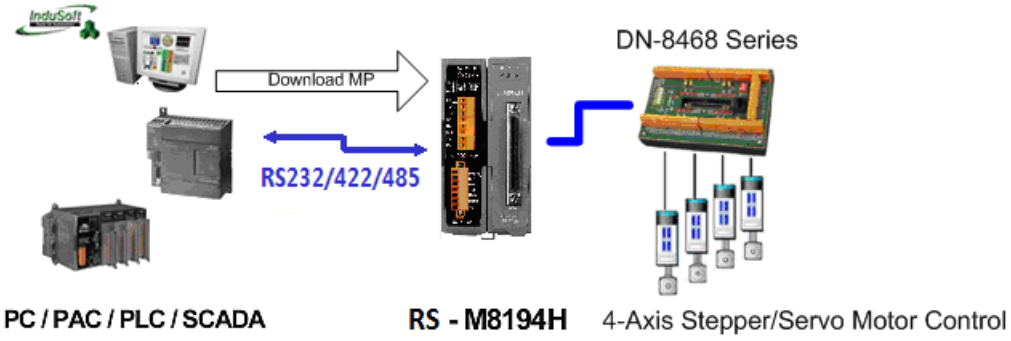
10 STAND ALONE CONTROLLER.....347
11 LIBRARY VERSION.....350
11.1 Modbus P824 Library.....350
11.2 i-8094H Library Version.....351
11.3 RS-M8194H DLL Version.....352
12 APPENDIX.....353
12.1 MODBUS Input Registers.....353
12.1.1 Read FRnet DI/O.....354
12.1.2 Read Motion Chip DI Status of Daughterboard.....355
12.1.3 Read Error Code.....356
12.1.4 Read Logic and Encoder Position, Acceleration, Velocity.....357
12.1.5 Read Stop Status.....358
12.1.6 Read Latch.....359
12.1.7 Read Error State and Free Buffer Size.....360
12.1.8 Read i-8094H Interrupt.....361
12.1.9 Read Firmware Version.....362
12.1.10 Read DI Signal of Daughterboard.....363
12.1.11 Read Absolute Logic Position.....364
12.1.12 Read RS-M8194H and i-8094H State.....365
12.1.13 Macro Program Download Error Messages.....367
12.1.14 Macro Program Execution Error Messages.....368
12.1.15 Motion Chip Triggered Interrupt.....369
12.2 Holding Registers.....370
12.2.1 FRnet DO.....371
12.2.2 Macro Call, FRnet Event and WORD Order Setting.....372
12.2.3 Read/ Write Logic and Encoder Position, Acceleration, Velocity.....373
12.2.4 Read/ Write VAR Variables.....374
12.2.5 Sub Function Code Definition.....375
12.3 Sub-Function Code Mapping Table.....376
12.4 MODBUS Coil Table.....385
12.4.1 FRnet Digital Output.....386
12.4.2 Servo On/Off.....390
12.5 MODBUS Discrete Input Table.....391
12.5.1 FRnet Digital Input.....392
12.5.2 DI or Status of Control Board.....396
12.5.3 Error Stop States.....399
12.5.4 DI Signal of Control Board.....402
13 RS-M8194H LED DESCRIPTION.....404
14 EZMOVE UTILITY.....406
14.1 Serial Communication Parameter Settings.....406
14.2 Firmware Update Procedure.....409

Version	Author	Date	Description of changes
v1.0	Edward	13-JAN-2008	▪ The First Version
v2.0	Allen	13-NOV-2012	▪ The Second Version
V3.0	Martin	12 –MAR-2015	▪ Added absolute position motion commands description. These commands are supported by o RS-M8194H firmware version 3.0 o DLL version 2.0.8
V3.1	Kevin	8–APR-2015	▪ Fix some typo.

1 Preface

1.1 Introduction

RS-M8194H is a RS-232/485/422 serial based 4-axis stepping/pulse-type motion controller and uses Modbus RTU as a communication protocol between master and slave. This intelligent motion controller also has a variety of built in motion control functions, such as 2/3- axis linear interpolation, 2-axis circular interpolation, T/S-curve acceleration/deceleration, various synchronous actions and automatic homing.



The RS-M8194H provides a non-volatile memory for downloading small MACRO programs. A MACRO program basically consists of a sequence of motion instructions. In addition arithmetic operation, conditional (if...else) execution, looping, unconditional jumps and nested MACRO program calls are supported by the module. More than 100 MACRO programs can be downloaded to the module and a MACRO program can contain up to 512 command lines. The MODBUS master can directly call a MACRO program for motion execution. In addition FRnet DI or motion events (e.g. position compare, start / finish of a constant speed drive, etc.) can trigger a MACRO program execution.

In addition the RS-M8194H acts as an FRnet master and can control up to 128 digital outputs and 128 digital inputs. FRnet is a two-wire serial bus and has a scan interval of 2.88 ms and it is specifically designed for easy and cost effective wiring. ICPDAS provides a large range of FRnet I/O terminal boards and modules.

An EzMove utility is provided for configuring the RS-M8194H and assisting the user in writing macro programs and in getting familiar with the RS-M8194H and its motion commands. Furthermore it can be used for motion monitoring and tracking of the motion path.

1.2 Function Categories

The RS-M8194H supports four types of commands:

RTC (Real Time Command): The functions are buffered on a FIFO buffer (DPRAM) and will be executed one by one in the sequence they have arrived. If the buffer is full the RS-M8194H will respond with a MB_SLAVE_DEVICE_BUSY (0x06) exception.

MP (Macro Program): MP is a small compact motion program stored in a non-volatile memory of the i-8094H. Macro programs can be executed by calling MP_CALL with its corresponding number.

ISR (Interrupt Service Routine): An ISR is similar to an MP. The ISR Macro program will be triggerd by an interrupt of the motion chip. The motion chip supports different type of hardware interrupts, like compare. Each interrupt routine can be assigned to a different ISR macro program which will be executed once the corresponding interrupts occurs.

IT (Initial Table): During power on all the configuration made in the initial table will be set. The initial table is intended for initializing the motion chip.

The maximum number of program lines reserved for the MP and ISR Macro programs are listed in the following table.

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			

Note:

- **In the following chapters**
  - **※ indicates functions which are supported by Macro programs MP.**
  - **Δ indicates functions which can be used inside ISR.**

- **In the following table**
  - **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.**

Table 1: Command types and command scope

Section	Function	RTC	MP	ISR	IT
Basic Settings					
3.2.1	RSM_RESET_CARD	◎			
3.2.2	RSM_CLEAR_CARD_BUFFER	◎			
3.3	RSM_SET_PULSE_MODE	◎			◎
3.3	RSM_MACRO_SET_PULSE_MODE		◎		◎
3.4	RSM_SET_MAX_V	◎			◎
3.4	RSM_MACRO_SET_MAX_V		◎		◎
3.5	RSM_SET_HLMT	◎			◎
3.5	RSM_MACRO_SET_HLMT		◎		◎
3.6	RSM_LIMITSTOP_MODE	◎			◎
3.6	RSM_MACRO_LIMITSTOP_MODE		◎		◎
3.7	RSM_SET_NHOME	◎			◎
3.7	RSM_MACRO_SET_NHOME		◎		◎
3.8	RSM_SET_HOME_EDGE	◎			◎
3.8	RSM_MACRO_SET_HOME_EDGE		◎		◎
3.9.1	RSM_SET_SLMT	◎			◎
3.9.1	RSM_MACRO_SET_SLMT		◎		◎
3.9.2	RSM_CLEAR_SLMT	◎			◎

Section	Function	RTC	MP	ISR	IT
3.9.2	RSM_MACRO_CLEAR_SLMT		◎		◎
3.10	RSM_SET_ENCODER	◎			◎
3.10	RSM_MACRO_SET_ENCODER		◎		◎
3.11.1	RSM_SERVO_ON	◎			◎
3.11.1	RSM_MACRO_SERVO_ON		◎		◎
3.11.2	RSM_SERVO_OFF	◎			◎
3.11.2	RSM_MACRO_SERVO_OFF		◎		◎
3.12	RSM_SET_ALARM	◎			◎
3.12	RSM_MACRO_SET_ALARM		◎		◎
3.13	RSM_SET_INPOS	◎			◎
3.13	RSM_MACRO_SET_INPOS		◎		◎
3.14	RSM_SET_FILTER	◎			◎
3.14	RSM_MACRO_SET_FILTER		◎		◎
3.15.1	RSM_VRING_ENABLE	◎			◎
3.15.1	RSM_MACRO_VRING_ENABLE		◎		◎
3.15.2	RSM_VRING_DISABLE	◎			◎
3.15.2	RSM_MACRO_VRING_DISABLE		◎		◎
3.16.1	RSM_AVTRI_ENABLE	◎			◎
3.16.1	RSM_MACRO_AVTRI_ENABLE		◎		◎
3.16.2	RSM_AVTRI_DISABLE	◎			◎
3.16.2	RSM_MACRO_AVTRI_DISABLE		◎		◎
3.17.1	RSM_EXD_MP	◎			
3.17.2	RSM_EXD_FP	◎			
3.17.3	RSM_EXD_CP	◎			
3.17.4	RSM_EXD_DISABLE	◎			
3.18.1	RSM_READ_bVAR	◎			
3.18.2	RSM_WRITE_bVAR	◎			
3.18.3	RSM_READ_VAR	◎			
3.18.4	RSM_WRITE_VAR	◎			
3.19.1	RSM_READ_MD	◎			
3.19.2	RSM_WRITE_MD	◎			
Motion Controller Status					
4.1.1	RSM_SET_LP	◎			
4.1.1	RSM_MACRO_SET_LP		◎	◎	
4.1.2	RSM_GET_LP	◎			

Section	Function	RTC	MP	ISR	IT
4.1.2	RSM_MACRO_GET_LP		◎	◎	
4.1.2	RSM_GET_LP_4_AXIS	◎			
4.2.1	RSM_SET_EP	◎			
4.2.1	RSM_MACRO_SET_EP		◎	◎	
4.2.2	RSM_GET_EP	◎			
4.2.2	RSM_MACRO_GET_EP		◎	◎	
4.2.2	RSM_GET_EP_4_AXIS	◎			
4.3.1	RSM_ABS_SET_POSITION	◎			
4.3.1	RSM_MACRO_ABS_SET_POSITION		◎	◎	
4.3.2	RSM_ABS_GET_POSITION	◎			
4.3.2	RSM_MACRO_ABS_GET_POSITION		◎	◎	
4.3.2	RSM_ABS_GET_POSITION_4_AXIS	◎			
4.4	RSM_GET_CV	◎			
4.4	RSM_GET_CV_4_AXIS	◎			
4.5	RSM_GET_CA	◎			
4.5	RSM_GET_CA_4_AXIS	◎			
4.6.1	RSM_MACRO_GET_DI		◎	◎	
4.6.2	RSM_GET_DI_ALL	◎			
4.6.2	RSM_MACRO_GET_DI_ALL		◎	◎	
4.6.2	RSM_GET_DI_ALL_4_AXIS	◎			
4.7.1	RSM_MACRO_GET_DI_SIGNAL		◎	◎	
4.7.2	RSM_GET_DI_SIGNAL_ALL	◎			
4.7.2	RSM_MACRO_GET_DI_SIGNAL _ALL		◎	◎	
4.7.2	RSM_GET_DI_SIGNAL_ALL_4_AXIS	◎			
4.8	RSM_GET_HOME_SEARCH_STATE	◎			
4.8	RSM_MACRO_GET_HOME_SEARCH_STATE		◎	◎	
4.8	RSM_GET_HOME_SEARCH_STATE _4_AXIS	◎			
4.9.1	RSM_GET_ERROR_STATE	◎			
4.9.1	RSM_MACRO_GET_ERROR		◎	◎	
4.9.2	RSM_GET_ERROR_CODE	◎			
4.9.2	RSM_MACRO_GET_ERROR_CODE		◎	◎	
4.9.2	RSM_GET_ERROR_CODE_4_AXIS	◎			
4.10	RSM_GET_FREE_BUFFER	◎			
4.11	RSM_GET_STOP_STATUS	◎			
4.11	RSM_GET_STOP_STATUS_4_AXIS	◎			

Section	Function	RTC	MP	ISR	IT
4.12	RSM_GET_EMERGENCY_STATE	◎			
4.13	RSM_GET_DRIVING_AXIS	◎			
4.14	RSM_GET_INTERPOL_RDY_FLAG	◎			
4.15	RSM_GET_TRIG_INTFACTOR	◎			
4.16.2	RSM_GET_DEVICE_STATE	◎			
FRnet Functions					
5.1.1	RSM_MACRO_FRNET_IN		◎	◎	
5.1.2	RSM_MACRO_FRNET_READ		◎	◎	
5.1.3	RSM_FRNET_READ_SINGLE_DIO	◎			
5.1.4	RSM_FRNET_READ_GROUP_DIO	◎			
5.1.5	RSM_FRNET_READ_MULTI_GROUP_DIO	◎			
5.2.1	RSM_MACRO_FRNET_OUT		◎	◎	
5.2.2	RSM_MACRO_FRNET_WRITE		◎	◎	
5.2.3	RSM_FRNET_WRITE_SINGLE_DO	◎			
5.2.4	RSM_FRNET_WRITE_GROUP_DO	◎			
5.2.5	RSM_FRNET_WRITE_MULTI_GROUP_DO	◎			
5.3	RSM_MACRO_FRNET_WAIT		◎	◎	
5.4	RSM_SET_FRNET_TRIGGER_EVENT	◎			
5.5	RSM_GET_FRNET_TRIGGER_EVENT_SETTING	◎			
Auto Homing Search					
6.1	RSM_SET_HV	◎			
6.1	RSM_MACRO_SET_HV		◎		
6.2	RSM_HOME_LIMIT	◎			
6.2	RSM_MACRO_HOME_LIMIT		◎		
6.3	RSM_SET_HOME_MODE	◎			
6.3	RSM_MACRO_SET_HOME_MODE		◎		
6.4	RSM_HOME_START	◎			
6.4	RSM_MACRO_HOME_START		◎		
Motion Control Commands					
7.1.1	RSM_NORMAL_SPEED	◎			
7.1.1	RSM_MACRO_NORMAL_SPEED		◎	◎	
7.1.2	RSM_SET_SV	◎			
7.1.2	RSM_MACRO_SET_SV		◎	◎	
7.1.3	RSM_SET_V	◎			
7.1.3	RSM_MACRO_SET_V		◎	◎	
7.1.4	RSM_SET_A	◎			
7.1.4	RSM_MACRO_SET_A		◎	◎	
7.1.5	RSM_SET_D	◎			
7.1.5	RSM_MACRO_SET_D		◎	◎	
7.1.6	RSM_SET_K	◎			

Section	Function	RTC	MP	ISR	IT
7.1.6	RSM_MACRO_SET_K		◎	◎	
7.1.7	RSM_SET_L	◎			
7.1.7	RSM_MACRO_SET_L		◎	◎	
7.1.8	RSM_SET_AO	◎			
7.1.8	RSM_MACRO_SET_AO		◎	◎	
7.1.9	RSM_FIXED_MOVE	◎			
7.1.9	RSM_MACRO_FIXED_MOVE		◎	◎	
7.1.10	RSM_SET_PULSE	◎			
7.1.10	RSM_MACRO_SET_PULSE		◎	◎	
7.1.11	RSM_ABS_FIXED_MOVE	◎			
7.1.11	RSM_ABS_MACRO_FIXED_MOVE		◎	◎	
7.1.12	RSM_CONTINUE_MOVE	◎			
7.1.12	RSM_MACRO_CONTINUE_MOVE		◎	◎	
7.2.1	RSM_AXIS_ASSIGN	◎			
7.2.1	RSM_MACRO_AXIS_ASSIGN		◎	◎	
7.2.2	RSM_VECTOR_SPEED	◎			
7.2.2	RSM_MACRO_VECTOR_SPEED		◎	◎	
7.2.3	RSM_SET_VSV	◎			
7.2.3	RSM_MACRO_SET_VSV		◎	◎	
7.2.4	RSM_SET_VV	◎			
7.2.4	RSM_MACRO_SET_VV		◎	◎	
7.2.5	RSM_SET_VA	◎			
7.2.5	RSM_MACRO_SET_VA		◎	◎	
7.2.6	RSM_SET_VD	◎			
7.2.6	RSM_MACRO_SET_VD		◎	◎	
7.2.7	RSM_SET_VK	◎			
7.2.7	RSM_MACRO_SET_VK		◎	◎	
7.2.8	RSM_SET_VL	◎			
7.2.8	RSM_MACRO_SET_VL		◎	◎	
7.2.9	RSM_SET_VAO	◎			
7.2.9	RSM_MACRO_SET_VAO		◎	◎	
7.2.10	RSM_LINE_2D	◎			
7.2.10	RSM_MACRO_LINE_2D		◎	◎	
7.2.11	RSM_ABS_LINE_2D	◎			
7.2.11	RSM_ABS_MACRO_LINE_2D		◎	◎	
7.2.12	RSM_LINE_3D	◎			
7.2.12	RSM_MACRO_LINE_3D		◎	◎	
7.2.13	RSM_ABS_LINE_3D	◎			
7.2.13	RSM_ABS_MACRO_LINE_3D		◎	◎	
7.2.14	RSM_ARC_CW	◎x2			
7.2.14	RSM_MACRO_ARC_CW		◎x2	◎x2	
7.2.15	RSM_ARC_CCW	◎x2			
7.2.15	RSM_MACRO_ARC_CCW		◎x2	◎x2	
7.2.16	RSM_ABS_ARC_CW	◎x2			
7.2.16	RSM_ABS_MACRO_ARC_CW		◎x2	◎x2	
7.2.17	RSM_ABS_ARC_CCW	◎x2			

Section	Function	RTC	MP	ISR	IT
7.2.17	RSM_ABS_MACRO_ARC_CCW		◎x2	◎x2	
7.2.18	RSM_CIRCLE_CW	◎			
7.2.18	RSM_MACRO_CIRCLE_CW		◎	◎	
7.2.19	RSM_CIRCLE_CCW	◎			
7.2.19	RSM_MACRO_CIRCLE_CCW		◎	◎	
7.3.1	RSM_SYNC_ACTION	◎x2			
7.3.1	RSM_MACRO_SYNC_ACTION		◎x2	◎x2	
7.3.2	RSM_CLEAR_SYNC_ACTION	◎			
7.3.2	RSM_MACRO_CLEAR_SYNC_ACTION		◎	◎	
7.3.3	RSM_SET_ACTIVATION_FACTORS	◎			
7.3.3	RSM_MACRO_SET_ACTIVATION_FACTORS		◎	◎	
7.3.4	RSM_SET_ACTIVATION_AXIS	◎			
7.3.4	RSM_MACRO_SET_ACTIVATION_AXIS		◎	◎	
7.3.5	RSM_SET_ACTION	◎			
7.3.5	RSM_MACRO_SET_ACTION		◎	◎	
7.3.6	RSM_SET_COMPARE	◎			
7.3.6	RSM_MACRO_SET_COMPARE		◎	◎	
7.3.7	RSM_GET_LATCH	◎			
7.3.7	RSM_MACRO_GET_LATCH		◎	◎	
7.3.7	RSM_GET_LATCH_4_AXIS	◎			
7.3.8	RSM_SET_PRESET	◎			
7.3.8	RSM_MACRO_SET_PRESET		◎	◎	
7.3.9	RSM_SET_OUT	◎			
7.3.9	RSM_MACRO_SET_OUT		◎	◎	
7.3.10	RSM_ENABLE_INT	◎			
7.3.10	RSM_MACRO_ENABLE_INT		◎		
7.3.11	RSM_DISABLE_INT	◎			
7.3.11	RSM_MACRO_DISABLE_INT		◎		
7.3.12	RSM_INTFACTOR_ENABLE	◎			
7.3.12	RSM_MACRO_INTFACTOR_ENABLE		◎	◎	
7.3.13	RSM_INTFACTOR_DISABLE	◎			
7.3.13	RSM_MACRO_INTFACTOR_DISABLE		◎	◎	
7.4.1	RSM_RECTANGLE	◎x4			
7.4.1	RSM_MACRO_RECTANGLE		◎x4		
7.4.2	RSM_LINE_2D_INITIAL	◎x2			
7.4.2	RSM_MACRO_LINE_2D_INITIAL		◎x2		
7.4.3	RSM_LINE_2D_CONTINUE	◎			
7.4.3	RSM_MACRO_LINE_2D_CONTINUE		◎		
7.4.4	RSM_ABS_LINE_2D_CONTINUE	◎			
7.4.4	RSM_ABS_MACRO_LINE_2D_CONTINUE		◎		
7.4.5	RSM_LINE_3D_INITIAL	◎x2			
7.4.5	RSM_MACRO_LINE_3D_INITIAL		◎x2		
7.4.6	RSM_LINE_3D_CONTINUE	◎			
7.4.6	RSM_MACRO_LINE_3D_CONTINUE		◎		
7.4.7	RSM_ABS_LINE_3D_CONTINUE	◎			
7.4.7	RSM_ABS_MACRO_LINE_3D_CONTINUE		◎		

Section	Function	RTC	MP	ISR	IT
7.4.8	RSM_MIX_2D_INITIAL	◎x2			
7.4.8	RSM_MACRO_MIX_2D_INITIAL		◎x2		
7.4.9	RSM_MIX_2D_CONTINUE	◎x2			
7.4.9	RSM_MACRO_MIX_2D_CONTINUE		◎x2		
7.4.10	RSM_ABS_MIX_2D_CONTINUE	◎x2			
7.4.10	RSM_ABS_MACRO_MIX_2D_CONTINUE		◎x2		
7.4.11	RSM_HELIX_3D	◎x3			
7.4.11	RSM_MACRO_HELIX_3D		◎x3		
7.4.12	RSM_LINE_SCAN	◎			
7.4.13	RSM_LINE_SCAN_START	◎			
7.4.14	RSM_GET_LINE_SCAN_DONE	◎			
7.5.1	RSM_DRV_HOLD	◎			
7.5.1	RSM_MACRO_DRV_HOLD		◎	◎	
7.5.2	RSM_DRV_START	◎			
7.5.2	RSM_MACRO_DRV_START		◎	◎	
7.5.3	RSM_STOP_SLOWLY	◎			
7.5.3	RSM_MACRO_STOP_SLOWLY		◎		
7.5.4	RSM_STOP_SUDDENLY	◎			
7.5.4	RSM_MACRO_STOP_SUDDENLY		◎		
7.5.5	RSM_VSTOP_SLOWLY	◎			
7.5.5	RSM_MACRO_VSTOP_SLOWLY		◎		
7.5.6	RSM_VSTOP_SUDDENLY	◎			
7.5.6	RSM_MACRO_VSTOP_SUDDENLY		◎		
7.5.7	RSM_CLEAR_STOP	◎			
7.5.7	RSM_MACRO_CLEAR_STOP		◎		
7.5.8	RSM_CLEAR_VSTOP	◎			
7.5.8	RSM_MACRO_CLEAR_VSTOP		◎		
7.5.9	RSM_INTP_END	◎			
7.5.9	RSM_MACRO_INTP_END		◎		
7.5.10	RSM_EMERGENCY_STOP	◎			
7.5.11	RSM_CLEAR_EMERGENCY_STOP	◎			
Initial Parameter Table					
8.1	RSM_LOAD_INITIAL	◎			
Macro Programming					
9.1.1	RSM_MP_CREATE	◎			
9.1.2	RSM_MACRO_MP_CLOSE		◎		
9.1.3	RSM_MP_CALL	◎			
9.1.3	RSM_MACRO_MP_CALL		◎		
9.2.1	RSM_MP_ISR_CREATE	◎			
9.2.2	RSM_MACRO_MP_ISR_CLOSE			◎	
9.2.3	RSM_MP_ISR_CALL	◎			
9.3.1	RSM_MACRO_SET_VAR		◎	◎	
9.3.2	RSM_MACRO_SET_RVAR		◎	◎	
9.4	RSM_MACRO_VAR_CALCULATE		◎	◎	
9.5.1	RSM_MACRO_FOR		◎		

Section	Function	RTC	MP	ISR	IT
9.5.2	RSM_MACRO_NEXT		◎		
9.5.3	RSM_MACRO_EXIT_FOR		◎		
9.6.1	RSM_MACRO_IF		◎	◎	
9.6.2	RSM_MACRO_ELSE		◎	◎	
9.6.3	RSM_MACRO_END_IF		◎	◎	
9.7.1	RSM_MACRO_GOTO		◎		
9.7.2	RSM_MACRO_LABEL		◎		
9.8	RSM_MACRO_TIMER		◎		
9.9	RSM_STOP_WAIT	◎			
9.9	RSM_MACRO_STOP_WAIT		◎		
9.10	RSM_MACRO_EXIT_MACRO		◎	◎	
9.11	RSM_MP_TERMINATE	◎			
9.11	RSM_MACRO_MP_TERMINATE		◎	◎	
9.12	RSM_GET_MP_DOWNLOAD_STATUS	◎			
9.13	RSM_CHANGE_VEL_ON_FLY	◎			
Library Version					
11.1	RSM_GET_RSM_FIRMWARE_VERSION	◎			
11.2	RSM_GET_i8094H_FIRMWARE_VERSION	◎			
11.3	RSM_GET_DLL_VERSION	◎			

1.3 API Command Modbus Setup

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:
The MODBUS RTU frame for the commands is defined as follows:

(1) UID: Unit ID (SlaveAddr).
(2) FC: Function Code.
(3) St_Addr.: Starting Address.
(4) Word Count: The length of the following Registers (in Word, 16-bit).
(5) Byte Count: The length of the following Registers (in Byte, 8-bit).
(6) Register: Content of parameter (16-bit).
(6-1) Sub_Function Code: The pre-defined code for each RS-M8194H function.
(6-2) Axis: The target Axis/Axes of the RS-M8194H function.

The MODBUS Registers only support 16-bit data format. Therefore, two Registers represent a 32-bit value. If the WORD order is set to 0, the first register represents the high-WORD (MSW) and the second register the low-WORD (LSW).

2 Communication Functions

2.1 Serial Communication Functions

In this chapter introduces function which enables the PC or any other device which acts as a Modbus RTU master to open and close a COM port which is necessary in order to communicate with the RS-M8194H device.

2.1.1 Open COM Port

- **HANDLE RSM_OPEN_COM_PORT (BYTE *bPort*,**
  DWORD *dwBaudRate*,
  BYTE *bDataBits*,
  BYTE *bParityBit*,
  BYTE *bStopBits*,
  eCOM_PORT **pError*)

Description:

In order to send data to the RS-M8194H a COM port has to be opened on the PC. After the COM port has been opened successfully Modbus RTU command can be sent to the remote device. The connection is required before sending a MODBUS command. Every connection returns a unique handle.

It is important to close the connection (RSM_CLOSE_COM_PORT ()) before the HANDLE goes out of scope otherwise internal memory reserved for this connection will not be release.

Parameters:

Paramters	Description
***bPort*:**	**Use the port number to which the RS-M8194H is connected to.**
dwBaudRate:	**Use the configured Baudrate of the RS-M8194H. The serial communication parameters (eg. Baudrate, Data bits, etc.) of the RS-M8194H can be set by using the EzMove Utility.**
bDataBits:	**The data bit of RS-M8194H**

bParityBit:	**The parity of RS-M8194H**
bStopBits:	**The stop bit of RS-M8194H**
pError:	▪ **COMM_SUCCESS: COM port has been opened successfully** ▪ **Other errors see error table (Table 2)**

Return: **INVALID_HANDLE_VALUE or NULL indicates a failed conncetion, other values indicates a successful conncetion.**

Example: **[VC++]**

```
#include "RS_M8194H_API.h"

Handle hRSM1;
eRTU iRet;
eCOM_PORT  Error
hRSM1 = RSM_OPEN_COM_PORT(bPort, dwBaudRate,
bDataBits, bParityBit, bStopBits, & Error); // Connection with
the first controller
if( (hRsm1 == INVALID_HANDLE_VALUE) ||
   (Error != COMM_SUCCESS) ) {return;}
iRet = RSM_SERVO_ON (hRSM1, 1, AXIS_XYZU); //Set servo
on
```

2.1.2 Close COM Port

eRTU RSM_CLOSE_COM_PORT (HANDLE *hRsm*, BYTE *bPort*)

Description:

This function closes the serial COM port on the PC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
bPort:	**Serial port number of PC to which the RS-M8194H is connected**

Return: **0: Success; Otherwise: Error (Please refer to chapter 2.2)**

2.1.3 Get Connection State

eRTU RSM_CONNECTION_STATE (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

Shows the status of the serial connection.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return: **1: Connected; Otherwise: Error (Please refer to chapter 2.2)**

2.1.4 Set Modbus Response Timeout

eCOM_PORT RSM_SET_TIMEOUT(HANDLE *hRsm*, DWORD *dwTimeOut*)

Description:

This function sets the time in which the Modbus RTU communication of one command between the PC and the RS-M8194H should be finished. If the communication has not been finished within the defined time the command will return with a timeout.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
dwTimeOut:	**Timeout value in milliseconds. A minimum value of 50 milliseconds is suggested**

Return: **0: Success; Others: Fail (Please refer to chapter 2.2)**

2.1.5 Get Modbus Response Timeout

eRTU RSM_GET_TIMEOUT(HANDLE *hRsm*, DWORD* *dwTimeOut*)

Description:

Get the current Modbus time out setting.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
dwTimeOut:	**Current timeout setting**

Return: **0: Success; Others: Fail (Please refer to chapter 2.2)**

2.1.6 Set Modbus Request Delay Time

eRTU RSM_SET_PACKET_DELAY(HANDLE *hRsm*,
DWORD *dwSendDelay*)

Description:

This function sets the minimum time the Master has to wait after receiving a response from the slave before sending the next request.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
dwSendDelay:	**Delay time after receiving a response before sending the next request. Timeout value in milliseconds. Default value is 0 ms.**

Return: **0: Success; Others: Fail (Please refer to chapter 2.2)**

2.1.7 Get Modbus Request Delay Time

```
eRTU RSM_GET_PACKET_DELAY(HANDLE hRsm,
                          DWORD* dwSendDelay)
```

Description:

Get the current Modbus request delay time setting.

Parameters:

Paramters	Description
hRsm:	COM port handle
dwSendDelay:	Current Modbus request delay time setting

Return: 0: Success; Others: Fail (Please refer to chapter 2.2)

2.1.8 Set Modbus Register Order

eRTU RSM_SET_WORD_ORDER (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bWordOrder*)

Description:

Set the order of the most and least significant WORD (MSW, LSW). This function sets the WORD order of bytes variables (long, unsigned long, float) and determines the position of the first two and last two bytes in the Modbus table. The size of each Modbus register is two bytes long.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
bWordOrder:	**0: MSW at the first Register ; 1: MSW at the second Register**

Return: **0: Success; Others: Fail (Please refer to chapter 2.2)**

2.1.9 Get Modbus Register Order

eRTU RSM_GET_WORD_ORDER (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE* *pbWordOrder*)

Description:

Get the current Modbus register order setting for all the four byte variables.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
bWordOrder:	**0: MSW at the first Register ; 1: MSW at the second Register**

Return: **0: Success; Others: Fail (Please refer to chapter 2.2)**

2.2 Communication Return Code

The error parameter of the COM open command indicates the cause of the open failure

Table 2: Open COM port error table

Open COM port error table		
Error	**Error Code**	**Description**
COMM_SUCCESS	**0**	Open COM port was successfull
COMM_RSM8194H_NOT_CONNECTED	**99**	RS-M8194H module has not been connected to the PC
COMM_PORT_NOT_EXIST	**100**	The COM port does not exist on the PC
COMM_PORT_IN_USE	**101**	The COM port is already open
COMM_PORT_OPEN_ERROR	**102**	The COM port could not be opened
COMM_BAUDRATE_ERR	**103**	The Baudrate entered is not supported
COMM_DATABITS_ERR	**104**	Incorrect data bit size. ICPDAS module only support a data bit size of 8.
COMM_STOPBITS_ERR	**105**	The number of Stop Bits are not supported. The following values are supported: 0(1 stop bit),1(1.5 stop bits), 2(2 stop bits)
COMM_PARITY_ERR	**106**	Wrong Parity value. The following values are supported: 0(NO PARITY), 1(ODD PARITY), 2(EVEN PARITY)
COMM_PORT_NOT_OPEN	**107**	The COM port is not open. The COM port needs to be opened before writing and reading communication starts
COMM_WRITE_ERROR	**108**	Sending data to the COM port failed
COMM_READ_ERROR	**109**	Reading data from the COM port failed
COMM_CLOSE_ERROR	**110**	The open COM port could not be closed
COMM_GET_STATE_ERR	**140**	Internal error --> COM port has been closed
COMM_SET_STATE_ERR	**141**	Internal error --> COM port has been closed
COMM_GET_TIMEOUT_ERR	**142**	Internal error
COMM_SET_TIMEOUT_ERR	**143**	Internal error
COMM_CLEAR_BUFFER_ERROR	**144**	Clearing the input and output COM buffer failed

The return value (eRTU) of the motion commands indicates whether it has been sent successfully to the RS-M8194H:

Table 3: API return values

Communication Error	Code	Description
RSM_SUCCESS	**0**	Successful: command has been received by the RS-M8194H
MB_ILLEGAL_FUNCTION_CODE	**1**	The specifed Modbus function code is not being supported.
MB_ILLEGAL_DATA_ADDRESS	**2**	Invalid starting-address of the MODBUS command. For instance, there are the two Registers are be to accessed (LP_X, LP_Y …). If the WORD Order is set to be 1, then the starting-address is the Low_Word address.
MB_ILLEGAL_DATA_VALUE	**3**	Invalid data of MODBUS command, such as the number of parameters, the target-axis, the addresses of MD, VAR bVAR. An API has been called which is not being supported by the RS-M8194H. Make sure that the API and RS-M8194H firmware version do match.
MB_SLAVE_DEVICE_FAILURE	**4**	Indicates that no i-8094H card is plugged into RS-M8194H.
MB_SLAVE_DEVICE_BUSY	**6**	Indicates the internal buffer (DPRAM-Buffer) of the i-8094H is full and the i8094H is still busy processing commands previously sent; Therefore the failed command has to be resend.
WRONG_CARD_NO	**11**	Invalid card number. Currently only card number 1 is being supported
MB_ILLEGAL_SLAVE_ADDRESS	**201**	Illegal slave address. Legal slave address range 1 to 247
MB_SEND_ERROR	**202**	The data send by the Modbus master to slave is incomplete. Less bytes have been sent to the slave than required by the MODBUS RTU frame. Possible cause: The byte string can not be send by the master within the the timeout period.
MB_TIMEOUT	**203**	The response frame has not been receive within the timeout period
MB_FRAME_LENGTH_ERROR	**206**	The request send to the slave results in a slave response which exceeds the allowable response frame size of 256 bytes Possible causes: The number of register read excceeds the limit

The following definitions refer to errors in the response frame		
MB_INVALID_SLAVE_ADDRESS	**210**	The slave address in the request and response frame is not identical
MB_INVALID_FUNCTION_CODE	**211**	The function code in the request and response frame is not identical
MB_REGISTER_READ_WRITE_ERROR	**212**	Less or more number of registers have been read or written to than requested
MB_STARTING_REGISTER_ERROR	**213**	The first register of the response frame is not identical to the first register address of the request frame
MB_COIL_READ_WRITE_ERROR	**214**	Less or more discrete values have been read or written to than requested
MB_STARTING_COIL_ERROR	**215**	The first discrete address of the response frame is not identical to the first discrete address of the request frame
MB_CRC_ERROR	**216**	The data received at the Master COM port has an invalid CRC
RTU_INVALID_HANDLE	**-5**	COM port has not been opened
RTU_UNDEFINED_ERROR	**-6**	Internal error
RTU_PAR_ERROR	**-7**	Paramter value range error
RTU_WRITE_DENY	**-8**	Access has been denied by write protection
RTU_PORT_NOT_OPEN	**-9**	COM port has not been opened
RTU_COM_PORT_ERROR	**-10**	Internal COM port communiation error

3 Basic Settings

3.1 Axes Code Definition

The definition of axis assignments is as follows: 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. This allows for the axis parameter to assign single axis as well as multiple axes. Available axis codes are listed below:

Table 4: 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 8), the definitions are as follow:

Table 5: Axis assignments and their corresponding codes for initialization table (IT)

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	

IT table can not be used inside macro program (MP or ISR).

3.2 Motion Module Reset

3.2.1 Restore the Default Settings

eRTU RSM_RESET_CARD (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function enables motion module (i-8094H) to restore the power-on default settings.

Category:

MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_RESET_CARD (hRsm, 1); //Reset the module1

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A 0A	***Sub_function code***			

3.2.2 Clear Command Buffer

eRTU RSM_CLEAR_CARD_BUFFER (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

Clear all data in i-8094H command buffer.

Category:

MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_CLEAR_CARD_BUFFER (hRsm, 1); //clear data buffer in module 1

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A 0B	***Sub_function code***			

Use RSM_GET_FREE_BUFFER function to get the available block-number inside the command buffer. The maximum block-number is 30).

3.3 Pules Output Mode Setting

eRTU RSM_SET_PULSE_MODE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nMode*)

※ eRTU RSM_MACRO_SET_PULSE_MODE (HANDLE *hRsm*, BYTE *SlaveAddr*, 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 their direction.

Category:

MODBUS sub_function; RTC, MP and IT.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***nMode*:**	**Assigned mode (Please refer to Table 6)**

Table 6: List of pulse output modes

	mode	Pulse output signal	
		nPP	nPM
CW / CCW	0	CW(rising edge)	CCW(rising 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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_PULSE_MODE is 0A 0C.

The Sub_function code of RSM_MACRO_SET_PULSE_MODE is 0C 0C.

MODBUS example:

RSM_SET_PULSE_MODE (hRsm, 1, AXIS_XYZ, 2);

// set the pulse mode of X, Y, and Z axes to be mode 2 for module 1

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 0C/0C 0C	*Sub_function code*			
1	00 07	*axis*			
2	00 02	*nMode*			

Example:

```
RSM_SET_PULSE_MODE (hRsm, 1, AXIS_XYZ, 2);
// set the pulse mode of X, Y, and Z axes to be mode 2 in module 1
RSM_SET_PULSE_MODE(hRsm, 1, AXIS_U, 3);
//set the pulse mode of U axis to be mode 3 in module 1
RSM_SET_PULSE_MODE (hRsm, 1, INITIAL_XYZU, 0);
//set the pulse mode of X Y Z U axes to be mode 0, write into the initial
parameter table (Table 5) in module 1
```

3.4 Maximum Speed Setting

eRTU RSM_SET_MAX_V (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※ eRTU RSM_MACRO_SET_MAX_V (HANDLE *hRsm*, BYTE *SlaveAddr*, 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. There are 8000 speed segments available. 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:

Paramters	Description
hRsm:	**COM port handle**
SlaveAddr	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
data	**Range of the maximum speed:** **single axis: 8,000~4,000,000 PPS;** **the second axis: 8,000~2,828,854 PPS;** **the third axis: 8,000~2,309,468 PPS.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_MAX_V is 0A 0D.
The Sub_function code of RSM_MACRO_SET_MAX_V is 0C 0D.

MODBUS example:

RSM_SET_MAX_V (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 0D/0C 0D	*Sub_function code*			
1	00 03	*axis*			
2	00 03	*MSW of data*			
3	0D 40	*LSW of data (200000 = 0x30D40)*			

3.5 Trigger Level Setting of Hardware Limit Switch

eRTU RSM_SET_HLMT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nFLEdge*, BYTE *nRLEdge*)

※ eRTU RSM_MACRO_SET_HLMT (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_HLMT is 0A 0E.
The Sub_function code of RSM_MACRO_SET_HLMT is 0C 0E.

MODBUS example:

```
RSM_SET_HLMT (hRsm, 1, AXIS_XYZU, 0, 0);
//set all the trigger levels as low-active for all limit switches on module 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 0E/0C 0E	***Sub_function code***			
1	00 0F	***axis***			
2	00 00	***nFLEdge***			
3	00 00	***nRLEdge***			

3.6 Motion Stop Mode Setting of Limit Switch

eRTU RSM_LIMITSTOP_MODE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nMode*)

※ eRTU RSM_MACRO_LIMITSTOP_MODE (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***nMode*:**	**0: stop immediately; 1: decelerating to stop**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_LIMITSTOP_MODE is 0A 0F.
The Sub_function code of RSM_MACRO_LIMITSTOP_MODE is 0C 0F.

MODBUS example:

```
RSM_LIMITSTOP_MODE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 0F/0C 0F	***Sub_function code***			
1	00 01	***axis***			
2	00 00	***nMode***			

3.7 Trigger Level Setting of Near Home Sensor

eRTU RSM_SET_NHOME (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nNHEdge*)

※ eRTU RSM_MACRO_SET_NHOME (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***nNHEdge*:**	**Active level setting for for the near home sensor** **0 = low active; 1 = high active**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_NHOME is 0A 10.
The Sub_function code of RSM_MACRO_SET_NHOME is 0C 10.

MODBUS example:

```
RSM_SET_NHOME (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 10/0C 10	***Sub_function code***			
1	00 03	***Axis (3 ➔ AXIS_XY)***			
2	00 00	***nNHEdge***			

3.8 Trigger Level Setting of Home Sensor

eRTU RSM_SET_HOME_EDGE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nHEdge*)

※ eRTU RSM_MACRO_SET_HOME_EDGE (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***nHEdge*:**	**Active level setting for the home sensor** **0 = low active; 1 = high active**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_HOME_EDGE is 0A 11.
The Sub_function code of RSM_MACRO_SET_HOME_EDGE is 0C 11.

MODBUS example:

```
RSM_SET_HOME_EDGE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 11/0C 11	*Sub_function code*			
1	00 0F	*Axis (F ➔ AXIS_XYZU)*			
2	00 01	*nHEdge*			

3.9 Software Limit Setting

3.9.1 Set the Software Limit Value

eRTU RSM_SET_SLMT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *dwFL*, long *dwRL*, BYTE *nType*)

※ eRTU RSM_MACRO_SET_SLMT (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_SLMT is 0A 12.
The Sub_function code of RSM_MACRO_SET_SLMT is 0C 12.

MODBUS example:

RSM_SET_SLMT (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 07	0E
Register[]	**Value (hex)**	**Remarks**			
0	0A 12/0C 12	*Sub_function 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*			

3.9.2 Clear the Software Limit Value

eRTU RSM_CLEAR_SLMT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_CLEAR_SLMT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

This function clears the software limits.

Category:

MODBUS sub_function; RTC, MP and IT.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_CLEAR_SLMT is 0A 13.
The Sub_function code of RSM_MACRO_CLEAR_SLMT is 0C 13.

MODBUS example:

RSM_CLEAR_SLMT (hRsm, 1, AXIS_XYZU);
//clear the software limits for all axes on module 1.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 13/0C 13	***Sub_function code***			
1	00 0F	***Axis (F ➔ AXIS_XYZU)***			

3.10 Encoder Related Parameter Settings

eRTU RSM_SET_ENCODER (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*,
BYTE *nMode*, BYTE *nDivision*, BYTE *nZEdge*)

※ eRTU RSM_MACRO_SET_ENCODER (HANDLE *hRsm*, BYTE *SlaveAddr*,
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:

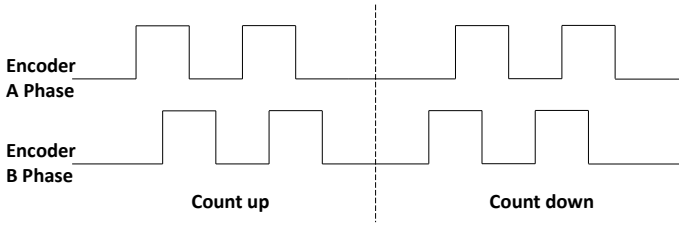
Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***nMode*:**	**Encoder input type:** **0 = A/B quadrature pulse (Count up when A-phase is advancing)** **1 = up/down pulse (A-phase is for counter up)** **2 = A/B quadrature pulse (Count up when B-phase is advancing)** **3 = up/down pulse (B-phase is for counter up)**
***nDivision*:**	**Division setting for A/B quadrature 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**

A/B quadrature pulse input mode:

The count is incremented and decremented at the rising edge and falling edge of both signals. In A/B quadrature pulse input mode, the input pulses can be divided into 1/2 or 1/4.

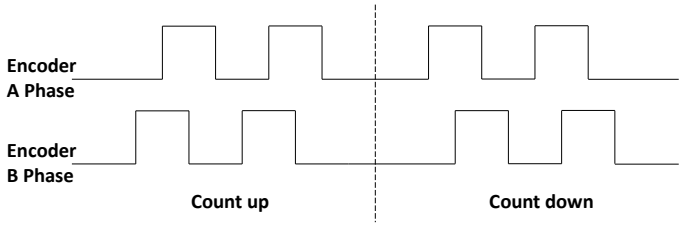
nMode = 0:

When A-phase is advancing with positive logical pulses, the count is incremented; and when the B-phase is advancing, the count is decremented.



nMode = 2:

When B-phase is advancing with positive logical pulses, the count is incremented; and when the A-phase is advancing, the count is decremented.

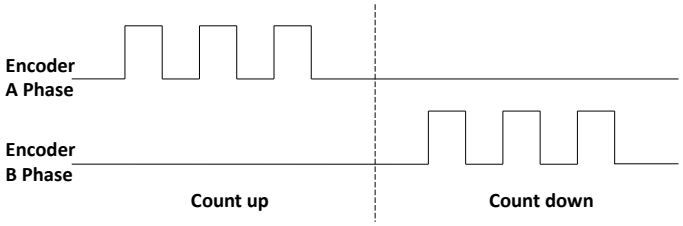


Up/down pulse input mode:

The counter counts at the rising edge of the positive pulse.

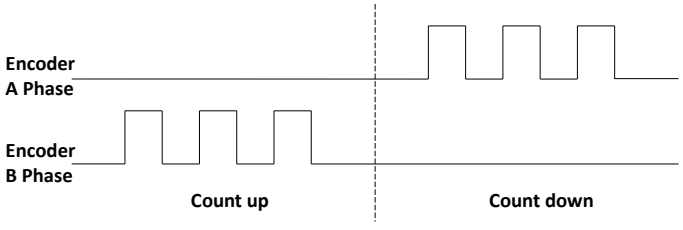
nMode = 1:

A-phase is count up input and B-phase is count down input.



nMode = 3:

B-phase is count up input and A-phase is count down input.

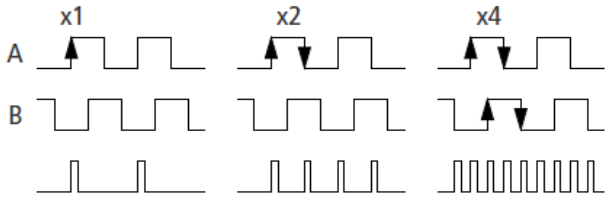


Division setting for A/B quadrature input signals:

nDivision = 0: Counting positive and negative edge of encoder A-phase and B-phase.

nDivision = 1: Counting positive and negative edge of the encoder A-phase only.

nDivision = 2: Considering the positive edge of the encoder A-phase only.



Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_ENCODER is 0A 14.

The Sub_function code of RSM_MACRO_SET_ENCODER is 0C 14.

MODBUS example:

RSM_SET_ENCODER (hRsm, 1, AXIS_XYZU, 0, 0, 0);

//set the encoder input type as A quad B; the division setting is 1/1; and the Z phase is low active.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0A 14/0C 14	*Sub_function 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	*nZEdge (0➔Z phase is low-active)*			

3.11 Servo Driver (ON/OFF) Setting

3.11.1 Servo On

eRTU RSM_SERVO_ON (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_SERVO_ON (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

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

Category:

MODBUS sub_function; RTC, MP and IT.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SERVO_ON is 0A 15.
The Sub_function code of RSM_MACRO_SERVO_ON is 0C 15.

MODBUS example:

```
RSM_SERVO_ON (hRsm, 1, AXIS_XYZU);
//enables all drivers on module 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 15/0C 15	***Sub_function code***			
1	00 0F	***axis (F ➔ AXIS_XYZU)***			

3.11.2 Servo Off

eRTU RSM_SERVO_OFF (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_SERVO_OFF (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

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

Category:

MODBUS sub_function; RTC, MP and IT.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SERVO_OFF is 0A 16.
The Sub_function code of RSM_MACRO_SERVO_OFF is 0C 16.

MODBUS example:

RSM_SERVO_OFF (hRsm, 1, AXIS_XYZU); //disables all drivers on module 1.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 16/0C 16	***Sub_function code***			
1	00 0F	***axis (F ➔ AXIS_XYZU)***			

3.12 Servo Alarm Setting

eRTU RSM_SET_ALARM (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nMode*, BYTE *nAEdge*)

※ eRTU RSM_MACRO_SET_ALARM (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_ALARM is 0A 17.
The Sub_function code of RSM_MACRO_SET_ALARM is 0C 17.

MODBUS example:

RSM_SET_ALARM (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 17/0C 17	*Sub_function code*			
1	00 0C	*axis (C ➔ AXIS_ZU)*			
2	00 01	*nMode (1 ➔ enable)*			
3	00 00	*nAEdge (0 ➔ low-active)*			

3.13 Active Level Setting of In-Position Signals

eRTU RSM_SET_INPOS (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nMode*, BYTE *nIEdge*)

※ eRTU RSM_MACRO_SET_INPOS (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nMode*, BYTE *nIEdge*)

Description:

This function sets the INPOS input signal related parameters.

Category:

MODBUS sub_function; RTC, MP and IT.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***nMode*:**	**Mode:** **0 = disable INPOS input;** **1 = enable INPOS input**
***nIEdge*:**	**Set the trigger level** **0 = low active; 1 = high active**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_INPOS is 0A 18.
The Sub_function code of RSM_MACRO_SET_INPOS is 0C 18.

MODBUS example:

RSM_SET_INPOS (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 18/0C 18	*Sub_function code*			
1	00 01	*axis (1 ➔ AXIS_X)*			
2	00 01	*nMode (1 ➔ enable)*			
3	00 00	*nIEdge (0 ➔ low-active)*			

3.14 Digital Filter Setting

eRTU RSM_SET_FILTER (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *FEn*, BYTE *FLn*)

※ eRTU RSM_MACRO_SET_FILTER (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***FEn*:**	**Enabled filters.** **The sum of the code numbers (0~31) are used to select input signals.**
***FLn*:**	**Sets the filter time constant (0~7).**

***FEn*: 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*: Please refer to the following table.**

Code	Removable max. noise width	Input signal delay time
0	1.75 μSEC	2 μSEC
1	224 μSEC	256 μSEC
2	448 μSEC	512 μSEC
3	896 μSEC	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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_FILTER is 0A 19.
The Sub_function code of RSM_MACRO_SET_FILTER is 0C 19.

MODBUS example:

```
RSM_SET_FILTER (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 19/0C 19	*Sub_function 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)*			

3.15 Position Counter Variable Ring Setting

3.15.1 Enable the Ring Counter Function

eRTU RSM_VRING_ENABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *nVRing*)

※ eRTU RSM_MACRO_VRING_ENABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**
***nVRing*:**	**Maximum value of the ring counter (0 ~ +2,000,000,000)**

Return:

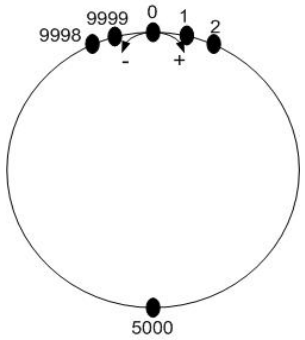
0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_VRING_ENABLE is 0A 1A.
The Sub_function code of RSM_MACRO_VRING_ENABLE is 0C 1A.

MODBUS example:

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



Max. ring encoder value = 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 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 1A/0C 1A	*Sub_function code*			
1	00 01	*axis*			
2	00 00	*MSW of nVRing*			
3	27 0F	*LSW of nVRing (9999 = 0x270F)*			

3.15.2 Disable the Ring Counter Function

eRTU RSM_VRING_DISABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_VRING_DISABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

This function disables the variable ring counter function.

Category:

MODBUS sub_function; RTC, MP and IT.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_VRING_DISABLE is 0A 1B.
The Sub_function code of RSM_MACRO_VRING_DISABLE is 0C 1B.

MODBUS example:

RSM_VRING_DISABLE (hRsm, 1, AXIS_X);
//disable the ring counter function for the X axis on module 1.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 1B/0C 1B	***Sub_function code***			
1	00 01	***axis (1 ➔ AXIS_X)***			

3.16 Triangle Velocity Profile Prevention Setting

3.16.1 Enable the Triangle Prevention Function

eRTU RSM_AVTRI_ENABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_AVTRI_ENABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

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

Category:

MODBUS sub_function; RTC, MP and IT.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_AVTRI_ENABLE is 0A 1C.
The Sub_function code of RSM_MACRO_AVTRI_ENABLE is 0C 1C.

MODBUS example:

```
RSM_AVTRI_ENABLE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 1C/0C 1C	***Sub_function code***			
1	00 01	***axis (1 ➔ AXIS_X)***			

3.16.2 Disable the Triangle Prevention Function

eRTU RSM_AVTRI_DISABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_AVTRI_DISABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4 or Table 5)** **If axis setting is defined in Table 4, the setting values will change current settings.** **If axis setting is defined in Table 5, the setting values will be writted into the parameter table without changing current settings.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_AVTRI_DISABLE is 0A 1D.
The Sub_function code of RSM_MACRO_AVTRI_DISABLE is 0C 1D.

MODBUS example:

```
RSM_AVTRI_DISABLE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 1D/0C 1D	***Sub_function code***			
1	00 01	***axis (1 ➔ AXIS_X)***			

3.17 External Pulse Input

3.17.1 Handwheel (Manual Pulse Generator) Driving

eRTU RSM_EXD_MP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

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

Category:

MODBUS sub_function; RTC.

Parameters:

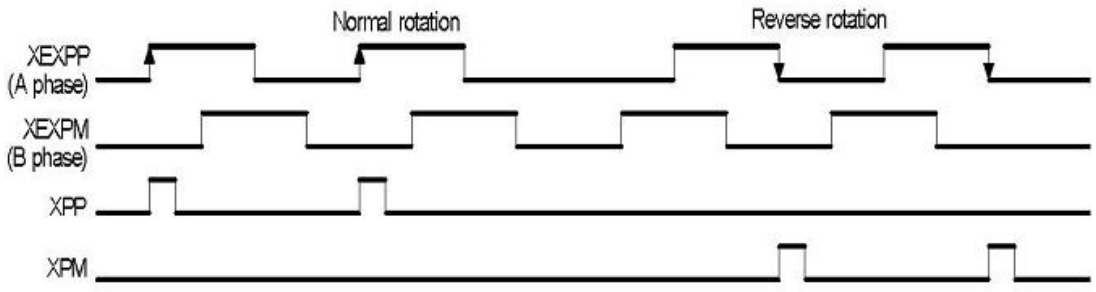
Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***data*:**	**Number of command pulses (Range: 0 ~ +2,000,000,000).**

Return:

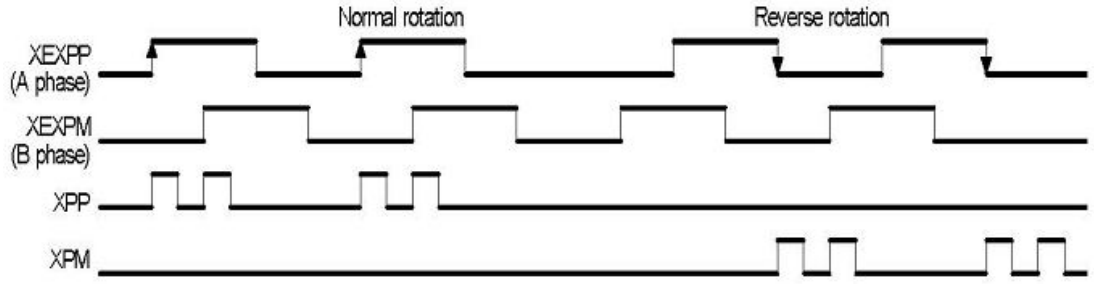
0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
RSM_EXD_MP (hRsm, 1, AXIS_X, 1);
//Each time the handwheel inputs a pulse to the X axis
on module 1, the controller will output 1 pulse to the motor driver.
```



RSM_EXD_MP (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 1E	*Sub_function code*			
1	00 01	*axis (1 ➔ AXIS_X)*			
2	00 00	*MSW of data*			
3	00 01	*LSW of data*			

3.17.2 Fixed Pulse Driving Mode

eRTU RSM_EXD_FP (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

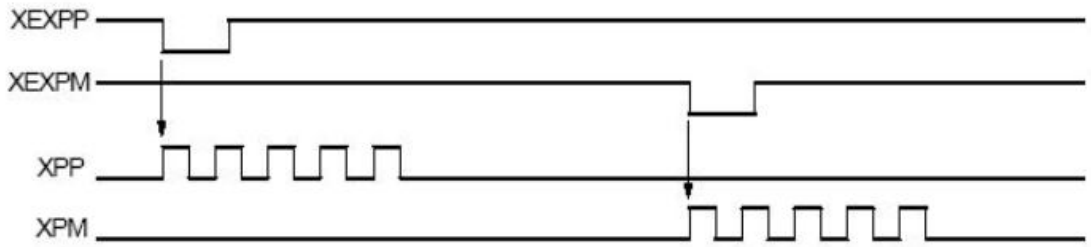
Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***data*:**	**Number of command pulses (Range: 0 ~ +2,000,000,000).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
RSM_EXD_FP (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 1F	*Sub_function code*			
1	00 01	*axis (1 ➔ AXIS_X)*			
2	00 00	*MSW of data*			
3	00 05	*LSW of data*			

3.17.3 Continuous Pulse Driving Mode

eRTU RSM_EXD_CP (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***data*:**	**Speed in PPS (Range: 0 ~ +2,000,000,000).**

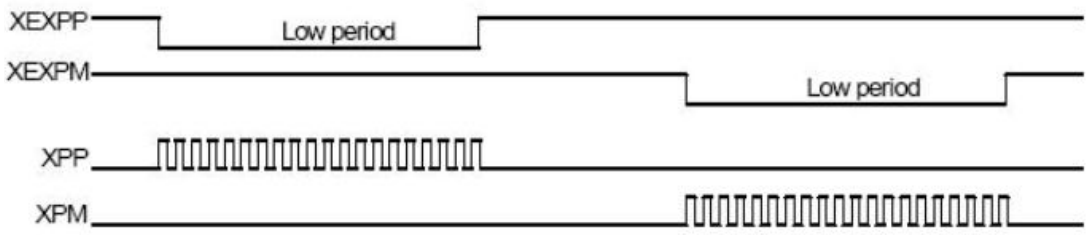
Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_EXD_CP (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 20	*Sub_function code*			
1	00 01	*axis*			
2	00 00	*MSW of data*			
3	00 14	*LSW of data (20 = 0x14)*			

3.17.4 External Signal Input Setting

eRTU RSM_EXD_DISABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

This function turns off the external input driving control functions.

Category:

MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

MODBUS example:

RSM_EXD_DISABLE (hRsm, 1, AXIS_X);
//disable the external input driving control function of X axis on module 1

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 21	***Sub_function code***			
1	00 01	***axis (1 ➔ AXIS_X)***			

3.18 Read/Write User-Defined Variables (VAR and bVAR)

3.18.1 Read Byte Variable

eRTU RSM_READ_bVAR (HANDLE *hRsm*, BYTE *SlaveAddr*, 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 9.3.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***bvarNo*:**	**custom variable: bVAR0 ~ bVAR127**
***bVARValue*:**	**The pointer to the memory that stores bVARn (0 ~ +255).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The corresponding holding register modbus address is as follow:

Macro Variable			
Variable Name	**Address**	**Type**	**Comment**
bVAR0	128	R/W	Value of bVAR0.
bVAR1	129	R/W	Value of bVAR1.
…	…		
bVAR126	254	R/W	Value of bVAR126.
bVAR127	255	R/W	Value of bVAR127.

MODBUS example:

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

BYTE *bVARValue*;
RSM_READ_bVAR (hRsm, 1, bVAR100, &*bVARValue*);
//read value of VAR100 in module 1

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	00 E4	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 Value (hex)
01	03	02	00 64

3.18.2 Write Byte Variable

eRTU RSM_WRITE_bVAR (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bvarNo*, BYTE *bVar*)

Description:

This function writes variable bVARn. bVARn could be passed to Macro Program (MP). For detail information, please refer to Chapter 9.3. bVAR parameter data are stored in a volatile memory.

Category:

MODBUS table, MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***bvarNo*:**	**custom variable: bVAR0 ~ bVAR127**
***bVar*:**	**Variable (0 ~ +255).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The corresponding holding register modbus address is as follow:

Macro Variable			
Variable Name	**Address**	**Type**	**Comment**
bVAR0	128	R/W	Value of bVAR0.
bVAR1	129	R/W	Value of bVAR1.
…	…		
bVAR126	254	R/W	Value of bVAR126.
bVAR127	255	R/W	Value of bVAR127.

MODBUS example:

- **Method 1: use Sub_Function code method：**

RSM_WRITE_bVAR (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 23	*Sub_function 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	00 E4	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	00 64	*bVar (100 = 0x64)*			

3.18.3 Read Long Variable

eRTU RSM_READ_VAR (HANDLE *hRsm*, BYTE *SlaveAddr*, 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 9.3.1.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***varNo*:**	**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 chapter 2.2)

MODBUS example:

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;
RSM_READ_VAR (hRsm, 1, VAR100, &VARValue);
//read value of VAR100 in module 1
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	01 F4	00 02

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count(hex)	Reg_0 (hex)	Reg_1 (hex)
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);

3.18.4 Write Long Variable

eRTU RSM_WRITE_VAR (HANDLE *hRsm*, BYTE *SlaveAddr*, long *varNo*, long *lVar*)

Description:

This function writes variable VARn. It could be passed to Macro Program (MP). For detail information, please refer to Chapter 9. VAR parameter data are stored in a volatile memory.

Category:

MODBUS table, MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***varNo*:**	**custom variable: VAR0 ~ VAR511**
***lVar*:**	**Value of variable (-2,000,000,000 ~ +2,000,000,000).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_WRITE_VAR (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0A 25	***Sub_function code***			
1	7F FF	***MSW of varNo (MSW of 0x7FFF0064)***			
2	00 64	***LSW of varNo (LSW of 0x7FFF0064)***			
3	00 00	***MSW of lVar (MSW 0f 0x2710)***			
4	27 10	***LSW of lVar (LSW 0f 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	01 F4	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	00 00	*MSW of lVar (MSW 0f 0x2710)*			
1	27 10	*LSW of lVar (LSW 0f 0x2710)*			

3.19 Read/Write Data for Power Outage Carry-Over (MD)

3.19.1 Read the Machine Data

eRTU RSM_READ_MD (HANDLE *hRsm*, BYTE *SlaveAddr*, long *mdNo*, long* *ldata*, float* *fdata*)

Description:

This function reads the machine data. MD data are stored in a non-volatile memory.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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 chapter 2.2)

MODBUS example:

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;
RSM_READ_MD (hRsm, 1, MD100, &ldata, 0);
//read MD100 long data in module 1
RSM_READ_MD (hRsm, 1, MD1500, 0, &fdata);
```

//read MD1500 float data in module 1

ldata: the MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	0C 80	00 02

ldata: The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
01	03	04	12 34	56 78

fdata: the MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	17 70	00 02

fdata: The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count(hex)	Reg_0 (hex)	Reg_1 (hex)
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);
```

3.19.2 Write the Machine Data

eRTU RSM_WRITE_MD (HANDLE *hRsm*, BYTE *SlaveAddr*, long *mdNo*, long *ldata*, float *fdata*)

Description:

This function writes the machine data.

Category:

MODBUS table, MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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 chapter 2.2)

MODBUS example:

```
RSM_WRITE_MD (hRsm, SlaveAddr, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 07	0E
Register[]	**Value (hex)**	**Remarks**			
0	0A 27	*Sub_function 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 07	0E
Register[]	**Value (hex)**	**Remarks**			
0	0A 27	*Sub_function 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
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**

Idata: the MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	17 70	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	43 A0	*MSW of MD1500*			
1	D3 33	*LSW of MD1500*			

4 Motion Controller Status

4.1 Set and Read Commanded Position

4.1.1 Set the Logical Position Counter

eRTU RSM_SET_LP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *wdata*)

※Δ eRTU RSM_MACRO_SET_LP (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***wdata*:**	**Position command (-2,000,000,000 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_LP is 0A 28.
The Sub_function code of RSM_MACRO_SET_LP is 0C 28.

MODBUS example:

```
RSM_SET_LP (hRsm, 1, AXIS_XYZU, 10000);
//Set the LP to 10000 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 28/0C 28	*Sub_function 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.**

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 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.

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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
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*			

4.1.2 Read the Logical Position Counter

eRTU RSM_GET_LP(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long* *LPValue*)

※Δ eRTU RSM_MACRO_GET_LP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_GET_LP_4_AXIS(HANDLE *hRsm*, BYTE *SlaveAddr*, long* *LpValueX*, long* *LpValueY*, long* *LpValueZ*, long* *LpValueU*)

Description:

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

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***LPValue*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000).**
***LpValueX*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of x axis.**
***LpValueY*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of y axis.**
***LpValueZ*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of z axis.**
***LpValueU*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of u axis.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

- **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;
RSM_GET_LP (hRsm, 1, AXIS_X, &X_LP);
//Reads the LP value of the X axis on module 1.
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	00 5A	00 02

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
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 use FC = 16 to write this command into an MP. Inside this MP, there can be an RSM_MACRO_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.

```
RSM_MACRO_GET_LP (hRsm, 1, AXIS_X);
//Reads the LP value of the X axis on module 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C 29	***Sub_function code***			
1	00 01	***axis (1 ➔ AXIS_X)***			

4.2 Set and Read the Encoder Counter

4.2.1 Set the Encoder Position Counter

eRTU RSM_SET_EP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *wdata*)

※Δ eRTU RSM_MACRO_SET_EP (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***wdata*:**	**Position command (-2,000,000,000 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_EP is 0A 2A.
The Sub_function code of RSM_MACRO_SET_EP is 0C 2A.

MODBUS example:

```
RSM_SET_EP (hRsm, 1, AXIS_XYZU, 10000);
//Set the EP as 10000 for X, Y, Z, and U axes of module 1.
```

■ **Method 1: use Sub_Function code method**

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 2A/0C 2A	*Sub_function 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 axis 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
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*			

4.2.2 Read the Encoder Position Counter

eRTU RSM_GET_EP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis* , long* *EPValue*)

※Δ eRTU RSM_MACRO_GET_EP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_GET_EP_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, long* *EpValueX*, long* *EpValueY*, long* *EpValueZ*, long* *EpValueU*)

Description:

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

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***EPValue*:**	**The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~ +2,000,000,000).**
EpValueX:	**The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of x axis.**
EpValueY:	**The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of y axis.**
EpValueZ:	**The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of z axis.**
EpValueU:	**The pointer to the memory that stores the encoder position counter (-2,000,000,000 ~+2,000,000,000) of u axis.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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 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.

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

```
long X_EP;
RSM_GET_EP (hRsm, 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:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	00 62	00 02

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count(hex)	Reg_0 (hex)	Reg_1 (hex)
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 RSM_MACRO_SET_RVAR() followed to get the return EP value. Please refer to MP related explanation literature.

```
RSM_MACRO_GET_EP (hRsm, 1, AXIS_X);
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	Value (hex)	Remarks			
0	0C 2B	*Sub_function code*			
1	00 01	*axis (1 ➔ AXIS_X)*			

4.3 Set and Read Absolute Position

4.3.1 Set the Absolute Position Counter

eRTU RSM_ABS_SET_POSITION (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *wdata*)

※Δ eRTU RSM_MACRO_ABS_SET_POSITION (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *wdata*)

Description:

This function assigns the current position to a new logic position value. The new value will be regarded as the new reference position for the absolute position reading and writing.

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
axis:	Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)
wdata:	Absolute Position value (-2,000,000,000 ~ +2,000,000,000)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_ABS_SET_POSITION is 0x0AF4.
The Sub_function code of RSM_MACRO_ABS_SET_POSITION is 0x0CF4.

MODBUS example:

```
RSM_ABS_SET_POSITION (hRsm, 1, AXIS_X, 40000);
//Set the current absolute position of x-axis to 40000
```

■ **Method 1: use sub_function code method**

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A F4/0C F4	*Sub_function code*			
1	00 0F	*axis (F ➔ AXIS_XYZU)*			
2	00 00	*MSW of wdata*			
3	9C 40	*LSW of wdata*			

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

The Start address is 82, or 0x52, for the absolute position of the x-axis. Since the absolute position needs 2 registers to hold the value, the start address setting for MODBUS request must begins from the MSW of the required absolute position. 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	00 52	00 08	10
Register[]	**Value (hex)**	**Remarks**			
0	00 00	*Absolute LP_X_MSW (address = 0x52)*			
1	27 10	*Absolute LP_X_LSW*			
2	00 00	*Absolute LP_Y_MSW (address = 0x54)*			
3	27 10	*Absolute LP_Y_LSW*			
4	00 00	*Absolute LP_Z_MSW (address = 0x56)*			
5	27 10	*Absolute LP_Z_LSW*			
6	00 00	*Absolute LP_U_MSW (address = 0x58)*			
7	27 10	*Absolute LP_U_LSW*			

4.3.2 Read the Absolute Position Counter

eRTU RSM_ABS_GET_POSITION (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long* *LPAbsValue*)

※Δ eRTU RSM_MACRO_ABS_GET_POSITION (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_ABS_GET_POSITION_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, long* *LpAbsValueX*, long* *LpAbsValueY*, long* *LpAbsValueZ*, long* *LpAbsValueU*)

Description:

This function reads the current absolute logic position (logical position counter).

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***LPAbsValue*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000).**
***LpAbsValueX*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of x axis.**
***LpAbsValueY*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of y axis.**
***LpAbsValueZ*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of z axis.**
***LpAbsValueU*:**	**The pointer to the memory that stores the logical position counter (-2,000,000,000 ~+2,000,000,000) of u axis.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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

Since each absolute 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_AbsLP;
RSM_ABS_GET_POSITION (hRsm, 1, AXIS_X, &X_AbsLP);
//Reads the current absolute position LP value of the X axis.
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	00 52	00 02

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
01	03	04	00 00	27 10

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

```
X_AbsLP = Register[0];
X_AbsLP = ((X_AbsLP <<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 use FC = 16 to write this command into an MP. Inside this MP, there can be an RSM_MACRO_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.

```
RSM_MACRO_ABS_GET_POSITION (hRsm, 1, AXIS_X);
//Reads the LP value of the X axis on module 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C F5	*Sub_function code*			
1	00 01	*axis (1 ➔ AXIS_X)*			

4.4 Get Current Velocity

eRTU RSM_GET_CV (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long* *CVValue*)

eRTU RSM_GET_CV_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, long* *CvValueX*, long* *CvValueY*, long* *CvValueZ*, long* *CvValueU*)

Description:

This function reads the current velocity value.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
CVValue:	**The pointer to the memory that stores the current velocity**
CvValueX:	**The pointer to the memory that stores the current velocity of x axis**
CvValueY:	**The pointer to the memory that stores the current velocity of y axis**
CvValueZ:	**The pointer to the memory that stores the current velocity of z axis**
CvValueU:	**The pointer to the memory that stores the current velocity of u axis**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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

```
long CVValue;
RSM_GET_CV (hRsm, 1, AXIS_X, &CVValue);
//reads the current velocity of the X axis on module 1
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	00 6A	00 02

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count(hex)	Reg_0 (hex)	Reg_1 (hex)
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);
```

4.5 Get Current Acceleration

eRTU RSM_GET_CA (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis* , long* *CAValue*)

eRTU RSM_GET_CA_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, long* *CaValueX*, long* *CaValueY*, long* *CaValueZ*, long* *CaValueU*)

Description:

This function reads the current acceleration value PPS/sec.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
CAValue:	**The pointer to the memory that stores the current acceleration (in PPS/Sec).**
CaValueX:	**The pointer to the memory that stores the current acceleration (in PPS/Sec) of x axis.**
CaValueY:	**The pointer to the memory that stores the current acceleration (in PPS/Sec) of y axis.**
CaValueZ:	**The pointer to the memory that stores the current acceleration (in PPS/Sec) of z axis.**
CaValueU:	**The pointer to the memory that stores the current acceleration (in PPS/Sec) of u axis.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

Suppose the CA is 10000, or 0x2710. The Start address is 114, or 0x72.

```
long CAValue;
RSM_GET_CA (hRsm, 1, AXIS_X, &CAValue);
//reads the current acceleration value of the X axis on module 1.
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	00 72	00 02

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	Reg_0 (hex)	Reg_1 (hex)
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);
```

4.6 Get DI Triggered Motion Status

4.6.1 Get Single DI Triggered Motion Status

※Δ eRTU RSM_MACRO_GET_DI (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE nType)

Description:

This function reads the motion chip status triggered by the digital input (DI) signal.

Category:

MODBUS sub_function; MP, ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***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)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

- **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

RSM_MACRO_SET_RVAR() followed to get the return DI value. Please refer to MP related explanation literature.

RSM_MACRO_GET_DI (hRsm, 1, AXIS_X, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0C 2E	***Sub_function code***			
1	00 01	***axis***			
2	00 01	***The specified DI (LIMIT+)***			

4.6.2 Get All DI Triggered Motion Status

eRTU RSM_GET_DI_ALL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis* , WORD* DIValue)

※Δ eRTU RSM_MACRO_GET_DI_ALL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_GET_DI_ALL_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD* *DiValueX*, WORD* *DiValueY*, WORD* *DiValueZ*, WORD* *DiValueU*)

Description:

These function reads the motion chip status triggered by the digital input (DI) signal.

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
axis:	Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)
DIValue:	The pointer to the memory that stores the all DI Status The bits in *DIValue* are defined as follows: *Bit* 0 → DRIVING (Axis is driving or not) *Bit* 1 → LIMIT+ (Limit+ is engaged or not) *Bit* 2 → LIMIT- (Limit- is engaged or not) *Bit* 3 → EMERGENCY (EMG signal) *Bit* 4 → ALARM (ALARM signal) *Bit* 5 → IN1 (IN1 input signal, low-active) *Bit* 6 → IN0 (IN0 input signal,low-active) *Bit* 7 → IN3 (IN3 input signal,low-active) *Bit* 8 → INPOS (INPOS input signal) *Bit* 9 →IN2 (IN2 input signal, low-active)
DiValueX:	The pointer to the memory that stores the all DI Status of x axis
DiValueY:	The pointer to the memory that stores the all DI Status of y axis
DiValueZ:	The pointer to the memory that stores the all DI Status of z axis
DiValueU:	The pointer to the memory that stores the all DI Status of u axis

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

- **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 *DIValue*;
  RSM_GET_DI_ALL (hRsm, 1, AXIS_X, &DIValue);

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 10	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	DI Status of AXIS_X. (Reg_0)
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 RSM_MACRO_SET_RVAR() followed to get the return DI value. Please refer to MP-related explanation literature.

RSM_MACRO_GET_DI_ALL (hRsm, 1, AXIS_X);
//RSM_MACRO_SET_RVAR(hRsm, 1, VAR0); //assign this DI value to //VAR0

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C 31	***Sub_function code***			
1	00 01	***axis (1: AXIS_X)***			

4.7 Get DI Signal

4.7.1 Get Single DI Signal Status

※Δ eRTU RSM_MACRO_GET_DI_SIGNAL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nType*)

Description:

This function reads the digital input (DI) signal.

Category:

MODBUS sub_function; MP, ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
SlaveAddr:	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
nType:	**0 → NHOME/IN0 signal 1 → HOME/IN1 signal 2 → Index/IN2 signal 3 → IN3 signal 4 → EXP+ (External input) signal 5 → EXP- (External input) signal 6 → INP signal 7 → ALARM signal**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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

The read DI signal 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 RSM_MACRO_SET_RVAR() followed to get the return DI signal value. Please refer to MP related explanation literature.

RSM_MACRO_GET_DI_SIGNAL (hRsm, 1, AXIS_X, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0C 40	***Sub_function code***			
1	00 01	***axis***			
2	00 01	***The specified DI signal (*HOME*)***			

4.7.2 Get All DI Signal Status

eRTU RSM_GET_DI_SIGNAL_ALL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis* , WORD* *DISignalValue*)

※Δ eRTU RSM_MACRO_GET_DI_SIGNAL _ALL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_GET_DI_SIGNAL_ALL_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD* *DiSignalValueX*, WORD* *DiSignalValueY*, WORD* *DiSignalValueZ*, WORD* *DiSignalValueU*)

Description:

This function reads the digital input (DI) signal.

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
SlaveAddr:	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
DISignalValue:	**The pointer to the memory that stores the all DI Status The bits in DIValue are defined as follows: Bit 0 → NHOME/IN0 signal Bit 1 → HOME/IN1 signal Bit 2 → Index/IN2 signal Bit 3 → IN3 signal Bit 4 → EXP+ (External input) signal Bit 5 → EXP- (External input) signal Bit 6 → INP signal Bit 7 → ALARM signal**
DiSignalValueX:	**The pointer to the memory that stores the all DI Status of x axis**
DiSignalValueY:	**The pointer to the memory that stores the all DI Status of y axis**
DiSignalValueZ:	**The pointer to the memory that stores the all DI Status of z axis**
DiSignalValueU:	**The pointer to the memory that stores the all DI Status of u axis**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

- **Method 1: It is used for getting current DI signal.**

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

  WORD *DIValue*;
  RSM_GET_DI_SIGNAL_ALL (hRsm, 1, AXIS_X, &DIValue);

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 58	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	DI Status of AXIS_X. (Reg_0)
01	04	02	00 00

- **Method 2: When it is used inside an MP.**

The get DI signal function 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 RSM_MACRO_SET_RVAR() followed to get the return DI value. Please refer to MP-related explanation literature.

RSM_MACRO_GET_DI_SIGNAL_ALL (hRsm, 1, AXIS_X);
//RSM_MACRO_SET_RVAR(hRsm, 1, VAR0); //assign this DI value to //VAR0

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C 41	***Sub_function code***			
1	00 01	***axis (1: AXIS_X)***			

4.8 Get Home Status

eRTU RSM_GET_HOME_SEARCH_STATE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis* , WORD* *HomeStatus*)

※Δ eRTU RSM_MACRO_ GET_HOME_SEARCH_STATE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_GET_HOME_SEARCH_STATE _4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD* *HomeStatusX*, WORD* *HomeStatusY*, WORD* *HomeStatusZ*, WORD* *HomeStatusU*)

Description:

This function reads the home search state.

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
SlaveAddr:	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
HomeStatus:	**The pointer to the memory that stores the home status The value in HomeStatus are defined as follows: 0 → Waits for an automatic home search execution command 3 → Execution step 1 : Waits for activation of the IN0 signal in the specified search direction 8 → Execution step 2: Waits for activation of the IN1 signal in the direction opposite to the specified search direction (irregular operation) 12→Execution step 2: Waits for detivation of the IN1 signal in the direction opposite to the specified search direction (irregular operation) 15→Execution step 2: Waits for activation of the IN1 signal in the specified search direction 20→Execution step 3: Waits for activation of the IN2 signal in the specified search direction**

	25→Execution step 4: Offset driving in the specified search direction
HomeStatusX:	**The pointer to the memory that stores the home status of x axis**
HomeStatusY:	**The pointer to the memory that stores the home status of y axis**
HomeStatusZ:	**The pointer to the memory that stores the home status of z axis**
HomeStatusU:	**The pointer to the memory that stores the home status of u axis**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

- **Method 1: It is used for getting current home state**

```
WORD HomeStatus;
RSM_GET_HOME_SEARCH_STATE (hRsm, 1, AXIS_X, &HomeStatus);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 5C	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	Home State of AXIS_X. (Reg_0)
01	04	02	00 00

- **Method 2: When it is used inside an MP.**

The get home state function 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 RSM_MACRO_SET_RVAR() followed to get the return home state value. Please refer to MP-related explanation literature.

```
RSM_MACRO_GET_HOME_STATE(hRsm, 1, AXIS_X);
//RSM_MACRO_SET_RVAR(hRsm, 1, VAR0);  //assign this home state
//value to VAR0
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C 42	***Sub_function code***			
1	00 01	***axis (1: AXIS_X)***			

4.9 Get and Clear Error Status

4.9.1 Check Error Status

eRTU RSM_GET_ERROR_STATE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE* *ErrState*)

※Δ eRTU RSM_MACRO_GET_ERROR (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function checks whether an error occured or not.

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
ErrState:	**The pointer to the memory that stores the error status:** **1: Error(s) occured.** **Please perform RSM_GET_ERROR_CODE () to get more information.** **0: No error.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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

```
BYTE ErrState;
RSM_GET_ERROR_STATE (hRsm, 1, &ErrState);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 46	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count(hex)	Error status (Reg_0)
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 RSM_MACRO_SET_RVAR() followed to get the return value. Please refer to MP related explanation literature.

RSM_MACRO_GET_ERROR (hRsm, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C 2F	***Sub_function code***			

4.9.2 Get Error Code

eRTU RSM_GET_ERROR_CODE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, WORD* *ErrCode*)

※Δ eRTU RSM_MACRO_GET_ERROR_CODE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_GET_ERROR_CODE_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD* *ErrCodeX*, WORD* *ErrCodeY*, WORD* *ErrCodeZ*, WORD* *ErrCodeU*)

Description:

This function reads the ERROR status.

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
ErrCode:	**T*he pointer to the memory that stores the error status***
ErrCodeX:	**The pointer to the memory that stores the error status of x axis**
ErrCodeY:	**The pointer to the memory that stores the error status of y axis**
ErrCodeZ:	**The pointer to the memory that stores the error status of z axis**
ErrCodeU:	**The pointer to the memory that stores the error status of u axis**

ErrCode:

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	SOFT STOP	Occurs when the software stop is asserted(use function RSM_STOP_SLOWLY, RSM_STOP_SUDDENLY, RSM_VSTOP_SLOWLY, RSM_VSTOP_SUDDENLY)
512	SOFT EMG	Occurs when the software EMG is asserted(use function RSM_EMERGENCY_STOP)

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

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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

WORD *ErrCode*;
RSM_GET_ERROR_CODE (hRsm, 1, AXIS_X, &*ErrCode*);

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 14	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count(hex)	ErrCode (Reg_0)
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 RSM_MACRO_SET_RVAR() followed to get the return value. Please refer to MP related explanation literature.

RSM_MACRO_GET_ERROR_CODE (hRsm, 1, AXIS_X);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C 30	*Sub_function code*			
1	00 01	*axis (AXIS_X)*			

4.10 Get Buffer Status

eRTU RSM_GET_FREE_BUFFER(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE* *FreeBufNum*)

Description:

This function returns the number of commands the internal FIFO buffer can still store. A maximum of 30 commands can be stored by the FIFO can store. All arriving commands are first stored in the FIFO buffer before they are processed.

Category:

MODBUS table；RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***FreeBufNum*:**	**Number of empty command lines in the FIFO buffer of the i-8094H**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
BYTE FreeBufNum;
RSM_GET_FREE_BUFFER(hRsm, 1, & FreeBufNum);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 47	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	*Block_Num* (hex)
01	04	02	00 1E

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

4.11 Get Stop Status

eRTU RSM_GET_STOP_STATUS(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE* *StopState*)

eRTU RSM_GET_STOP_STATUS_4_AXIS(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE* *StopStateX*, BYTE* *StopStateY*, BYTE* *StopStateZ*, BYTE* *StopStateU*)

Description:

This function reads current stop status.

Category:

MODBUS table；RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***StopState*:**	**The pointer to the memory that stores the stop status**
***StopStateX*:**	**The pointer to the memory that stores the stop status of x axis**
***StopStateY*:**	**The pointer to the memory that stores the stop status of y axis**
***StopStateZ*:**	**The pointer to the memory that stores the stop status of z axis**
***StopStateU*:**	**The pointer to the memory that stores the stop status of u axis**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

STOP status	Address	Remarks
X_STOP_STATUS	58 (0x3A)	1 ➔ AXIS_X has stopped 0 ➔ AXIS_X is moving
Y_STOP_STATUS	59 (0x3B)	1 ➔ AXIS_Y has stopped 0 ➔ AXIS_Y is moving
Z_STOP_STATUS	60 (0x3C)	1 ➔ AXIS_Z has stopped 0 ➔ AXIS_Z is moving
U_STOP_STATUS	61 (0x3D)	1 ➔ AXIS_U has stopped 0 ➔ AXIS_U is moving

```
BYTE StopState;
RSM_GET_STOP_STATUS (hRsm, 1, AXIS_X, & StopState);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 3A	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	AXIS Stop_Status (hex)
01	04	02	00 01

4.12 Get Software Emergency Status

eRTU RSM_GET_EMERGENCY_STATE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE * *EmgState*)

Description:

This function reads the current emergency status generated by the RSM_CLEAR_EMERGENCY_STOP command. In an emergency stop state all four drives are in standstill and remain in standstill until the emergency stop has bee cleared.

Category:

MODBUS table: RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
EmgState:	**▪ 0: no emergency stop active ▪ 1: emergency stop activated**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

Emergency stop status	Address	Remarks
X_STOP_STATUS	25 (0x19)	1 ➔ AXIS_X has stopped 0 ➔ AXIS_X is moving

```
BYTE EmgState;
RSM_GET_EMERGENCY_STATE (hRsm, 1, & EmgState);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count(hex)
01	04	00 19	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count(hex)	EmgState (hex)
01	04	02	00 01

4.13 Get Driving Axis

eRTU RSM_GET_DRIVING_AXIS(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE* *DrvAxis*)

Description:

This function returns the currently driving axis.

Category:

MODBUS table: RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
DrvAxis:	▪ **0 - no axis running;** ▪ **>0 - each running axis is indicated by a bit (see table Table 4)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
BYTE DrvAxis;
RSM_GET_DRIVING_AXIS (hRsm, 1, & DrvAxis);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 60	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	Driving Axis (hex)
01	04	02	00 01

4.14 Get Interpolation Ready Flag

eRTU RSM_GET_INTERPOL_RDY_FLAG(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE* *InterpolationRdyFlag*)

Description:

This function indicates whether the motion control chip is ready to process the next interpolation command.

Category:

MODBUS table: RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***InterpolationRdyFlag*:**	▪ **0 - not ready for the next interpolation command** ▪ **1 - ready for the next interpolation command**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
BYTE InterpolationRdyFlag;
RSM_GET_INTERPOL_RDY_FLAG (hRsm, 1, & InterpolationRdyFlag);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 57	00 01

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	InterpolationRdyFlag (hex)
01	04	02	00 01

4.15 Get Triggered Interrupt Factors

eRTU RSM_GET_TRIG_INTFACTOR (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nINT*, BYTE **pAxis*)

Description:

This function reads which axis has triggered an interrupt for the set interrupt factor since the last RSM_INTFACTOR_ENABLE function call. The function RSM_INTFACTOR_ENABLE enables the motion chip interrupt and clears the interrupt status for the respective interrupt factor.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
nINT:	**Consult Table 7**
pAxis:	**Consult Table 4** **Example:** **0x0F: All 4 axis have triggered** **0x02: only the Y axis has triggered** **0x0B: axis XYU have triggered**

The Modbus function code 4 register table is as follows.

Table 7: Motion chip interrupt factor table with respective Modbus FC 4 register addresses

Modbus register address FC4	*nINT*	Symbol	Description
130	0	PULSE	Interupt occurs when pulse is up
131	1	P>=C-	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register.
132	2	P<C-	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register

133	3	P<C+	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register.
134	4	P>=C+	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register.
135	5	C-END	Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output
136	6	C-STA	Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output
137	7	D-END	Interrupt occurs when the driving is finished
138	8	HMEND	Automatic home search terminated
139	9	SYNC	Synchronous action was activated

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
BYTE bAxis;
RSM_GET_TRIG_INTFACTOR (hRsm, 1, 0, & bAxis);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01 (Slave Address)	04	00 82 (130) (Pulse output)	00 01

MODBUS response:

UID (hex)	FC (hex)	Byte Count (hex)	Registers(hex)
01	04	02	00 09 (Axis XY triggered)

4.16 Get RS-M8194H State

4.16.1 Command Download and Execution Procedure

The motion unit basically consists of two modules:
- **RS-M8194H**
- **i-8094H**

• The RS-M8194H module (Figure 1) in slot 0 is the unit which is mainly responsible for the Modbus communication between the Master and Slave. The main task of the module is to verify and filter every command arriving at its serial port according to its current state and consequently decide whether to transfer the command to the i-8094H or, if the command is not supported by the respective state, ignore the command and respond to the Master with a Modbus exception code.

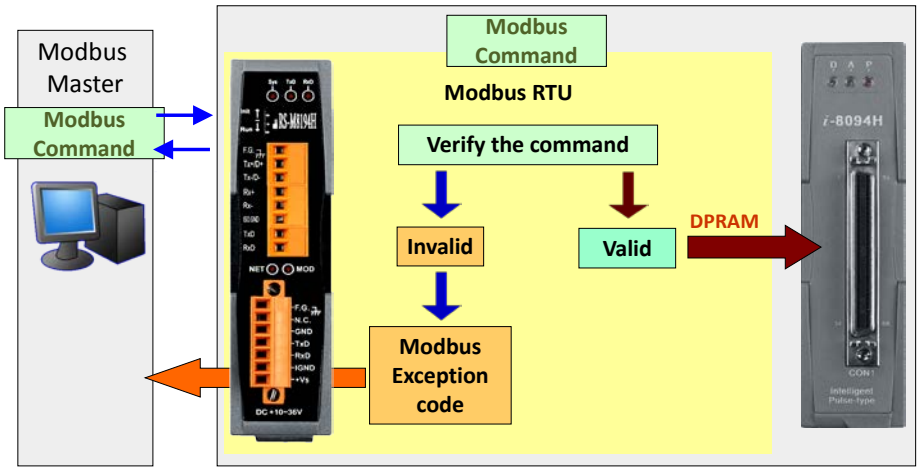


Figure 1: RS-M8194H module in slot 0 of the motion device

The module can be in one of the following three states:
- **Ready state**
- **MP Macro download state**
- **ISR Macro download state**

When the RS-M8194H (module in the first slot is responsible for the communication part) is in a certain state certain commands can not be processed by the module and therefore the module responds with a Modbus exception code when a command is being sent by the Master which is not being supported by the current module state. Table 1 describes for each command the valid scope. When the module is in the ready state all RTC commands marked in Table 1 are valid commands.

For example, the RS-M8194H is in the MP Macro download state. In this state only

commands which are marked with MP in Table 1 are valid commands. As soon as a non MP command is being sent (e.g. MP_CREATE) the module will ignore this command and return a Modbus exception code. The RSM_GET_DEVICE_STATE allows the user to determine the reason for the Modbus exception code by reading the current module state.

• The i-8094H module (Figure 2) in slot 1 is mainly responsible for executing motion commands and storing the Macro tables in its non-volatile memory. The motion control chip and the non-volatile memory for storing Macro tables are part of the i-8094H module. Furthermore it has a DPRAM (volatile FIFO buffer) which stores up to 29 commands for further processing.

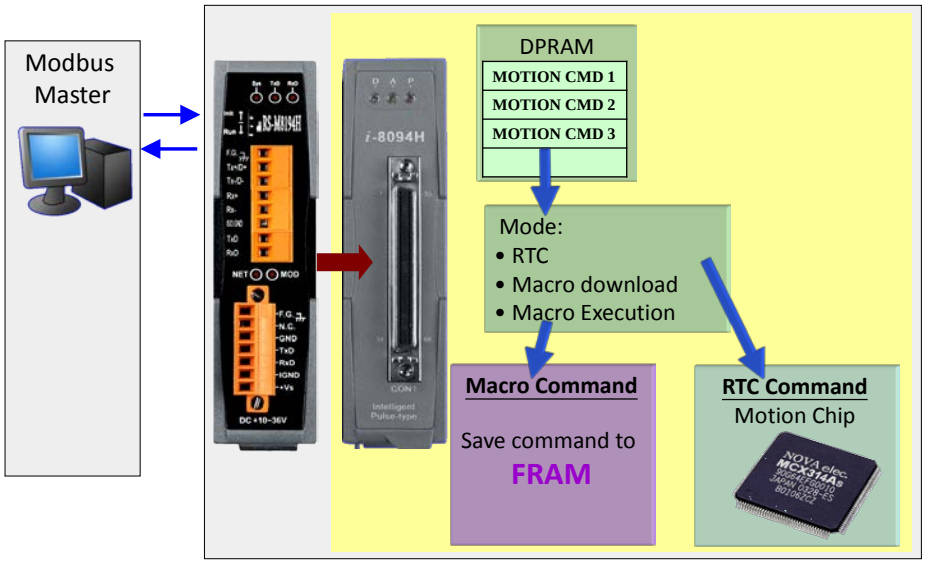


Figure 2: i-8094H situated in slot 1 of the motion device

4.16.2 Get RS-M8194H Status

```
eRTU RSM_GET_DEVICE_STATE (HANDLE hRsm, BYTE SlaveAddr,
        RsmState *pState)

typedef struct
{
     WORD Rsm8194hState;
     WORD I8094hState;
     WORD IllegalFuncErr;
}RsmState;
```

Description:

This function returns the current state of the RS-M8194H and of the i8094H module and furthermore the cause of the last Modbus exception code.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
pState:	***Rsm8194hState* (Table 8)** **RSM_READY:** **Only RTC commands can be sent.** **RSM_MP_DOWNLOAD:** **Only MP Macro commands are accepted.** **RSM_ISR_DOWNLOAD:** **Only ISR Macro commands are valid.** ***I8094hState* (Table 9)** **i8094H_READY:** **The i-8094H is ready for receiving RTC commands.** **i8094H_INITIALIZATION:** **Executing initialization table (IT commands).** **i8094H_MACRO_DOWNLOAD:** **i-8094H is in the MACRO download mode.** **i8094H_MP_EXECUTION:** **MP Macro table is being executed.**

	i8094H_ISR_EXECUTION: **Interrupt MACRO (ISR) program is being executed.** **i8094H_ERR_FIRMWARE:** **Module exited the DPRAM scan engine, Module needs to be rebooted.** **i8094H_FIRMWARE_BOOTING:** **i-8094H firmware has not completed the booting process.** ***IllegalFuncErr* (Table 10)** **ERR_NO_ERROR:** **No error occurred.** **ERR_FC_NOT_SUPPORTED:** **Function code not supported.** **ERR_EXCEED_MP_MEMORY:** **Macro program exceeds the number of command lines reserved for the selected table.** **ERR_MP_INSIDE_ISR:** **A MP command is being used inside ISR Macro.** **ERR_ISR_INSIDE_MP:** **An ISR command is being used inside MP Macro.** **ERR_RTC_INSIDE_MP_OR_ISR:** **A RTC command is being used inside a MP or ISR Macro.** **ERR_MP_OR_ISR_AS_RTC:** **A MP or ISR Macro command is being used outside a Macro program.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
RsmState rsmState;
RSM_GET_DEVICE_STATE (hRsm, 1, & rsmState);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01 (Slave Address)	04	00 64 (100)	00 03

MODBUS response:

UID (hex)	FC (hex)	Byte Count (hex)	Registers (hex)
01	04	0A	See next table

Register[]	Value (hex)	Remarks
0	00 01	*Rsm8194hState:* RSM_MP_DOWNLOAD
1	00 03	*I8094hState :* i8094H_MACRO_DOWNLOAD
2	00 06	*IllegalFuncErr:* ERR_RTC_INSIDE_MP_OR_ISR

Table 8: RS-M8194H states

States of the RS-M8194H module	Number (hex)	Description
RSM_READY	0x00	only RTC commands can be sent
RSM_MP_DOWNLOAD	0x01	only MP Macro commands are accepted
RSM_ISR_DOWNLOAD	0x02	only ISR Macro commands are valid

Table 9: i-8094H states

Different states of the i8094H	Number (hex)	Description
i8094H_READY	0x00	The i-8094H is ready for receiving RTC commands
i8094H_INITIALIZATION	0x01	executing initialization table (IT commands)
i8094H_MACRO_DOWNLOAD	0x03	i-8094H is in the MACRO download mode
i8094H_MP_EXECUTION	0x05	MP Macro table is being executed
i8094H_ISR_EXECUTION	0x07	Interrupt MACRO (ISR) program is being executed
i8094H_ERR_FIRMWARE	0x10	i-8094H firmware experienced an error and has to be power off/on
i8094H_FIRMWARE_BOOTING	0xFF	i-8094H firmware has not completed the booting process

Table 10: Causes of Modbus exception response

Illegal function errors	Number (hex)	Description
ERR_NO_ERROR	0x00	No error occurred
ERR_FC_NOT_SUPPORTED	0x01	Function code not supported
ERR_EXCEED_MP_MEMORY	0x03	Macro program exceeds the number of command lines reserved for the selected table
ERR_MP_INSIDE_ISR	0x04	A MP command is being used inside ISR macro
ERR_ISR_INSIDE_MP	0x05	A ISR command is being used inside MP macro
ERR_RTC_INSIDE_MP_OR_ISR	0x06	A RTC command is being used inside a MP or ISR macro
ERR_MP_OR_ISR_AS_RTC	0x07	A MP or ISR macro command is being used outside a macro program

5 FRnet Functions

5.1 Get FRnet DIO Signals

5.1.1 Read FRnet Group Data (MP)

※Δ eRTU RSM_MACRO_FRNET_IN (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *wGroup*)

Description:

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

Category:

MODBUS sub_function; MP, ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***wGroup*:**	**DI group number is between 8~15. If you enter a value between 0 to 7, you can get the output state of the external DO group.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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

- **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 RSM _MACRO_SET_RVAR() followed to get the return DI value. Please refer to MP related explanation literature.

RSM_MACRO_FRNET_IN (hRsm, 1, 8);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C 32	*Sub_function code*			
1	00 08	*Group number (8~15)*			

5.1.2 Read FRnet Channel Data (MP)

※Δ eRTU RSM_MACRO_FRNET_READ (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bGroup*, DWORD *dwChannel*)

Description:

Read the channel data of FRnet group.

Category:

MODBUS sub_function; MP, ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
wGroup:	DI group number is between 8~15. If you enter a value between 0 to 7, you can get the output state of the external DO group.
dwChannel:	The channel inside the group define the range of 0 to 15.

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_FRNET_READ (hRsm, 1, 8, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0C 34	*Sub_function code*			
1	00 00	*MSW of 0x0008*			
2	00 08	*LSW of 0x0008*			
3	00 00	*MSW of 0x0001*			
4	00 01	*LSW of 0x0001*			

5.1.3 Read FRnet Channel Data (RTC)

eRTU RSM_FRNET_READ_SINGLE_DIO (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD *wGroup*, BYTE *bChannelNo*, BYTE **pbChannelStatus*)

Description:

Read the channel data of FRnet group.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***wGroup*:**	**DI group number is between 8~15. If you enter a value between 0 to 7, you can get the output state of the external DO group.**
***bChannelNo*:**	**The channel number inside the group defines the range of 0 to 15.**
***pbChannelStatus*:**	**The pointer to the memory that stores the data which you read from the channel.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

5.1.4 Read FRnet Group Data (RTC)

eRTU RSM_FRNET_READ_GROUP_DIO (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD *wGroup*, WORD* *pwGroupStatus*)

Description:

Read the data of FRnet group.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***wGroup*:**	**DI group number is between 8~15. If you enter a value between 0 to 7, you can get the output state of the external DO group.**
pwGroupStatus	**The pointer to the memory that stores the data which you read from the group.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
WORD pwGroupStatus;
RSM_MACRO_FRNET_READ(hRsm, 1, 8, &pwGroupStatus);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 08	00 01

MODBUS response:

UID (hex)	FC (hex)	Byte Count(hex)	*pwGroupStatus* (hex)
01	04	02	0001

5.1.5 Read FRnet Multi-group Data (RTC)

eRTU RSM_FRNET_READ_MULTI_GROUP_DIO (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD *wStartGroup*, BYTE *bGroupQty*, WORD* *pwInputBuffer*, BYTE *bBufferSize*)

Description:

Read the data of FRnet multi-group.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
wStartGroup:	**The starting group.**
***bGroupQty*:**	**The group quantity.**
***pwInputBuffer*:**	**The pointer to the memory that store the data which you read from the group.**
***bBufferSize*:**	**The size of the registers（bBufferSize*2 must >= bGroupQty)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
WORD pwInputBuffer[2];
RSM_FRNET_READ_MULTI_GROUP_DIO(hRsm, 1, 8, 2,
&pwInputBuffer,  4);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 08	00 02

MODBUS response:

UID (hex)	FC (hex)	Byte Count (hex)	*pwInputBuffer[0]* (hex)	*pwInputBuffer[1]* (hex)
01	04	04	00 01	00 03

5.2 Set FRnet DO

5.2.1 Set FRnet DO Group Data (MP)

※Δ eRTU RSM_MACRO_FRNET_OUT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bGroup*, DWORD *data*)

Description:

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

Category:

MODBUS table, MODBUS sub_function; MP, ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***bGroup*:**	**Group number is between 0~7. Eight or above is not defined.**
data:	**0x00000000 ~ 0x0000ffff, can also use VARn to set.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

- **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.**

```
RSM_MACRO_FRNET_OUT (hRsm, 1, 0, 0x0000FFFF);
RSM_MACRO_FRNET_OUT (hRsm, 1, 1, 0x0000AAAA);
RSM_MACRO_FRNET_OUT (hRsm, 1, 2, 0x00001100);
RSM_MACRO_FRNET_OUT (hRsm, 1, 3, 0x00000011);
```

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	00 00	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	FF FF	*DO setting (for address = 0)*			
1	AA AA	*DO setting (for address = 1)*			
2	11 00	*DO setting (for address = 2)*			
3	00 11	*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.

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

The MODBUS command is listed as follows:

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

```
RSM_MACRO_FRNET_OUT (hRsm, 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 start address of VAR1 is 0x012E (= 300 + 2*n).

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0C 33	*Sub_function code*			
1	00 80	*bVAR0*			
2	00 00	*MSW of VAR1*			
3	01 2E	*LSW of VAR1*			

5.2.2 Set FRnet DO Channel Data (MP)

※Δ eRTU RSM_MACRO_FRNET_WRITE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bDoGroup*, DWORD *dwChannel*, DWORD *dwOutput*)

Description:

Write data to the channel of FRnet DO group.

Category:

MODBUS sub_function; MP, ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
bDoGroup:	Group number is between 0~7. Eight or above is not defined.
dwChannel:	The channel of the group is defined 0~15.
dwOutput:	The written data.

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_FRNET_WRITE(hRsm, 1, 1, 1, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 07	0E
Register[]	**Value (hex)**	**Remarks**			
0	0C 35	*Sub_function code*			
1	00 00	*MSW of 0x0001*			
2	00 01	*LSW of 0x0001*			
3	00 00	*MSW of 0x0001*			
4	00 01	*LSW of 0x0001*			
5	00 00	*MSW of 0x0001*			
6	00 01	*LSW of 0x0001*			

5.2.3 Set FRnet DO Channel Data (RTC)

eRTU RSM_FRNET_WRITE_SINGLE_DO (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bGroup*, DWORD *dwChannelNo*, BYTE *bOutput*)

Description:

Write data to the channel of FRnet DO group.

Category:

MODBUS table；RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***bGroup*:**	**Group number is between 0~7. Eight or above is not defined.**
dwChannelNo:	**The channel of the group is defined 0~15.**
***bOutput*:**	**The written data.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_FRNET_WRITE_SINGLE_DO (hRsm, 1, *bGroup*, *dwChannelNo*, 1);

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Value (hex)
01	05	(bGroup * 16 + dwChannelNo)	00 00(off); FF 00(on)

5.2.4 Set FRnet DO Group Data (RTC)

eRTU RSM_FRNET_WRITE_GROUP_DO (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bGroup*, DWORD *dwdata*)

Description:

Write data to FRnet DO group.

Category:

MODBUS table；RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***bGroup*:**	**Group number is between 0~7. Eight or above is not defined.**
***dwdata*:**	**0x00000000 ~ 0x0000ffff, can also use VARn to set.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_FRNET_WRITE_GROUP_DO(hRsm, 1, 0, 0x0000FFFF); // on module 1, set group number RA=0, output data is 0x0000ffff

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	00 00	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	00 00	***MSW of 0xFFFF***			
1	FF FF	***LSW of 0xFFFF***			

5.2.5 Set FRnet Multi-group DO Data (RTC)

eRTU RSM_FRNET_WRITE_MULTI_GROUP_DO (HANDLE *hRsm*, BYTE *SlaveAddr*, WORD *wStartGroup*, BYTE *bGroupQty*, WORD* *pwOutBuffer*, BYTE *bBufferSize*)

Description:

Write data to FRnet DO multi-group.

Category:

MODBUS table；RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
wStartGroup:	**The starting group.**
***bGroupQty*:**	**The group quantity.**
***pwOutBuffer*:**	**The pointer points to the memory that store the data of group which you want to write.**
***bBufferSize*:**	**The size of the registers（bBufferSize*2 >= bGroupQty)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
WORD pwOutBuffer[4];
pwOutBuffer[0]=0xFFFF;
pwOutBuffer[1]=0xAAAA;
pwOutBuffer[2]=0x1100;
pwOutBuffer[3]=0x0011;
RSM_FRNET_WRITE_MULTI_GROUP_DO(hRsm, 1, 0, 4, &
pwOutBuffer, 8);
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	00 00	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	FF FF	*DO setting (for address = 0)*			
1	AA AA	*DO setting (for address = 1)*			
2	11 00	*DO setting (for address = 2)*			
3	00 11	*DO setting (for address = 3)*			

5.3 FRnet Wait

※Δ eRTU RSM_MACRO_FRNET_WAIT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bDiGroup*, DWORD *dwChannel*, DWORD *dwDiInput*, DWORD *dwTimeout*)

Description:

Wait until FRnet DI turned ON.

Category:

MODBUS sub_function; MP, ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
bDiGroup:	Group number, 8~15
dwChannel:	The channel of the group is defined 0~15.
dwDiInput:	1:ON, 0:OFF
dwTimeout:	The waiting time(ms) 0: Represent the waiting time is infinite until ON. n: Represent only wait n(ms).

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_FRNET_WAIT(hRsm, 1, 8, 1, 1, 1000);

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01	10	1F 40	00 08	10

Register[]	Value (hex)	Remarks
0	0C 36	*Sub_function code*
1	00 08	*bDiGroup*
2	00 00	*MSW of 0x0001*
3	00 01	*LSW of 0x0001*
4	00 00	*MSW of 0x0001*
5	00 01	*LSW of 0x0001*
6	00 00	*MSW of 0x03E8*
7	03 E8	*LSW of 0x03E8*

5.4 FRnet DI Trigger Event Setting

eRTU RSM_SET_FRNET_TRIGGER_EVENT(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *bEnableDisable*)

Description:

This function enables the FRnet DI trigger event setting. Once this function has been called the FRnet DI trigger table set by the EzMove utility will be activated (see chapter 10)

Category:

RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***bEnableDisable*:**	**0-disable; 1-enabled**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_SET_FRNET_TRIGGER_EVENT (hRsm, 1, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	00 09	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	00 01	0-disable; 1-enabled			

5.5 Get FRnet DI Trigger Event Setting

eRTU RSM_GET_FRNET_TRIGGER_EVENT_SETTING(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE **pbEnableDisable*)

Description:

This function reads the current the FRnet DI trigger event setting. (See chapter 10)

Category:

RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***pbEnableDisable*:**	**0-disable; 1-enabled**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

BYTE bSetting;

RSM_GET_FRNET_TRIGGER_EVENT_SETTING (hRsm, 1, & bSetting);

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	03	00 09	00 01

6 Auto Homing Search

i-8094H module provides automatic home search function. After the configuration of proper settigs, the whole process for Homing searching can function automatically. The main steps are as bellows:

- **Near-home sensor searching (NHOME) under high-speed motion.**
- **Home sensor serachig 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 by adjusting settings accordingly to meet customer's actual needs. This operation could be performed automatically, therefore economize on CPU resource and reduce programming efforts.

6.1 Set Home Search Speed

eRTU RSM_SET_HV (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※ eRTU RSM_MACRO_SET_HV (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

This function sets the homing search speed.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**Homing speed (Vmin~Vmax PPS).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_HV is 0A 3C.
The Sub_function code of RSM_MACRO_SET_HV is 0C 3C.

MODBUS example:

```
RSM_SET_HV (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 3C/0C 3C	***Sub_function code***			
1	00 01	***axis***			
2	00 00	***MSW of data***			
3	01 F4	***LSW of data (500 = 0x1F4)***			

6.2 Limit Switch as the Home Sensor Setting

eRTU RSM_HOME_LIMIT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nType*)

※ eRTU RSM_MACRO_HOME_LIMIT (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
axis:	Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)
nType:	0: Disable. 1: Use the limit switch as the HOME sensor.

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_HOME_LIMIT is 0A 3D.
The Sub_function code of RSM_MACRO_HOME_LIMIT is 0C 3D.

MODBUS example:

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

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 3D/0C 3D	*Sub_function code*			
1	00 01	*axis*			
2	00 00	*nType*			

6.3 Home Search Mode Setting

eRTU RSM_SET_HOME_MODE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nStep1*, BYTE *nStep2*, BYTE *nStep3*, BYTE *nStep4* , DWORD *data*)

※ eRTU RSM_MACRO_SET_HOME_MODE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nStep1*, BYTE *nStep2*, BYTE *nStep3*, BYTE *nStep4* , DWORD *data*)

Description:

This function sets the automatic home search method and other related parameters.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***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: Positive correction. 2: Negative correction.**
***data*:**	**Offset value (0 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

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	

Remark:

The Sub_function code of RSM_SET_HOME_MODE is 0A 3E.
The Sub_function code of RSM_MACRO_SET_HOME_MODE is 0C 3E.

MODBUS example:

RSM_SET_HOME_MODE (hRsm, 1, 0x1, 2, 2, 1, 1, 3500);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 08	10
Register[]	**Value (hex)**	**Remarks**			
0	0A 3E/0C 3E	*Sub_function 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)*			

6.4 Start Automatic Home Search Execution

eRTU RSM_HOME_START (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_HOME_START (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

This function starts the automatic home search execution of the assigned axes.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

Remark:

The Sub_function code of RSM_HOME_START is 0A 3F.
The Sub_function code of RSM_MACRO_HOME_START is 0C 3F.

MODBUS example:

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

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 3F/0C 3F	*Sub_function code*			
1	00 01	*axis (AXIS_X)*			

7 Motion Control Commands

7.1 Single Axis Motion Control

- **Each axis is moving independently.**
- **The motion of each axis can be started independently.**
- **Multiple axes can move at the same time.**
- **Each individual axis can be controlled seperatly. The number of output pulses, speed, acceleration, etc. can be set for each axis:**
- **Each axis can receive commands to stop slowly or suddenly to meet the specific requirements.**
- **Independent axis motion can work with interpolation or synchrous action to perform more complicated and versatile motion.**

7.1.1 Acceleration/Deceleration Mode Setting

eRTU RSM_NORMAL_SPEED (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis* , BYTE *nMode*)

※Δ eRTU RSM_MACRO_NORMAL_SPEED (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis* , BYTE *nMode*)

Description:

This function sets the speed mode.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***nMode*:**	**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).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

Please refer the configurations of speed-related parameters.
The Sub_function code of RSM_NORMAL_SPEED is 0A 46.
The Sub_function code of RSM_MACRO_NORMAL_SPEED is 0C 46.

MODBUS example:

```
RSM_NORMAL_SPEED (hRsm, 1, AXIS_XYZU, 0);
//use a symmetric T-curve for all axes on module 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 46/0C 46	*Sub_function code*			
1	00 0F	*axis (AXIS_XYZU)*			
2	00 00	*nMode*			

Related example:

```
BYTE SlaveAddr=1; //select module 1.
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 20000);
//set the max speed of XYZU axes to be 20K PPS.

//=======================================================
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU, 0);
//use a symmetric T- curve  for all axes on module 1.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU, 1000);
//set the acceleration of all axes on module 1 to be 1000 PPS/Sec.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the start speed of all axes on module 1 to be 2000 PPS.
RSM_SET_AO(hRsm, SlaveAddr, AXIS_XYZU, 9);
//set the number of remaining offset pulses for all axes to be 9 pulses.
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_XYZU, 10000);
//move all axes on module 1 to be 10000 pulses.

//=======================================================
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU,1);
//use a symmetric S- curve for all axes on module 1.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
RSM_SET_K(hRsm, SlaveAddr, AXIS_XYZU, 500);
//set the acceleration rate of all axes on module 1 to be 500 PPS/sec^2.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 200);
//set the start speed of all axes on module 1 to be 200 PPS.
RSM_SET_AO(hRsm, SlaveAddr, AXIS_XYZU, 9);
//set the number of remaining offset pulses to be 9 pulses for all axes.
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_XYZU, -10000);
//move all axes on module 1 to be 10000 pulses in reverse direction.

//=======================================================
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU,2);
```

```
//use an asymmetric T-curve for all axes on module 1.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU, 1000 );
//set the acceleration of all axes on module 1 to be 1000 PPS/sec.
RSM_SET_D(hRsm, SlaveAddr, AXIS_XYZU, 500);
//set the deceleration of all axes on module 1 to be 500 PPS.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 200);
//set the start speed of all axes on module 1 to 200 PPS.
RSM_SET_AO(hRsm, SlaveAddr, AXIS_XYZU, 9);
//set the number of remaining offset pulses to be 9 pulses for all axes.
RSM_FIXED_MOVE(hRsm, SlaveAddr, axis, 10000);
//move all axes on module 1 to be 10000 pulses.

//=======================================================
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU,3);
//use an asymmetric S-curve for all axes on module 1.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
RSM_SET_K(hRsm, SlaveAddr, AXIS_XYZU, 500);
//set the acceleration rate of all axes on module 1 to be 500 PPS/sec^2.
RSM_SET_L(hRsm, SlaveAddr, AXIS_XYZU, 300);
//set the deceleration rate of all axes on module 1 to be 300 PPS/sec^2.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 200);
//set the start speed of all axes on module 1 to be 200 PPS.
RSM_SET_AO(hRsm, SlaveAddr, AXIS_XYZU, 9);
//set the number of remaining offset pulses to be 9 pulses for all axes.
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_XYZU, 10000);
//move all axes on module 1 to be 10000 pulses.
```

7.1.2 Initial Speed Setting

eRTU RSM_SET_SV (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_SV (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

This function sets the initial speed for the assigned axes.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**Initial speed (Please refer to description of RSM_SET_MAX_V for maximm speed) PPS.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_SV is 0A 47.
The Sub_function code of RSM_MACRO_SET_SV is 0C 47.

MODBUS example:

```
RSM_SET_SV (hRsm, 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:

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

7.1.3 Drive Speed Setting

eRTU RSM_SET_V (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_V (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

This function sets the desired drive speed for the assigned axes.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**Drive speed (Please refer to description of RSM_SET_MAX_V for maximm speed) PPS.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_V is 0A 48.
The Sub_function code of RSM_MACRO_SET_V is 0C 48.

MODBUS example:

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

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 48/0C 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)***			

7.1.4 Acceleration Setting

eRTU RSM_SET_A (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD data)

※Δ eRTU RSM_MACRO_SET_A (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

This function sets the acceleration value for the assigned axes.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**The acceleration value (PPS/sec). This value is related to the maximum speed value defined by RSM_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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_A is 0A 49.
The Sub_function code of RSM_MACRO_SET_A is 0C 49.

MODBUS example:

```
RSM_SET_A (hRsm, 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:

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

7.1.5 Deceleration Setting

eRTU RSM_SET_D (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_D (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

This function sets the deceleration value for the assigned axes.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**The deceleration value (PPS/sec). This value is related to the maximum speed value defined by RSM_SET_MAX_V () function. The maximum available deceleration value is MAX_V * 125. The minimum deceleration value is MAX_V ÷ 64, and all other deceleration 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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_D is 0A 4A.
The Sub_function code of RSM_MACRO_SET_D is 0C 4A.

MODBUS example:

```
RSM_SET_D (hRsm, 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:

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

7.1.6 Acceleration Rate Setting

eRTU RSM_SET_K (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_K (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**The acceleration rate (jerk) value (PPS/ sec^2). This value is related to the maximum speed value defined by RSM_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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_K is 0A 4B.
The Sub_function code of RSM_MACRO_SET_K is 0C 4B.

MODBUS example:

```
RSM_SET_K (hRsm, 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:

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

7.1.7 Deceleration Rate Setting

eRTU RSM_SET_L (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_L (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**The deceleration rate (jerk) value (PPS/ Sec^2). This value is related to the maximum speed value defined by RSM_SET_MAX_V () function. The minimum deceleration rate is MAX_V * 0.0119211; The maximum deceleration rate value is 2,000,000,000**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_L is 0A 4C.
The Sub_function code of RSM_MACRO_SET_L is 0C 4C.

MODBUS example:

```
RSM_SET_L (hRsm, 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:

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

7.1.8 Offset Setting

eRTU RSM_SET_AO (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

※Δ eRTU RSM_MACRO_SET_AO (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

Description:

This function sets for the assigned axes the offset for deceleration. The default value for offset is 8 after power-on. The offset function can be used for compensation the pulses when the deceleration speed does not reach the initial speed during S-curve fixed pulse driving.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**The number of remaining offset pulses. (-32,768 ~ +32,767).**

Return:

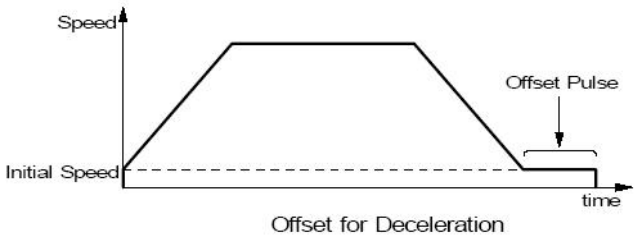
0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_AO is 0A 4D.
The Sub_function code of RSM_MACRO_SET_AO is 0C 4D.

MODBUS example:

```
RSM_SET_AO (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 4D/0C 4D	*Sub_function code*			
1	00 01	*axis (1 = AXIS_X)*			
2	00 00	*MSW of data*			
3	00 C8	*LSW of data (200 = 0xC8)*			

7.1.9 Relative Distance Motion

eRTU RSM_FIXED_MOVE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

※Δ eRTU RSM_MACRO_FIXED_MOVE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

Description:

Command a point-to-point motion for fixed position movement.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**Pulses (-2,000,000,000 ~ +2,000,000,000).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_FIXED_MOVE is 0A 4E.
The Sub_function code of RSM_MACRO_FIXED_MOVE is 0C 4E.

MODBUS example:

```
RSM_FIXED_MOVE (hRsm, SlaveAddr, AXIS_XYZU, 10000);
// AXIS_XYZU move 10000 Pulses
```

The MODBUS command is listed as follows:

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

Related example:

```
BYTE SlaveAddr=1; //select module 1
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 20000);
//set the max. velocity of all axes on module 1 to be 20K PPS
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU, 0);
//set the speed profile of all axes on module 1 to be symmetric T-curve
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU,1000);
//set the acceleration value of all axes on module 1 to be 1000 PPS/S
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the start velocity of all axes on module 1 to be 2000 PPS
RSM_SET_AO(hRsm, SlaveAddr, AXIS_XYZU, 9);
//set the remaining offset pulses to be 9 PPS
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_XYZU, 10000);
// move 10000 Pulses for each axis on module 1
```

7.1.10 Change Target Position on the Fly

eRTU RSM_SET_PULSE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_PULSE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

This command supports the change of target position on the fly for each axis (but does not support change of the direction during driving).

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**Pulses (0 ~ +2,000,000,000).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_PULSE is 0A 4F.
The Sub_function code of RSM_MACRO_SET_PULSE is 0C 4F.

MODBUS example:

```
RSM_SET_PULSE (hRsm, SlaveAddr, AXIS_XYZU, 9000);
//Change the total pulses to 9000 pulses during fixed pulse moving.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 4F/0C 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)*			

Related example:

```
BYTE SlaveAddr=1; //select module 1
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 20000);
//set the max velocity of all axes on module 1 to be 20K PPS
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU, 0);
//set the speed profile of all axes on module 1 to be symmetric T-curve
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU,1000);
//set the acceleration value of all axes on module 1 to be 1000 PPS/S
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the start velocity of all axes on module 1 to be 2000 PPS
RSM_SET_AO(hRsm, SlaveAddr, AXIS_XYZU, 9);
//set the remaining offset pulses to be 9 PPS
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_XYZU, 10000);
// move 10000 Pulses for each axis on module 1
RSM_SET_PULSE(hRsm, SlaveAddr, AXIS_XYZU, 9000);
//Set pulse as 9000 Pulse.
```

7.1.11 Move to Absoulte Position

eRTU RSM_ABS_FIXED_MOVE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

※Δ eRTU RSM_ABS_MACRO_FIXED_MOVE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

Description:

This function commands a controlled point-to-point motion to a specified absolute position.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***data*:**	**Absolute position (-2,000,000,000 ~ +2,000,000,000).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_ABS_FIXED_MOVE is 0x0AF6.
The Sub_function code of RSM_ABS_MACRO_FIXED_MOVE is 0x0CF6

MODBUS example:

```
RSM_ABS_FIXED_MOVE (hRsm, SlaveAddr, AXIS_XYZU, 10000);
// AXIS_XYZU moves from the current position to the logic position 10000 in a straight line
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A F6/0C F6	*Sub_function code*			
1	00 0F	*axis (0xF = AXIS_XYZU)*			
2	00 00	*MSW of data*			
3	27 10	*LSW of data (10000 = 0x2710)*			

Related example:

```
BYTE SlaveAddr=1; //select module 1
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 20000);
//set the max. velocity of all axes on module 1 to be 20K PPS
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU, 0);
//set the speed profile of all axes on module 1 to be symmetric T-curve
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU,1000);
//set the acceleration value of all axes on module 1 to be 1000 PPS/S
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the start velocity of all axes on module 1 to be 2000 PPS
RSM_SET_AO(hRsm, SlaveAddr, AXIS_XYZU, 9);
//set the remaining offset pulses to be 9 PPS
RSM_ABS_FIXED_MOVE(hRsm, SlaveAddr, AXIS_XYZU, 10000);
// All four axis are moving to the absolute position 10000
```

7.1.12 Continue Pulse Driving Output

eRTU RSM_CONTINUE_MOVE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

※Δ eRTU RSM_MACRO_CONTINUE_MOVE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

Description:

When the Continuous Pulse Driving is performed the RS-M8194H will drive pulse output at a specified speed until a stop command or external stop signal is activated. The drive speed can be changed freely during continuous pulse driving.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
***SlaveAddr*:**	Modbus RTU Slave Address (valid range: 1 to 247)
***axis*:**	Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)
***data*:**	The specified speed (positive value for CW motion; negative value for CCW motion)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_CONTINUE_MOVE is 0A 50.
The Sub_function code of RSM_MACRO_CONTINUE_MOVE is 0C 50.

MODBUS example:

```
RSM_CONTINUE_MOVE (hRsm, SlaveAddr, AXIS_XYZU, 1000);
//continue to move at the speed of 1K PPS
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 50/0C 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)*			

Related example:

```
BYTE SlaveAddr=1; //select module 1
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 20000);
//set the maximum speed of all axes on module 1 to 20K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU, 0);
//set the speed profile for all axes as a symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to 2000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU,1000);
//set the acceleration value of all axes to 1000 PPS/S.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 2000);
//set the start velocity of all axes to 2000 PPS
RSM_CONTINUE_MOVE(hRsm, SlaveAddr, AXIS_XYZU, 1000);
//move all axes on module 1 at a speed of 1000 PPS.
```

7.2 Interpolation Commands

7.2.1 Interpolation Axes Assignement

eRTU RSM_AXIS_ASSIGN (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*)

※Δ eRTU RSM_MACRO_AXIS_ASSIGN (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis1*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***axis2*:**	**The second axis; can be either X, Y, Z, or U (1 or 2 or 4 or 8).**
***axis3*:**	**The third axis; can be either X, Y, Z, or U (1 or 2 or 4 or 8). Without 3rd axis the value is 0.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_AXIS_ASSIGN is 0A 5A.
The Sub_function code of RSM_MACRO_AXIS_ASSIGN is 0C 5A.

MODBUS example:

```
RSM_AXIS_ASSIGN (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 5A/0C 5A	*Sub_function code*			
1	00 01	*axis1 (axis1 = AXIS_X)*			
2	00 02	*axis2 (axis1 = AXIS_Y)*			
3	00 00	*Axis3 (not defined)*			

7.2.2 Speed and Acc/Dec Mode Setting

eRTU RSM_VECTOR_SPEED (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nMode*)

※Δ eRTU RSM_MACRO_VECTOR_SPEED (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nMode*)

Description:

This function sets the acceleration and deceleration profile for the interpolation axis.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***nMode*:**	**0 → 2-axis linear or circular motion at a constant vector speed (Set VV and VSV; and VV=VSV).** **1 → 2-axis linear motion using a symmetric T-curve velocity profile (set VSV, VV, VA, and VAO)** **2 → 2-axis linear motion using a symmetric S-curve velocity profile(set VSV, VV, VK, and VAO)** **3 → 2-axis linear motion using an asymmetric T-curve velocity profile (set VSV, VV, VA, VD, and VAO)** **4 → 2-axis linear motion using an asymmetric S-curve velocity profile (set VSV, VV, VK, VL, and VAO)** **5 → 2-axis circular motion using a symmetric T-curve velocity profile (set VSV, VV, VA, and VAO)** **6 → 2-axis circular motion using an asymmetric T-curve velocity rofile (set VSV, VV, VA, VD, and VAO)** **7 → 3-axis linear motion at a constant vector speed set VV and VSV; and VV=VSV)** **8 → 3-axis linear motion at using a symmetric T-curve velocity rofile (set VSV, VV, VA, and VAO)** **9 → 3-axis linear motion using a symmetric S-curve velocity profile (set VSV, VV, VK, and VAO)** **10 → 3-axis linear motion using an asymmetric T-curve velocity profile (set VSV, VV, VA, VD, and VAO)** **11 → 3-axis linear motion using an asymmetric S-curve velocity profile (set VSV, VV, VK, VL, and VAO)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

Please refer the configurations of speed-related parameters.
The Sub_function code of RSM_VECTOR_SPEED is 0A 5B.
The Sub_function code of RSM_MACRO_VECTOR_SPEED is 0C 5B.

MODBUS example:

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

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 5B/0C 5B	*Sub_function code*			
1	00 00	*nMode*			

Related example:

```
BYTE SlaveAddr=1; //select module 1.
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 20000);
//set the maximum speed of all axes to 20K PPS.

//=============================================
RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
RSM_VECTOR_SPEED(hRsm, SlaveAddr, 0);
//set module 1 to perform 2-axis linear or circular motion
RSM_SET_VV(hRsm, SlaveAddr, 1000);
//set the vector speed to 1000 PPS.
RSM_LINE_2D(hRsm, SlaveAddr, 12000, 10000);
//execute the 2-axis linear interpolation motion.

//=============================================
RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
//set the X axis as the first axis and the Y axis as the second axis.
RSM_VECTOR_SPEED(hRsm, SlaveAddr, 1);
//set module 1 to perform 2-axis linear motion using a symmetric
//T-curve velocity profile (VSV、VV、VA、VAO).
RSM_SET_VSV(hRsm, SlaveAddr, 500); //set the starting vector speed to
500 PPS.
RSM_SET_VV(hRsm, SlaveAddr, 2000); //set the vector speed to 2000 PPS.
RSM_SET_VA(hRsm, SlaveAddr, 1000);
//set the vector acceleration to 1000 PPS/sec.
```

```
        RSM_LINE_2D(hRsm, SlaveAddr, 20000, 10000);
        //execute the 2-axis linear interpolation motion.

        //=============================================
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
        //set the X axis as the first axis and the Y axis as the second axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 2);
        //2-axis linear motion using a symmetric S-curve velocity profile(VSV、VV、
        //VK、AO).
        RSM_SET_VSV(hRsm, SlaveAddr, 200); //set the starting vector speed to
200 PPS.
        RSM_SET_VV(hRsm, SlaveAddr, 2000); //set the vector speed to 2000 PPS.
        RSM_SET_VK(hRsm, SlaveAddr, 500); //set the acceleration rate to 500
PPS/Sec.
        RSM_SET_VAO(hRsm, SlaveAddr, 20);
        //set the value of remaining offset pulses to 20.
        RSM_LINE_2D(hRsm, SlaveAddr, 10000, 10000);
        //execute the 2-axis linear interpolation motion.

        //=============================================
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
        //set the X axis as the first axis and the Y axis as the second axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 3);
        //2-axis linear motion using an asymmetric T-curve velocity profile(VSV、
        //VV、VA、VD、VAO).
        RSM_SET_VSV(hRsm, SlaveAddr, 100); //set the start vector speed to 100
PPS.
        RSM_SET_VV(hRsm, SlaveAddr, 2000); //set the vector speed to 2000 PPS.
        RSM_SET_VA(hRsm, SlaveAddr, 1000);
        //set the vector acceleration to 1000 PPS/sec.
        RSM_SET_VD(hRsm, SlaveAddr, 500); //set the vector deceleration to 500
PPS/sec.
        RSM_SET_VAO(hRsm, SlaveAddr, 20);
        //set the value of remaining offset pulses to 20.
        RSM_LINE_2D(hRsm, SlaveAddr, 10000, 5000);
        //execute the 2-axis linear interpolation motion.

        //=============================================
        long fp1=4000;
        long fp2=10000;
        unsigned short  sv=200;
        unsigned short  v=2000;
        RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 8000);
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
        //set the X axis as the first axis and the Y axis as the second axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 4);
```

```
        //2-axis linear motion using an asymmetric S-curve velocity profile(VSV、
        //VV、VK、VL、VAO).
        RSM_SET_VSV(hRsm, SlaveAddr, sv); //set the starting velocity to sv PPS.
        RSM_SET_VV(hRsm, SlaveAddr, v); //set the vector speed to v PPS.
        RSM_SET_VK(hRsm, SlaveAddr, 500); //set the acceleration rate to 500
PPS/Sec^2.
        RSM_SET_VL(hRsm, SlaveAddr, 300); //set the deceleration rate to 300
PPS/Sec^2.
        RSM_SET_VAO(hRsm, SlaveAddr, 20);
        //set the value of remaining offset pulses to 20.
        RSM_LINE_2D(hRsm, SlaveAddr, fp1, fp2); //execute the 2-axis linear
motion.
        //=============================================
        long fp1=11000;
        long fp2=9000;
        long c1=10000;
        long c2=0;
        unsigned short  sv=100;
        unsigned short  v=3000;
        unsigned long  a=5000;
        RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 8000);
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
        //set the X axis as the first axis and the Y axis as the second axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 5);
        //2-axis circular motion using a symmetric T-curve velocity profile(VSV、
        // VV、VA、VAO).
        RSM_SET_VSV(hRsm, SlaveAddr, sv); //set the starting vector speed to sv
PPS.
        RSM_SET_VV(hRsm, SlaveAddr, v); //set vector speed to v PPS.
        RSM_SET_VA(hRsm, SlaveAddr, a); //set the vector acceleration to a
PPS/sec.
        RSM_SET_VAO(hRsm, SlaveAddr, 0);
        //set the value of remaining offset pulses to 0 Pulse.
        RSM_ARC_CW(hRsm, SlaveAddr, c1,c2, fp1, fp2);
        //execute the 2-axis CW circular motion.
        //=============================================
        long c1=300;
        long c2=0;
        unsigned short sv=100;
        unsigned short v=3000;
        unsigned long a=125;
        unsigned long d=12;
        RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 8000);
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
        //set the X axis as the first axis and the Y axis as the second axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 6);
```

```
        //2-axis circular motion using an asymmetric T-curve velocity
        //profile(VSV、VV、VA、VD、VAO).
        RSM_SET_VSV(hRsm, SlaveAddr, sv); //set the starting vector speed to sv
PPS.
        RSM_SET_VV(hRsm, SlaveAddr, v); //set vector speed to v PPS.
        RSM_SET_VA(hRsm, SlaveAddr, a); //set acceleration to a PPS/Sec.
        RSM_SET_VD(hRsm, SlaveAddr, d); //set the deceleration to d PPS/Sec.
        RSM_SET_VAO(hRsm, SlaveAddr, 0); //set the value of remaining offset
pulses to 0.
        RSM_CIRCLE_CW(hRsm, SlaveAddr, c1, c2);
        //execute the 2-axis CW circular motion.

        //=============================================
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, AXIS_Z);
        // set axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 7);
        //3-axis linear motion at a constant vector speed (VSV=VV).
        RSM_SET_VSV(hRsm, SlaveAddr, 1000); //set the start speed to 1000 PPS.
        RSM_SET_VV(hRsm, SlaveAddr, 1000); //set the constant speed to 1000
PPS.
        RSM_LINE_3D(hRsm, SlaveAddr, 10000, 10000,10000);
        //execute the 3-axis linear motion.

        //=============================================
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, AXIS_Z);
        // set axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z-axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 8);
        //3-axis linear motion using a symmetric T-curve velocity profile(VSV、VV、
        //VA、VAO).
        RSM_SET_VSV(hRsm, SlaveAddr, 100); //set the starting speed to 100 PPS.
        RSM_SET_VV(hRsm, SlaveAddr, 3000); //set the vector speed to 3000 PPS.
        RSM_SET_VA(hRsm, SlaveAddr, 500); //set the vector acceleration to 500
PPS/Sec.
        RSM_SET_VAO(hRsm, SlaveAddr, 20);
        //set the value of remaining offset pulses to 20.
        RSM_LINE_3D(hRsm, SlaveAddr, 10000, 1000,20000);
        //execute the 3-axis linear motion
        //=============================================
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, AXIS_Z);
        // set the axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 9);
        //3-axis linear motion using a symmetric S-curve velocity profile(VSV、VV、
        //VK、VAO).
        RSM_SET_VSV(hRsm, SlaveAddr, 100); //set the starting speed to 100 PPS.
        RSM_SET_VV(hRsm, SlaveAddr, 3000); //set the vector speed to 3000 PPS.
        RSM_SET_VK(hRsm, SlaveAddr, 500);
```

```
        //set the vector acceleration rate to 500 PPS/sec^2.
        RSM_SET_VAO(hRsm, SlaveAddr, 20);
        //set the value of remaining offset pulses to 20.
        RSM_LINE_3D(hRsm, SlaveAddr, 10000, 1000,1000);
        //execute the 3-axis linear motion.

        //=============================================
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, AXIS_Z);
        // set the axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 10);
        //set the module 1 to perform 3-axis linear motion
        //using an asymmetric T-curve speed profile(VSV、VV、VA、VD、VAO).
        RSM_SET_VSV(hRsm,SlaveAddr, 100); //set the starting speed to 100 PPS.
        RSM_SET_VV(hRsm,SlaveAddr, 2000); //set the vector speed as 2000 PPS.
        RSM_SET_VA(hRsm, SlaveAddr, 1000);
        //set the vector acceleration to 1000 PPS/sec.
        RSM_SET_VD(hRsm, SlaveAddr, 500); //set the vector deceleration to 500
PPS/sec.
        RSM_SET_VAO(hRsm, SlaveAddr, 20);
        //set the value of remaining offset pulses to 20.
        RSM_LINE_3D(hRsm, SlaveAddr, 10000, 1000,1000);
        //execute the 3-axis linear motion.

        //=============================================
        long fp1=4000;
        long fp2=10000;
        long fp3=20000;
        unsigned short sv=200;
        unsigned short v=2000;
        RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 8000);
        RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, AXIS_Z);
        // set axis1 as the X axis, axis2 as the Y axis, and axis3 as the Z axis.
        RSM_VECTOR_SPEED(hRsm, SlaveAddr, 11);
        //3-axis linear motion using an asymmetric S-curve velocity profile(VSV、
        //VV、VK、VL、VAO).
        RSM_SET_VSV(hRsm, SlaveAddr, sv); //set the starting speed to sv PPS.
        RSM_SET_VV(hRsm, SlaveAddr, v); //set the vector speed to v PPS.
        RSM_SET_VK(hRsm, SlaveAddr, 500);
        //set the vector acceleration rate to 500 PPS/sec^2.
        RSM_SET_VL(hRsm, SlaveAddr, 300);
        //set the vector deceleration rate to 300 PPS/sec^2.
        RSM_SET_VAO(hRsm, SlaveAddr, 20);
        //set the value of remaining offset pulses to 20.
        RSM_LINE_3D(hRsm, SlaveAddr, fp1, fp2, fp3);
        //execute the 3-axis linear motion.
```

7.2.3 Vector Initial Speed Setting

eRTU RSM_SET_VSV (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_VSV (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

Description:

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

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***data*:**	**The vector starting speed value (in PPS) (For maximum value please refers to description of RSM_SET_MAX_V).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_VSV is 0A 5C.
The Sub_function code of RSM_MACRO_SET_VSV is 0C 5C.

MODBUS example:

```
RSM_SET_VSV (hRsm, 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:

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

7.2.4 Vector Speed Setting

eRTU RSM_SET_VV (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_VV (HANDLE *hRsm*, BYTE *SlaveAddr*, 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 RSM_AXIS_ASSIGN() function.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***data*:**	**The vector speed value (in PPS) (For maximum value please refers to description of RSM_SET_MAX_V).**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_VV is 0A 5D.
The Sub_function code of RSM_MACRO_SET_VV is 0C 5D.

MODBUS example:

RSM_SET_VV (hRsm, 1, 120000L);
//set the vector speed of the interpolation on module 1 to 120000 PPS.

The MODBUS command is listed as follows:

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

7.2.5 Vector Acceleration Setting

eRTU RSM_SET_VA (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_VA (HANDLE *hRsm*, BYTE *SlaveAddr*, 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 RSM_AXIS_ASSIGN() function.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***data*:**	**The vector acceleration value (in PPS/sec). This value is related to the maximum speed value defined by RSM_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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_VA is 0A 5E.
The Sub_function code of RSM_MACRO_SET_VA is 0C 5E.

MODBUS example:

```
RSM_SET_VA (hRsm, 1, 100000L);
//set the vector acceleration of the interpolation motion
//on module 1 to 100K PPS/sec.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 5E/0C 5E	*Sub_function code*			
1	00 01	*MSW of data*			
2	86 A0	*LSW of data (100000 = 0x186A0)*			

7.2.6 Vector Deceleration Setting

eRTU RSM_SET_VD (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_VD (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

Description:

This function sets the vector deceleration of the interpolation motion.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***data*:**	**The vector deceleration value (in PPS/sec). This value is related to the maximum speed value defined by RSM_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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_VD is 0A 5F.
The Sub_function code of RSM_MACRO_SET_VD is 0C 5F.

MODBUS example:

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

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 5F/0C 5F	*Sub_function code*			
1	00 01	*MSW of data*			
2	86 A0	*LSW of data (100000 = 0x186A0)*			

7.2.7 Vector Acceleration Rate Setting

eRTU RSM_SET_VK (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_VK (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

Description:

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

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***data*:**	**The acceleration rate (jerk) value (PPS/sec^2). This value is related to the maximum speed value defined by RSM_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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_VK is 0A 60.
The Sub_function code of RSM_MACRO_SET_VK is 0C 60.

MODBUS example:

```
RSM_SET_VK (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 60/0C 60	*Sub_function code*			
1	00 00	*MSW of data*			
2	27 10	*LSW of data (10000 = 0x2710)*			

7.2.8 Vector Deceleration Rate Setting

eRTU RSM_SET_VL (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

※Δ eRTU RSM_MACRO_SET_VL (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD *data*)

Description:

Set the deceleration rate of the interpolation motion.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***data*:**	**The deceleration rate (jerk) value (PPS/sec^2). This value is related to the maximum speed value defined by RSM_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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_VL is 0A 61.
The Sub_function code of RSM_MACRO_SET_VL is 0C 61.

MODBUS example:

```
RSM_SET_VL (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 61/0C 61	*Sub_function code*			
1	00 00	*MSW of data*			
2	27 10	*LSW of data (10000 = 0x2710)*			

7.2.9 Vector Offset Pulses Setting

eRTU RSM_SET_VAO (HANDLE *hRsm*, BYTE *SlaveAddr*, long *data*)

※Δ eRTU RSM_MACRO_SET_VAO (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***data*:**	**The number of remaining offset pulses. (-32,768 ~ +32,767).**

Return:

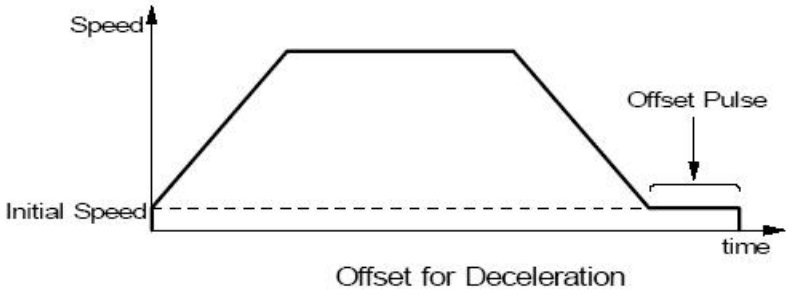
0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_VAO is 0A 62.
The Sub_function code of RSM_MACRO_SET_VAO is 0C 62.

MODBUS example:

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



The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 62/0C 62	***Sub_function code***			
1	00 00	***MSW of data***			
2	00 C8	***LSW of data (200 = 0xC8)***			

7.2.10 2-Axis Interpolation Relative Distance Motion

eRTU RSM_LINE_2D (HANDLE *hRsm*, BYTE *SlaveAddr*, long *fp1*, long *fp2*)

※Δ eRTU RSM_MACRO_LINE_2D (HANDLE *hRsm*, BYTE *SlaveAddr*, long *fp1*, long *fp2*)

Description:

This function executes a 2-axis linear interpolation motion.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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:

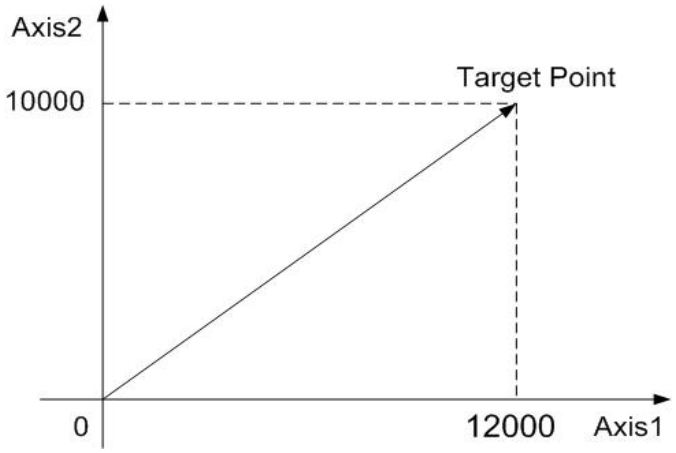
0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_LINE_2D is 0A 63.
The Sub_function code of RSM_MACRO_LINE_2D is 0C 63.

MODBUS example:

```
RSM_LINE_2D (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0A 63/0C 63	*Sub_function code*			
1	00 00	*MSW of fp1*			
2	2E E0	*LSW of fp1 (12000 = 0x2EE0)*			
3	00 00	*MSW of fp2*			
4	27 10	*LSW of fp2 (10000 = 0x2710)*			

7.2.11 2-Axis Interpolation Absoult Position Motion

eRTU RSM_ABS_LINE_2D (HANDLE *hRsm*, BYTE *SlaveAddr*, long *fp1*, long *fp2*)

※Δ eRTU RSM_ABS_MACRO_LINE_2D (HANDLE *hRsm*, BYTE *SlaveAddr*, long *fp1*, long *fp2*)

Description:

This function commands a controlled a 2-axis linear interpolation motion to a specified absolute position.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***fp1*:**	**Absolute position of axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**Absolute position of axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)**

Return:

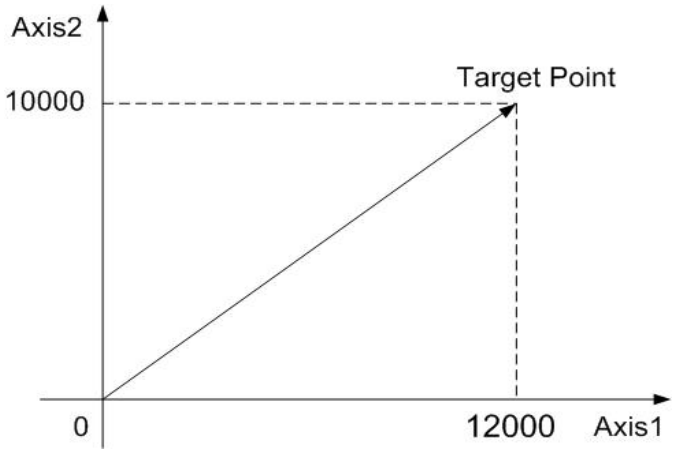
0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_ABS_LINE_2D is 0x0AF7.
The Sub_function code of RSM_ABS_MACRO_LINE_2D is 0x0CFX.

MODBUS example:

```
RSM_ABS_LINE_2D (hRsm, 1, 12000, 10000);
//Moves from the origin to the absolute position (12000, 10000)
```



2-axis linear interpolation motion

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0A FX/0C FX	*Sub_function code*			
1	00 00	*MSW of fp1*			
2	2E E0	*LSW of fp1 (12000 = 0x2EE0)*			
3	00 00	*MSW of fp2*			
4	27 10	*LSW of fp2 (10000 = 0x2710)*			

7.2.12 3-Axis Interpolation Relative Distance Motion

eRTU RSM_LINE_3D (HANDLE *hRsm*, BYTE *SlaveAddr*, long *fp1*, long *fp2*, long *fp3*)

※Δ eRTU RSM_MACRO_LINE_3D (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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:

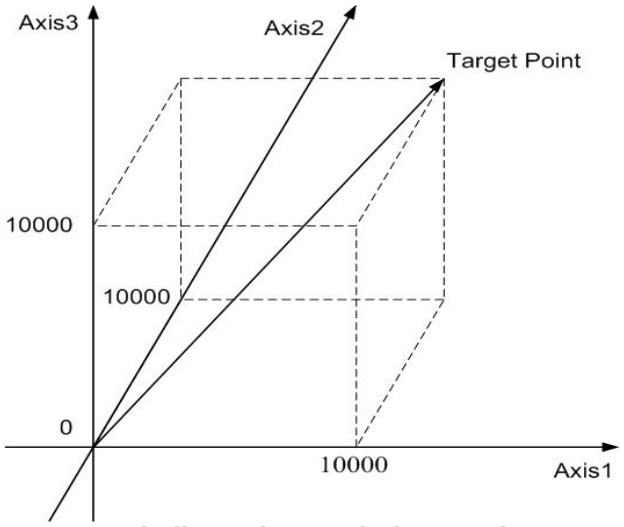
0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_LINE_3D is 0A 64.
The Sub_function code of RSM_MACRO_LINE_3D is 0C 64.

MODBUS example:

```
RSM_LINE_3D (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 07	0E
Register[]	**Value (hex)**	**Remarks**			
0	0A 64/0C 64	*Sub_function 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)*			

7.2.13 3-Axis Interpolation Absolute Position Motion

eRTU RSM_ABS_LINE_3D (HANDLE *hRsm*, BYTE *SlaveAddr*, long *fp1*, long *fp2*, long *fp3*)

※Δ eRTU RSM_ABS_MACRO_LINE_3D (HANDLE *hRsm*, BYTE *SlaveAddr*, long *fp1*, long *fp2*, long *fp3*)

Description:

This function executes a 3-axis linear interpolation motion to the specified absolute position.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***fp1*:**	**Absolute position of axis 1 in Pulses(-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**Absolute position of axis 2 in Pulses(-2,000,000,000 ~ +2,000,000,000)**
***fp3*:**	**Absolute position of axis 2 in Pulses(-2,000,000,000 ~ +2,000,000,000)**

Return:

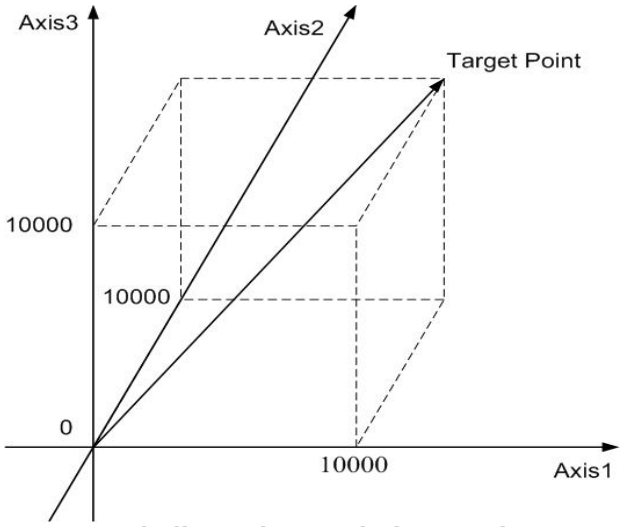
0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_LINE_3D is 0x0AF8.
The Sub_function code of RSM_MACRO_LINE_3D is 0x0CF8.

MODBUS example:

```
RSM_ABS_LINE_3D (hRsm, 1, 10000, 10000, 10000);
//Move to the specified target position.
```



3-axis linear interpolation motion

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 07	0E
Register[]	**Value (hex)**	**Remarks**			
0	0A F8/0C F8	*Sub_function 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)*			

7.2.14 2-Axis Circular Interpolation Relative Distance Motion (an CW Arc)

eRTU RSM_ARC_CW (HANDLE *hRsm*, BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

※Δ eRTU RSM_MACRO_ARC_CW (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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).**
***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:

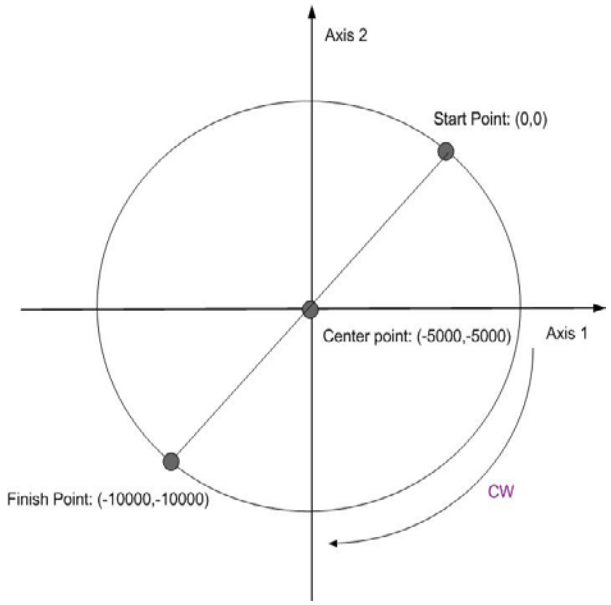
0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_ARC_CW is 0A 65.
The Sub_function code of RSM_MACRO_ARC_CW is 0C 65.

MODBUS example:

```
RSM_ARC_CW (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 09	12
Register[]	**Value (hex)**	**Remarks**			
0	0A 65/0C 65	*Sub_function 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)*			

7.2.15 2-Axis Circular Interpolation Relative Distance Motion (an CCW Arc)

eRTU RSM_ARC_CCW (HANDLE *hRsm*, BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

※Δ eRTU RSM_MACRO_ARC_CCW (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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).**
***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:

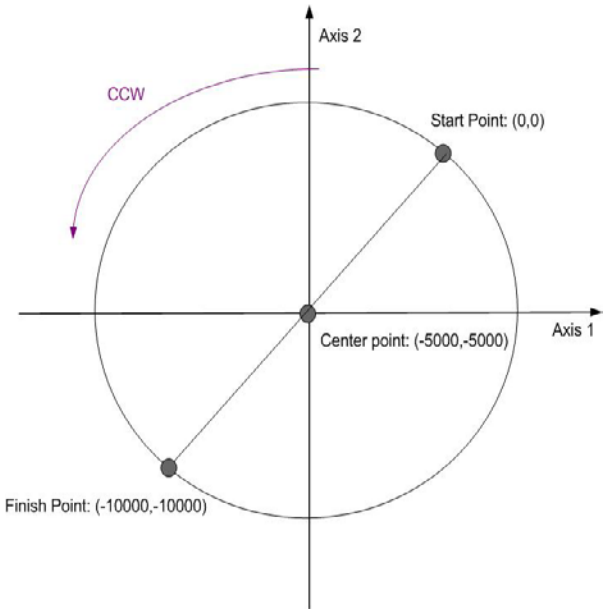
0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_ARC_CCW is 0A 67.
The Sub_function code of RSM_MACRO_ARC_CCW is 0C 67.

MODBUS example:

```
RSM_ARC_CCW (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 09	12
Register[]	**Value (hex)**	**Remarks**			
0	0A 67/0C 67	*Sub_function 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)*			

7.2.16 2-Axis Circular Interpolation Absoult Position Motion (an CW Arc)

eRTU RSM_ABS_ARC_CW (HANDLE *hRsm*, BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

※Δ eRTU RSM_ABS_MACRO_ARC_CW (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***cp1*:**	**The absolute position of the center to the current position of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).**
***cp2*:**	**The absolute position of the center to the current position of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).**
***fp1*:**	**The absolute position of the axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**The absolute position of the axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)**

Return:

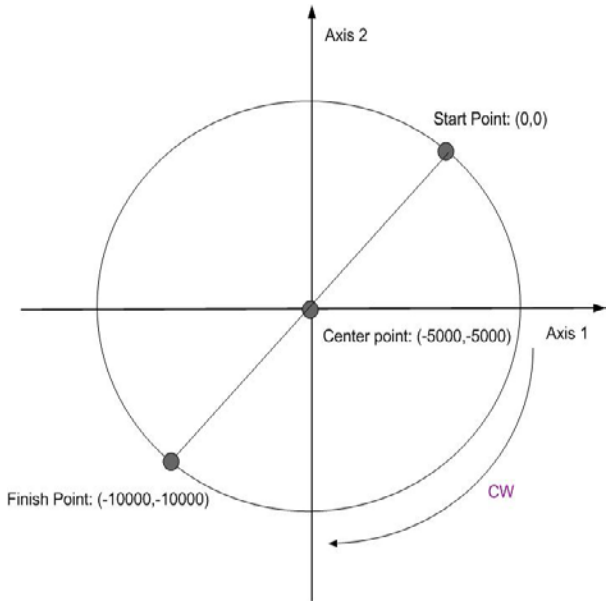
0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_ABS_ARC_CW is 0x0AF9.
The Sub_function code of RSM_ABS_MACRO_ARC_CW is 0x0CF9.

MODBUS example:

```
RSM_ABS_ARC_CW (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 09	12
Register[]	**Value (hex)**	**Remarks**			
0	0A F9/0C F9	*Sub_function 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)*			

7.2.17 2-Axis Circular Interpolation Absoult Position Motion (an CCW Arc)

eRTU RSM_ABS_ARC_CCW (HANDLE *hRsm*, BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

※Δ eRTU RSM_ABS _MACRO_ARC_CCW (HANDLE *hRsm*, BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

Description:

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

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***cp1*:**	**The absolute position of the center of axis 1 in pulses. (-2,000,000,000 ~ +2,000,000,000).**
***cp2*:**	**The absolute position of the center of axis 2 in pulses. (-2,000,000,000 ~ +2,000,000,000).**
***fp1*:**	**The absolute position of the axis 1 in Pulses (-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**The absolute position of the axis 2 in Pulses (-2,000,000,000 ~ +2,000,000,000)**

Return:

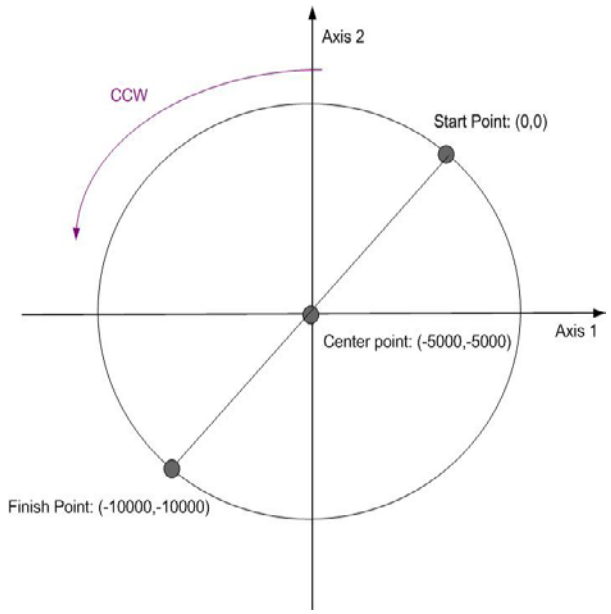
0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_ABS_ARC_CCW is 0x0AFA.
The Sub_function code of RSM_ABS_MACRO_ARC_CCW is 0x0CFA.

MODBUS example:

RSM_ARC_CCW (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 09	12
Register[]	**Value (hex)**	**Remarks**			
0	0A FA/0C FA	*Sub_function 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)*			

7.2.18 2-Axis Circular Interpolation Motion (Complete CW Circle)

eRTU RSM_CIRCLE_CW (HANDLE *hRsm*, BYTE *SlaveAddr*, long *cp1*, long *cp2*)
※Δ eRTU RSM_MACRO_CIRCLE_CW (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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 chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_CIRCLE_CW is 0A 69.
The Sub_function code of RSM_MACRO_CIRCLE_CW is 0C 69.

MODBUS example:

```
RSM_CIRCLE_CW (hRsm, 1, 0, 10000);
// execute a circular motion (a complete circle) in a CW direction
// on module 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0A 69/0C 69	*Sub_function 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)*			

7.2.19 2-Axis Circular Interpolation Motion (Complete CCW Circle)

eRTU RSM_CIRCLE_CCW (HANDLE *hRsm*, BYTE *SlaveAddr*, long *cp1*, long *cp2*)

※Δ eRTU RSM_MACRO_CIRCLE_CCW (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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 chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the errors.

Remark:

The Sub_function code of RSM_CIRCLE_CCW is 0A 6A.
The Sub_function code of RSM_MACRO_CIRCLE_CCW is 0C 6A.

MODBUS example:

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

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0A 6A/0C 6A	*Sub_function 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)*			

7.3 Synchronous Actions

7.3.1 Setting the Activation Factors

eRTU RSM_SYNC_ACTION (HANDLE *hRsm,* BYTE *SlaveAddr*, BYTE *axis1*, BYTE *axis2*, DWORD *nSYNC*, BYTE *nDRV*, BYTE *nLATCH*, BYTE *nPRESET*, BYTE *nOUT*, BYTE *nINT*, BYTE *isrNoX*, BYTE *isrNoY*, BYTE *isrNoZ*, BYTE *isrNoU*)

※Δ **eRTU** RSM_MACRO_SYNC_ACTION (HANDLE *hRsm,* BYTE *SlaveAddr*, 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.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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 4).**
***axis2*:**	**This defined the other axes (or axis) that will take action when one of the activation factors occurs.**

***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≥C+	The logical/real position counter value exceeded the COMP+ register value. Use the RSM_SET_COMPARE () function for selection of a logical/real position (7.3.6).
0x00000002	P<C+	The logical/real position counter value became less than the COMP+ register value. Use the RSM_SET_COMPARE () function for selection of a logical/real position (7.3.6).
0x00000004	P<C-	The logical/real position counter value became less than the COMP- register value. Use the RSM_SET_COMPARE () function for selection of a logical/real position (7.3.6).
0x00000008	P≥C-	The logical/real position counter value exceeded the COMP- register value. Use the RSM_SET_COMPARE () function for selection of a logical/real position (7.3.6).
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.

Example: Choose P≥C+ and IN3↑ (0x00000001 + 0x00000040 = 0x00000041)

***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 RSM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, RSM_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 RSM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, RSM _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 has occured, the RSM_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 RSM_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 RSM_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 RSM_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 RSM_SET_PRESET() function. [V ← PRESET]

Must be used with RSM_SET_PRESET together, please refer to section 7.3.4.

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

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

Must be used with RSM_SET_OUT together, please refer to section 7.3.5.

***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 RSM_ENABLE_INT together, please refer to section 7.3.6.

***isrNoX*:** **ISR1 ~ ISR20 : Specify the interrupt number of X-axis.**
0: disable
***isrNoY*:** **ISR1 ~ ISR20 : Specify the interrupt number of Y-axis.**
0: disable
***isrNoZ*:** **ISR1 ~ ISR20 : Specify the interrupt number of Z-axis.**
0: disable
***isrNoU*:** **ISR1 ~ ISR20 : Specify the interrupt number of U-axis.**
0: disable

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SYNC_ACTION is 0A 6E.
The Sub_function code of RSM_MACRO_SYNC_ACTION is 0C 6E.

MODBUS example:

RSM_SYNC_ACTION (hRsm, SlaveAddr, AXIS_U, AXIS_U, 0X00000040, 0, 2, 4, 0, 0, 0, 0, 0, 0);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 0E	1C
Register[]	**Value (hex)**	**Remarks**			
0	0A 6E/0C 6E	*Sub_function code*			
1	00 08	*axis1 (8 = AXIS_U)*			
2	00 08	*axis2*			
3	00 00	*MSW of nSYNC*			
4	00 40	*LSW of nSYNC (0x40 ➔ rising edge of IN3 will trigger the action)*			
5	00 00	*nDRV (0 ➔ no driving control)*			
6	00 02	*nLATCH (2 ➔ will latch EP value)*			
7	00 04	*nPRESET (4 ➔ set new velocity when SYNC condition is true)*			
8	00 00	*nOUT (0 ➔ no pulse out)*			
9	00 00	*nINT (0 ➔ disable)*			
10	00 00	*isrNoX (It can be any value since nINT is disable.)*			
11	00 00	*isrNoY (It can be any value since nINT is disable.)*			
12	00 00	*isrNoZ (It can be any value since nINT is disable.)*			
13	00 00	*isrNoU (It can be any value since nINT is disable.)*			

Related example:

```
//Example1: When the U axis received IN3 positive edge trigger signal, it
//will change the LATCH encoder value.
 RSM_SYNC_ACTION(hRsm, SlaveAddr, AXIS_U, AXIS_U, 0X00000040, 0, 2,
4, 0, 0, 0, 0, 0, 0);
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_U, 5000);
//Set the maximum speed of u axis on module 1 to 5K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_U, 0);
// set the speed profile for u axis as a symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_U, 2000);
// set the speed of u axis on module 1 to 2000 PPS
RSM_SET_A(hRsm, SlaveAddr, AXIS_U, 100000);
// set the acceleration value of u axis to 100K PPS/sec.
RSM_SET_SV(hRsm,SlaveAddr, AXIS_U, 100);
//set the start velocity of u axis to  100 PPS
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_U, 10000);
```

```
//move 10000 Pulses for u axis on module 1
RSM_SET_PRESET(h SlaveAddr, AXIS_U, 100);
// set the new speed of u axis on module 1 to 100 PPS
RSM_STOP_WAIT(hRsm, SlaveAddr, AXIS_U);
long Vsb = RSM_GET_LATCH(hRsm, SlaveAddr, AXIS_U);


//Example2: When the EP value of u axis exceeds COMP+(5000), it will start
//the Y-axis moving 2,000 PPS.
 RSM_SYNC_ACTION(hRsm, SlaveAddr, AXIS_U, AXIS_Y, 0X00000001, 1, 0,
3, 0, 0, 0, 0, 0, 0);
RSM_SET_COMPARE(hRsm, SlaveAddr, AXIS_U, 0, 1, 5000);
//Set the value of COMP+ is 5000, source reference the EP of U axis.
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_YU, 9000);
// Set the maximum speed of YU axes on module 1 to 9K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_YU, 0);
// set the speed profile for YU axes as symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_YU, 3000);
// set the speed of YU axes on module 1 to 3000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_YU, 200000);
// set the acceleration value of YU axes to 200K PPS/sec.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_YU, 200);
// set the start velocity of yu axes to 200 PPS
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_U, 10000);
// move 10000 Pulses for U axis on module 1
RSM_SET_PRESET(hRsm, SlaveAddr, AXIS_Y, 2000);
// Set the Y axis moving 2000 PPS。


//Example3: When the LP value of X axis exceeds COMP+(200), it will
//start the nout output 5V of Y axis.
RSM_SYNC_ACTION(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0X00000001, 0, 0,
0, 1, 0, 0,       0, 0, 0);
RSM_SET_COMPARE(hRsm, SlaveAddr, AXIS_X, 0, 0, 200);
//Set the value of COMP+ is 200, source reference the LP of X axis.
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_X, 5000);
// Set the maximum speed of X axis on module 1 to 5K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_X, 0);
// set the speed profile for X axis as a symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_X, 2000);
// set the speed of X axis on module 1 to 2000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_X, 100000);
// set the acceleration value of X axis to 100K PPS/s.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_X, 100);
// set the start velocity of X axis to 100 PPS
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_X, 500);
```

```
        // move 500 Pulses for X axis on module 1
        RSM_SET_OUT(hRsm, SlaveAddr, AXIS_Y, 1, 0);
        // Set the Y axis high level output 5V.


        //Example4: When the LP value of X axis exceeds COMP+(5000), it will
        emergency stop and generate an interrupt to call ISR1.
        RSM_SYNC_ACTION(hRsm, SlaveAddr, AXIS_X, AXIS_X, 0X00000001, 6, 0,
0, 0, 1,                ISR1, 0, 0, 0);
        RSM_ENABLE_INT(hRsm, SlaveAddr);  //Enable interrupt
        RSM_SET_COMPARE(hRsm, SlaveAddr, AXIS_X, 0, 0, 5000);
        // Set the value of COMP+ is 5000, source reference the LP of X axis.
        RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XY, 5000);
        // Set the maximum speed of XY axes on module 1 to 5K PPS.
        RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XY, 0);
        // set the speed profile for XY axes as symmetric T-curve.
        RSM_SET_V(hRsm, SlaveAddr, AXIS_XY, 2000);
        // set the speed of XY axes on module 1 to 2000 PPS.
        RSM_SET_A(hRsm, SlaveAddr, AXIS_XY, 100000);
        // set the acceleration value of XY axes to 100K PPS/sec.
        RSM_SET_SV(hRsm, SlaveAddr, AXIS_XY, 100);
        // set the start velocity of XY axes  to 100 PPS.
        RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_X, 10000);
        // move 10K Pulses for X axis on module 1


        ISR1:
        RSM_MP_ISR_CREATE(hRsm, SlaveAddr, ISR1); // Create ISR1
        RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_Y, 1000);
        //move 1000 Pulses  for y axis on module 1
        RSM_MACRO_MP_ISR_CLOSE(hRsm, SlaveAddr); // End ISR1


        //Example5: Use the for loop to iterate, when the LP value of X axis
        // exceeds COMP+(38000), it will generate an interrupt to call ISR1, and
        // determine whether receive IN3 signal in the ISR1, if receive then
        // emergency stop.
        RSM_MP_CREATE(hRsm, SlaveAddr, MP59); //Create a macro program
MP59.
        RSM_MACRO_SET_MAX_V(hRsm, SlaveAddr, AXIS_XY, 5000);
        // Set the maximum speed of XY axes on module 1 to 5K PPS.
        RSM_MACRO_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XY, 0);
        // set the speed profile for XY axes as symmetric T-curve.
        RSM_MACRO_SET_V(hRsm, SlaveAddr, AXIS_XY, 2000);
        // set the speed of XY axes  on module 1 to 2000 PPS.
        RSM_ MACRO_SET_A(hRsm, SlaveAddr, AXIS_XY, 100000);
```

```
        // set the acceleration value of XY axes to 100K PPS/S.
        RSM_ MACRO_SET_SV(hRsm, SlaveAddr, AXIS_XY, 100);
        // set the start velocity of XY axes to 100 PPS.
        RSM_MACRO_FOR(hRsm, SlaveAddr, 100); //Set the times of for loop is
100.
        RSM_ MACRO_SET_LP(hRsm, SlaveAddr, AXIS_XY, 0);
        //Set the LP value of XYaxes to 0
        RSM_MACRO_SET_FILTER(hRsm, SlaveAddr, AXIS_X, 16, 1); //Set the filter
of IN3
        RSM_MACRO_CLEAR_SYNC_ACTION(hRsm, SlaveAddr, AXIS_X);
        //Clear all the sync_action conditions of X axis.
        RSM_MACRO_SYNC_ACTION(hRsm, SlaveAddr, AXIS_X, AXIS_X,
0X00000001, 0, 0,      0, 0, 1, ISR1, 0, 0, 0);
        // Set the LP value of X axis exceeds COMP+, it will generate an interrupt to
        // call ISR1.
        RSM_MACRO_ENABLE_INT(hRsm, SlaveAddr);  //Enable interrupt
        RSM_MACRO_SET_COMPARE(hRsm, SlaveAddr, AXIS_X, 0, 0, 38000);
        //Set the value of COMP+ is 38000, source reference the LP of X axis.
        RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_X, 42000);
        // move 42K Pulses for X axis on module 1
        RSM_MACRO_STOP_WAIT(hRsm, SlaveAddr, AXIS_X); //Wait until X axis
stop.   RSM_MACRO_NEXT(hRsm, SlaveAddr); //Match with for loop.
        RSM_MACRO_MP_CLOSE(hRsm, SlaveAddr); // End a macro program
MP59


        ISR1:
        RSM_MP_ISR_CREATE(hRsm, SlaveAddr, ISR1); //Create ISR1
        RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_Y, 1000);
        // move 1000 Pulses for Y axis on module 1.
        RSM_MACRO_SYNC_ACTION(hRsm, SlaveAddr, AXIS_X, AXIS_X,
0X00000040, 6, 0,      0, 0, 0, 0, 0, 0, 0); //Set the X axis to receive the IN3 positive
edge trigger   //signals from low level to high level, then emergency stop.
        RSM_MACRO_MP_ISR_CLOSE(hRsm, SlaveAddr); // End ISR1
```

7.3.2 Clear Synchronization Condition

eRTU RSM_CLEAR_SYNC_ACTION (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *triggerAxis*)

※Δ eRTU RSM_MACRO_CLEAR_SYNC_ACTION (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *triggerAxis*)

Description:

Clear all the synchronization conditions of the triggered axis.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
triggerAxis	**The axis number set by RSM_SYNC_ACTION (Please refer to Table 4)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_CLEAR_SYNC_ACTION is 0A 6A.
The Sub_function code of RSM_MACRO_CLEAR_SYNC_ACTION is 0C 6A.

MODBUS example:

```
RSM_CLEAR_SYNC_ACTION (hRsm, 1, AXIS_X);
// Clear all the sync_action conditions of X axis.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 95/0C 95	***Sub_function code***			
1	00 01	***TriggerAxis***			

7.3.3 Set the Synchronization Condition

eRTU RSM_SET_ACTIVATION_FACTORS(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *triggerAxis*, DWORD *nSYNC*)

※Δ eRTU RSM_MACRO_SET_ACTIVATION_FACTORS (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *triggerAxis*, DWORD *nSYNC*)

Description:

Set the synchronization conditions of triggered axis.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
triggerAxis	**Set the axis number(Please refer to Table 4)**
***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≥C+	The logical/real position counter value exceeded the COMP+ register value. Use the RSM_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 RSM_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 RSM_SET_COMPARE () function for selection of a logical/real position (6.3.2).
0x00000008	P≥C-	The logical/real position counter value exceeded the COMP- register value. Use the RSM_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.

Example: Choose P≥C+ and IN3 ↑ (0x00000001 + 0x00000040 = 0x00000041)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_ACTIVATION_FACTORS is 0A 6A.
The Sub_function code of RSM_MACRO_SET_ACTIVATION_FACTORS is 0C 6A.

MODBUS example:

RSM_SET_ACTIVATION_FACTORS (hRsm, 1, AXIS_X, 0X00000001);
// Set the synchronization conditions of x axis is P≥C+.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 96/0C 96	*Sub_function code*			
1	00 01	*TriggerAxis*			
2	00 00	*MSW of nSYNC*			
3	00 01	*LSW of nSYNC*			

7.3.4 Set the Synchronization Axis

eRTU RSM_SET_ACTIVATION_AXIS(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *triggerAxis*, BYTE *activationAxis*)

※Δ eRTU RSM_MACRO_SET_ACTIVATION_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *triggerAxis*, BYTE *activationAxis*)

Description:

Set the synchronization axis.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
triggerAxis	Set the axis number(Please refer to Table 4)
activationAxis:	Set the synchronization axis number(Please refer to Table 4)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_ACTIVATION_AXIS is 0A 97.
The Sub_function code of RSM_MACRO_SET_ACTIVATION_AXIS is 0C 97.

MODBUS example:

```
RSM_SET_ACTIVATION_AXIS (hRsm, 1, AXIS_X, AXIS_Y);
// Set the synchronization axis of x axis is y axis.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A 97/0C 97	*Sub_function code*			
1	00 01	*TriggerAxis(AXIS_X)*			
2	00 02	*ActivationAxis(AXIS_Y)*			

7.3.5 Set the Synchronization Action

eRTU RSM_SET_ACTION(HANDLE *hRsm,* BYTE *SlaveAddr*, BYTE *activationAxis*, BYTE *nDRV*, BYTE *nLATCH*, BYTE *nPRESET*, BYTE *nOUT*, BYTE *nINT*, BYTE *isrNoX*, BYTE *isrNoY*, BYTE *isrNoZ*, BYTE *isrNoU*)

※Δ eRTU RSM_MACRO_SET_ACTION (HANDLE *hRsm,* BYTE *SlaveAddr*, BYTE *activationAxis*, BYTE *nDRV*, BYTE *nLATCH*, BYTE *nPRESET*, BYTE *nOUT*, BYTE *nINT*, BYTE *isrNoX*, BYTE *isrNoY*, BYTE *isrNoZ*, BYTE *isrNoU*)

Description:

Set the action of synchronization motion.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***activationAxis*:**	**Set the synchronization axis number(Please refer to Table 4)**

***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 RSM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, RSM_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 RSM_SET_PRESET() function will set the offset value for this FDRV. Therefore, the companion function, RSM_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 occoured, the RSM_GET_LATCH() function can be use to get the latched value, please refer to section 7.3.3.

***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 RSM_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 RSM_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 RSM_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 RSM_SET_PRESET() function. [V ← PRESET]**

Must be used with RSM_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 RSM_SET_OUT together, please refer to section 7.3.5.

***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 RSM_ENABLE_INT together, please refer to section 7.3.6.

***isrNoX*:** **ISR1 ~ ISR20 : Specify the interrupt number of X-axis.**
0: disable
***isrNoY*:** **ISR1 ~ ISR20 : Specify the interrupt number of Y-axis.**
0: disable
***isrNoZ*:** **ISR1 ~ ISR20 : Specify the interrupt number of Z-axis.**
0: disable
***isrNoU*:** **ISR1 ~ ISR20 : Specify the interrupt number of U-axis.**
0: disable

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_ACTION is 0A 98.
The Sub_function code of RSM_MACRO_SET_ACTION is 0C 98.

MODBUS example:

```
RSM_SET_ACTION (hRsm, 1, AXIS_Y, 0, 2, 4, 0, 0, 0, 0, 0, 0);
// Set the action of synchronization motion is when the IN3 positive
//edge trigger signals received, it will change the velocity and the
//encoder of LATCH.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 0B	16
Register[]	**Value (hex)**	**Remarks**			
0	0A 98/0C 98	*Sub_function code*			
1	00 02	*ActivationAxis(AXIS_Y)*			
2	00 00	*nDRV (0 ➔ no driving control)*			
3	00 02	*nLATCH (2 ➔ will latch EP value)*			
4	00 04	*nPRESET (4 ➔ set new velocity when SYNC condition is true)*			
5	00 00	*nOUT (0 ➔ no pulse out)*			
6	00 00	*nINT (0 ➔ disable)*			
7	00 00	*isrNoX (It can be any value since nINT is disable.)*			
8	00 00	*isrNoY (It can be any value since nINT is disable.)*			
9	00 00	*isrNoZ (It can be any value since nINT is disable.)*			
10	00 00	*isrNoU (It can be any value since nINT is disable.)*			

Related example:

```
//Example1: Set two groups synchronization motion, when the LP value of x
//axis exceeds COMP+(30000), the synchronization motion axis z-axis
//generate an interrupt to call ISR1, and when the LP value of y axis
//exceeds COMP+(35000), the synchronization motion axis u-axis generate
//an interrupt to call ISR2.
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 5000);
// Set the maximum speed of xyzu axes on module 1 to 5K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU, 0);
// set the speed profile for xyzu axes as symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
// set the speed of xyzu axes on module 1 to 2000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU, 100000);
// set the acceleration value of xyzu axes to 100K PPS/sec.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 100);
// set the start velocity of xyzu axes to 100 PPS.
RSM_SET_ACTIVATION_FACTORS(hRsm, SlaveAddr, AXIS_XY, 1);
// Set the synchronization conditions of xy axes is P≥C+.
RSM_SET_ACTIVATION_AXIS(hRsm, 1, AXIS_X, AXIS_Z);
// Set the synchronization axis of x axis is z axis.
RSM_SET_ACTIVATION_AXIS(hRsm, 1, AXIS_Y, AXIS_U);
```

```
// Set the synchronization axis of y axis is u axis.
RSM_SET_ACTION (hRsm, 1, AXIS_Z, 0, 0, 0, 0, 1, 0, 0, ISR1, 0);
// Set the action of synchronization motion axis z-axis is to generate an
// interrupt to call ISR1.
RSM_SET_ACTION (hRsm, 1, AXIS_U, 0, 0, 0, 0, 1, 0, 0, 0, ISR2);
// Set the action of synchronization motion axis u-axis is to generate an
// interrupt to call ISR2.
RSM_ENABLE_INT(hRsm, SlaveAddr);  //Enable interrupt
RSM_SET_COMPARE (hRsm, SlaveAddr, AXIS_X, 0, 0, 30000);
//Set the value of COMP+ is 30000, source reference the LP of x axis.
RSM_SET_COMPARE (hRsm, SlaveAddr, AXIS_Y, 0, 0, 35000);
//Set the value of COMP+ is 35000, source reference the LP of y axis.
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_XY, 42000);
// move 42K Pulses for xy axes on module 1

ISR1:
RSM_MP_ISR_CREATE(hRsm, SlaveAddr, ISR1); // Create ISR1
RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_Z, 10000);
// move 10K Pulses for z axis on module 1
RSM_MACRO_MP_ISR_CLOSE(hRsm, SlaveAddr); // End ISR1

ISR2:
RSM_MP_ISR_CREATE(hRsm, SlaveAddr, ISR2); // Create ISR2
RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_U, 10000);
// move 10K Pulses for u axis on module 1
RSM_MACRO_MP_ISR_CLOSE(hRsm, SlaveAddr); // End ISR2
```

7.3.6 COMPARE Value Setting

eRTU RSM_SET_COMPARE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nSELECT*, BYTE *nTYPE*, long *data*)

※Δ eRTU RSM_MACRO_SET_COMPARE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nSELECT*, BYTE *nTYPE*, long *data*)

Description:

This function sets the values of COMPARE registers. Howerer, it will disable the software limit setting.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
***SlaveAddr*:**	Modbus RTU Slave Address (valid range: 1 to 247)
***axis*:**	Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)
***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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_COMPARE is 0A 70.
The Sub_function code of RSM_MACRO_SET_COMPARE is 0C 70.

MODBUS example:

```
RSM_SET_COMPARE (hRsm, SlaveAddr, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 06	0C
Register[]	**Value (hex)**	**Remarks**			
0	0A 70/0C 70	*Sub_function 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)*			

7.3.7 Get LATCH Value

eRTU RSM_GET_LATCH (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long* *LatchValue*)

※Δ eRTU RSM_MACRO_GET_LATCH (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

eRTU RSM_GET_LATCH_4_AXIS (HANDLE *hRsm*, BYTE *SlaveAddr*, long* *LatchValueX*, long* *LatchValueY*, long* *LatchValueZ*, long* *LatchValueU*)

Description:

This function gets the values from the LATCH register.

Category:

MODBUS table, MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**
***LatchValue*:**	**The pointer points to the memory that stores the value of LATCH. (-2,000,000,000 ~ +2,000,000,000).**
LatchValueX:	**The pointer points to the memory that stores the value of LATCH. (-2,000,000,000 ~ +2,000,000,000) of x axis.**
LatchValueY:	**The pointer points to the memory that stores the value of LATCH. (-2,000,000,000 ~ +2,000,000,000) of y axis.**
LatchValueZ:	**The pointer points to the memory that stores the value of LATCH. (-2,000,000,000 ~ +2,000,000,000) of z axis.**
LatchValueU:	**The pointer points to the memory that stores the value of LATCH. (-2,000,000,000 ~ +2,000,000,000) of u axis.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

- **Method 1: It can get latched values directly.**

The latched values are 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	62 (0x3E)	MSW of X_LATCH
X_LATCH_LSW	63 (0x3F)	LSW of X_LATCH
Y_LATCH_MSW	64 (0x40)	MSW of Y_LATCH
Y_LATCH_LSW	65 (0x41)	LSW of Y_LATCH
Z_LATCH_MSW	66 (0x42)	MSW of Z_LATCH
Z_LATCH_LSW	67 (0x43)	LSW of Z_LATCH
U_LATCH_MSW	68 (0x44)	MSW of U_LATCH
U_LATCH_LSW	69 (0x45)	LSW of U_LATCH

```
long LATCH_Y;
RSM_GET_LATCH (hRsm, 1, AXIS_Y, &LATCH_Y);
```

The MODBUS command is listed as follows:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 40	00 02

The response via MODBUS may be:

UID (hex)	FC (hex)	Byte Count (hex)	*MSW of Y_LATCH* (Reg_0)	*LSW of Y_LATCH* (Reg_1)
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 RSM_MACRO_SET_RVAR() followed to save the return latched value. Please refer to MP related explanation literature.

```
RSM_MACRO_GET_LATCH (hRsm, 1, AXIS_Y);
//Get the latched value which is from Y-axis of card 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C 71	*Sub_function code*			
1	00 02	*axis*			

7.3.8 PRESET Data for Synchronous Action Setting

eRTU RSM_SET_PRESET (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

※Δ eRTU RSM_MACRO_SET_PRESET (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, long *data*)

Description:

This function sets the PRESET value for synchronous action. The synchronous action axis can not be set individually.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**The number of synchronization motion axis, this axis must be assined as the axis2 parameter of RSM_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 description of RSM_SET_MAX_V. If there are more than two synchronous action axes, please set RSM_SET_MAX_V to be same value.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_SET_PRESET is 0A 72.
The Sub_function code of RSM_MACRO_SET_PRESET is 0C 72.

MODBUS example:

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.

```
RSM_SET_PRESET (hRsm, SlaveAddr, AXIS_U, 100);
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 72/0C 72	*Sub_function code*			
1	00 08	*Axis (8 ➔ AXIS_U)*			
2	00 00	*MSW of data*			
3	00 64	*LSW of data (100 = 0x64)*			

7.3.9 OUT Data Setting

eRTU RSM_SET_OUT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *outEdge*, BYTE *PulseWidth*)

※Δ eRTU RSM_MACRO_SET_OUT (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *outEdge*, BYTE *PulseWidth*)

Description:

This function configures the output pulse settings.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4), currently only support Axes X & Y**
***outEdge*:**	**Output active logic: 0 = low active; 1 = high active**
***PulseWidth*:**	**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 chapter 2.2)

Remark:

The Sub_function code of RSM_SET_OUT is 0A 73.
The Sub_function code of RSM_MACRO_SET_OUT is 0C 73.

MODBUS example:

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.

RSM_SET_OUT (hRsm, 1, AXIS_U, 1, 0);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 73/0C 73	***Sub_function code***			
1	00 08	***Axis (8 ➔ AXIS_U)***			
2	00 01	***outEdge***			
3	00 00	***PulseWidth***			

7.3.10 Enable Interrupt Functions of i-8094H

eRTU RSM_ENABLE_INT (HANDLE *hRsm*, BYTE *SlaveAddr*)

※ eRTU RSM_MACRO_ENABLE_INT (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function enables the interrupt function of the motion chip inside the i-8094H.

Category:

MODBUS sub_function; RTC, MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_ENABLE_INT is 0A AA.
The Sub_function code of RSM_MACRO_ENABLE_INT is 0C AA.

MODBUS example:

RSM_ENABLE_INT (hRsm, 1);
//Enable the interrupt function for module 1.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A AA/0C AA	***Sub_function code***			

7.3.11 Disable Interrupt Functions of i-8094H

eRTU RSM_DISABLE_INT (HANDLE *hRsm*, BYTE *SlaveAddr*)

※ eRTU RSM_MACRO_DISABLE_INT (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function disables the interrupt function of the motion chip inside the i-8094H.

Category:

MODBUS sub_function; RTC, MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_DISABLE_INT is 0A AB.
The Sub_function code of RSM_MACRO_DISABLE_INT is 0C AB.

MODBUS example:

RSM_DISABLE_INT (hRsm, 1);
//Disable the interrupt function for module 1.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A AB/0C AB	***Sub_function code***			

7.3.12 Set Interrupt Factor of i-8094H

eRTU RSM_INTFACTOR_ENABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nINT*, BYTE *isrNo*)

※Δ eRTU RSM_MACRO_INTFACTOR_ENABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nINT*, BYTE *isrNo*)

Description:

This function sets the condition factor of interrupt of the motion chip.

Category:

MODBUS sub_function; RTC, MP, ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**

***nINT*:** **condition factors of interrupt, please refer to the following table**

Table 11: Motion chip interrupt factor table

nINT	Symbol	Description
0	PULSE	Interupt 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. Please refer to RSM_SET_COMPARE() (6.3.2)
2	P<C-	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register Please refer to RSM_SET_COMPARE() (6.3.2)
3	P<C+	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register. Please refer to RSM_SET_COMPARE() (6.3.2)
4	P>=C+	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register. Please refer to RSM_SET_COMPARE() (6.3.2)
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.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

This function is conflict with RSM_SET_SLMT(2.10).
The Sub_function code of RSM_INTFACTOR_ENABLE is 0A AC.
The Sub_function code of RSM_MACRO_INTFACTOR_ENABLE is 0C AC.

MODBUS example:

This function assigns an ISRn to process an interrupt that happens at a specified axis.

```
RSM_INTFACTOR_ENABLE (hRsm, 1, AXIS_X, 4, ISR1);//When the position counter of x axis exceeds C+, then call ISR1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A AC/0C AC	*Sub_function code*			
1	00 01	*axis (1 ➔ AXIS_X)*			
2	00 04	*nINT*			
3	00 01	*isrNo (ISR1 = 1)*			

Related example:

//Example1: Using call the ISR to set multi-group compare.

ISR1:
Set Compare >=4000 Call ISR2
Fixed Move Y 1000

ISR2:
Set Compare >=6000 Call ISR3
Fixed Move Z 1000

ISR3:
Fixed Move U 1000

```
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 5000);
// Set the maximum  speed of xyzu axes on module 1 to 5K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XYZU, 0);
// set the speed profile for xyzu axes as symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XYZU, 2000);
// set the speed of xyzu axes on module 1 to 2000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_XYZU, 100000);
// set the acceleration value of xyzu axes to 100K PPS/S.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XYZU, 100);
// set the start velocity of xyzu axes to 100 PPS.
RSM_SET_COMPARE(hRsm, SlaveAddr, AXIS_X, 0, 0, 2000);
//Set the value of COMP+ is 2000, source reference the LP of x axis.
RSM_INTFACTOR_ENABLE (hRsm, 1, AXIS_X, 4, ISR1);
//When the LP value of x axis exceeds C+(2000), then call ISR1.
RSM_ENABLE_INT (hRsm, 1);       //Enable interrupt
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_X, 16000);
// move 16K Pulses for x axis on module 1

ISR1:
RSM_MP_ISR_CREATE(hRsm, SlaveAddr, ISR1); //Create ISR1
RSM_MACRO_SET_COMPARE(hRsm, SlaveAddr, AXIS_X, 0, 0, 4000);
//Set the value of COMP+ is 4000, source reference the LP of x axis.
RSM_MACRO_INTFACTOR_ENABLE (hRsm, 1, AXIS_X, 4, ISR2);
```

```
//When the LP value of x axis exceeds C+(4000), then call ISR2.
RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_Y, 1000);
// move 1000 Pulses for y axis on module 1
RSM_MACRO_MP_ISR_CLOSE(hRsm, SlaveAddr); // End ISR1

ISR2:
RSM_MP_ISR_CREATE(hRsm, SlaveAddr, ISR2); //Create ISR2
RSM_MACRO_SET_COMPARE(hRsm, SlaveAddr, AXIS_X, 0, 0, 6000);
//Set the value of COMP+ is 6000, source reference the LP of x axis.
RSM_MACRO_INTFACTOR_ENABLE (hRsm, 1, AXIS_X, 4, ISR3);
//When the LP value of x axis exceeds C+(6000), then call ISR3.
RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_Z, 1000);
// move 1000 Pulses for z axis on module 1
RSM_MACRO_MP_ISR_CLOSE(hRsm, SlaveAddr); // End ISR2

ISR3:
RSM_MP_ISR_CREATE(hRsm, SlaveAddr, ISR3); //Create ISR3
RSM_MACRO_FIXED_MOVE(hRsm, SlaveAddr, AXIS_U, 1000);
// move 1000 Pulses for u axis on module 1
RSM_MACRO_MP_ISR_CLOSE(hRsm, SlaveAddr); // End ISR3
```

7.3.13 Clear Interrupt Factor of i-8094H

eRTU RSM_INTFACTOR_DISABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nINT*)

※Δ eRTU RSM_MACRO_INTFACTOR_DISABLE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, BYTE *nINT*)

Description:

This function disables the condition factor interrupt of the motion chip.

Category:

MODBUS sub_function; RTC, MP, ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8)**

***nINT*: condition factors of interrupt, please refer to the following table**

Number	Symbol	Description
0	PULSE	Interupt 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
10	DISABLE ALL	Disable all the trigger conditions interrupt factor

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_INTFACTOR_DISABLE is 0A AD.
The Sub_function code of RSM_MACRO_INTFACTOR_DISABLE is 0C AD.

MODBUS example:

```
RSM_INTFACTOR_DISABLE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0A AD/0C AD	*Sub_function code*			
1	00 0F	*axis (0xF ➔ AXIS_XYZU)*			
2	00 04	*nINT*			

Related example:

```
RSM_INTFACTOR_DISABLE(hRsm, 1, AXIS_XYZU, 1);
RSM_INTFACTOR_DISABLE(hRsm, 1, AXIS_XYZU, 2);
RSM_INTFACTOR_DISABLE(hRsm, 1, AXIS_XYZU, 3);
RSM_INTFACTOR_DISABLE(hRsm, 1, AXIS_XYZU, 4);
// Disable the interrupt factors of Module 1
```

7.4 Continuous Interpolation

7.4.1 2-Axis Rectangular Motion

eRTU RSM_RECTANGLE (HANDLE *hRsm,* BYTE *SlaveAddr*, BYTE *axis1*, BYTE *axis2*, BYTE *nAcc*, BYTE *Sp*, BYTE *nDir*, long *Lp*, long *Wp*, long *Rp*, DWORD *RSV*, DWORD *RV*, DWORD *RA*, DWORD *RD*)

※ eRTU RSM_MACRO_RECTANGLE (HANDLE *hRsm,* BYTE *SlaveAddr*, 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 of the same value and it can also be changed. The deceleration point will be calculated automatically. This is a command that appears in various motion applications.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis1*:**	**The first axis (axis 1). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***axis2*:**	**The second axis (axis 2). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***nAcc*:**	**0 → constant vector speed interpolation mode 1 → symmetric T-curve Acc/Dec interpolation mod**
***Sp*:**	**Start point 0 ~ 7. (Sp0 ~ Sp7 are defined in the following.**
***nDir*:**	**Direction of movement 0: CCW; 1: CW**
***Lp*:**	**Length in Pulses (1 ~ 2,000,000,00)**
***Wp*:**	**Width in Pulses (1 ~ 2,000,000,00)**
***Rp*:**	**Radius of each in pulses (1 ~ 2,000,000,00)**
***RSV*:**	**Starting speed (in PPS)**
***RV*:**	**Vector speed (in PPS)**
***RA*:**	**Acceleration (PPS/sec)**
***RD*:**	**Deceleration of the last segment (in PPS/sec)**

Return:

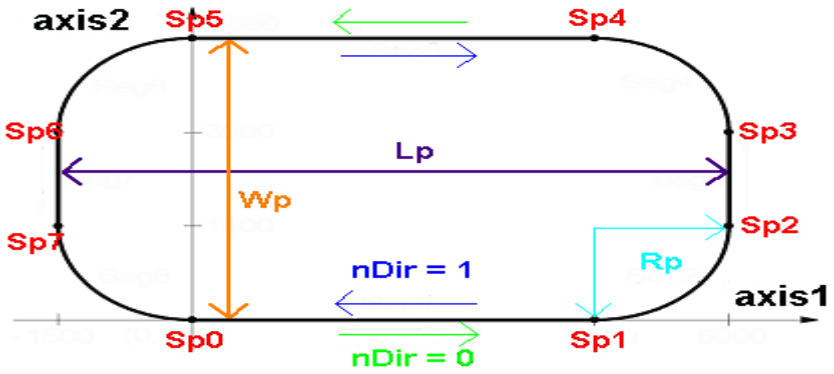
0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_RECTANGLE is 0A 78.
The Sub_function code of RSM_MACRO_RECTANGLE is 0C 78.

MODBUS example:

RSM_RECTANGLE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 14	28
Register[]	**Value (hex)**	**Remarks**			
0	0A 78/0C 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)***
17	**13 88**	***LSW of RA (5000 = 1388)***
18	**00 00**	***MSW of RD (define dec value)***
19	**13 88**	***LSW of RD (5000 = 1388)***

7.4.2 Set the Speed of 2-Axis Continuous Linear Interpolation

eRTU RSM_LINE_2D_INITIAL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis1*, BYTE *axis2*, DWORD *VSV* , DWORD *VV* , DWORD *VA*)

※ eRTU RSM_MACRO_LINE_2D_INITIAL (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis1*:**	**The first axis (axis 1). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***axis2*:**	**The second axis (axis 2). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***VSV*:**	**Starting speed (in PPS)**
***VV*:**	**Vector speed (in PPS)**
***VA*:**	**Vector acceleration (PPS/sec)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_LINE_2D_INITIAL is 0A 7C.
The Sub_function code of RSM_MACRO_LINE_2D_INITIAL is 0C 7C.

MODBUS example:

```
RSM_LINE_2D_INITIAL (hRsm, 1, AXIS_X, AXIS_Y, 300, 18000, 500000);
//This function should be defined before the RSM_LINE_2D_CONTINUE()
//function is used. Please refer to the example of this function.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 09	12
Register[]	**Value (hex)**	**Remarks**			
0	0A 7C/0C 7C	*Sub_funciton 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	*MSW of VV*			
6	46 50	*LSW of VV (18000 = 0x4650)*			
7	00 07	*MSW of VA*			
8	A1 20	*LSW of VA (500000 = 0x0007A120)*			

7.4.3 Execute a 2-Axis Continuous Linear Interpolation (Relative Distance)

eRTU RSM_LINE_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*)

※ eRTU RSM_MACRO_LINE_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*)

Description:

This function executes a 2-axis continuous linear interpolation.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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 chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_LINE_2D_CONTINUE is 0A 7E.
The Sub_function code of RSM_MACRO_LINE_2D_CONTINUE is 0C 7E.

MODBUS example:

```
RSM_LINE_2D_CONTINUE (hRsm, 1, 0, 100, 100);
//execute X, Y 2-axis linear continuous interpolation on module 1
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 06	0C
Register[]	**Value (hex)**	**Remarks**			
0	0A 7E/0C 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*			

Related example:

```
BYTE SlaveAddr=1;
unsigned short sv=300; //Set the vector start velocity is 300 PPS。
unsigned short v=18000; //Set the vector velocity is 18000 PPS。
unsigned long a=500000; //Set the vector acceleration is 500K PPS/s。
unsigned short loop1;
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 160000L);
// Set the maximum  speed of xyzu axes on module 1 to 160K PPS.
RSM_LINE_2D_INITIAL(hRsm, SlaveAddr, AXIS_X, AXIS_Y, sv, v, a);
for (loop1 = 0; loop1 < 10000; loop1++)
{
        RSM_LINE_2D_CONTINUE (hRsm, SlaveAddr, 0, 100, 100);
        RSM_LINE_2D_CONTINUE (hRsm, SlaveAddr, 0, -100, -100);
        //execute X, Y 2-axis linear continuous interpolation on module 1
}
RSM_LINE_2D_CONTINUE (hRsm, SlaveAddr, 1, 100, 100);
//end X, Y 2-axis linear continuous interpolation on module 1
```

7.4.4 Execute a 2-Axis Continuous Linear Interpolation (Absoult Position)

eRTU RSM_ABS_LINE_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*)

※ eRTU RSM_ABS_MACRO_LINE_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*)

Description:

This function executes a 2-axis continuous linear interpolation motion to the specified target position.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***nType*:**	**0 → 2-axis linear continuous interpolation 1 → end of 2-axis linear continuous interpolation**
***fp1*:**	**Absolute position of axis 1 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**Absolute position of axis 2 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_ABS_LINE_2D_CONTINUE is 0x0AFB.
The Sub_function code of RSM_ABS_MACRO_LINE_2D_CONTINUE is 0x0CFB.

MODBUS example:

```
RSM_ABS_LINE_2D_CONTINUE (hRsm, 1, 0, 100, 100);
//execute X, Y 2-axis linear continuous interpolation on module 1
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 06	0C
Register[]	**Value (hex)**	**Remarks**			
0	0A FB/0C FB	*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*			

Related example:

```
BYTE SlaveAddr=1;
unsigned short sv=300; //Set the vector start velocity is 300 PPS。
unsigned short v=18000; //Set the vector velocity is 18000 PPS。
unsigned long a=500000; //Set the vector acceleration is 500K PPS/s。
unsigned short loop1;
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 160000L);
// Set the maximum  speed of xyzu axes on module 1 to 160K PPS.
RSM_LINE_2D_INITIAL(hRsm, SlaveAddr, AXIS_X, AXIS_Y, sv, v, a);
for (loop1 = 0; loop1 < 10000; loop1++)
{
        RSM_ABS_LINE_2D_CONTINUE (hRsm, SlaveAddr, 0, 100, 100);
        RSM_ABS_LINE_2D_CONTINUE (hRsm, SlaveAddr, 0, -100, -100);
        //execute X, Y 2-axis linear continuous interpolation on module 1
}
RSM_ABS_LINE_2D_CONTINUE (hRsm, SlaveAddr, 1, 100, 100);
//end X, Y 2-axis linear continuous interpolation on module 1
```

7.4.5 Set the Speed of 3-Axis Continuous Linear Interpolation

eRTU RSM_LINE_3D_INITIAL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*, DWORD *VSV* , DWORD *VV* , DWORD *VA*)

※ eRTU RSM_MACRO_LINE_3D_INITIAL (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
axis1:	The first axis (axis 1). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).
axis2:	The second axis (axis 2). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).
axis3:	The third axis (axis 3). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).
VSV:	Starting speed (in PPS)
VV:	Vector speed (in PPS)
VA:	Vector acceleration (PPS/sec)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_LINE_3D_INITIAL is 0A 7F.
The Sub_function code of RSM_MACRO_LINE_3D_INITIAL is 0C 7F.

MODBUS example:

```
RSM_LINE_3D_INITIAL (hRsm, 1, AXIS_X, AXIS_Y, AXIS_Z, 300, 18000, 500000);
//This function should be defined before the RSM_LINE_3D_CONTINUE()
//function is used. Please refer to the example of this function.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 0A	14
Register[]	**Value (hex)**	**Remarks**			
0	0A 7F/0C 7F	*Sub_funciton code*			
1	00 01	*axis1 (1 ➔ AXIS_X)*			
2	00 02	*axis2 (2 ➔ AXIS_Y)*			
3	00 04	*axis3 (4 ➔ AXIS_Z)*			
4	00 00	*MSW of VSV*			
5	01 2C	*LSW of VSV (300 = 0x12C)*			
6	00 00	*MSW of VV*			
7	46 50	*LSW of VV (18000 = 0x4650)*			
8	00 07	*MSW of VA*			
9	A1 20	*LSW of VA (500000 = 0x7A120)*			

7.4.6 Execute a 3-Axis Continuous Linear Interpolation (Relative Distance)

eRTU RSM_LINE_3D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*, long *fp3*)

※ eRTU RSM_MACRO_LINE_3D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*, long *fp3*)

Description:

This function executes a 3-axis continuous linear interpolation.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
nType:	0 → 3-axis linear continuous interpolation 1 → end of 3-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 chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_LINE_3D_CONTINUE is 0A 81.
The Sub_function code of RSM_MACRO_LINE_3D_CONTINUE is 0C 81.

MODBUS example:

```
RSM_LINE_3D_CONTINUE (hRsm, 1, 0, 100, 100, 100);
//execute X, Y, Z 3-axis linear continuous interpolation on module 1
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 08	10
Register[]	**Value (hex)**	**Remarks**			
0	0A 81/0C 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*			
6	00 00	*MSW of fp3*			
7	00 64	*LSW of fp3*			

Related example:

```
BYTE SlaveAddr=1;
unsigned short sv=300; //Set the vector start velocity is 300 PPS。
unsigned short v=18000; //Set the vector velocity is 18000 PPS。
unsigned long a=500000; //Set the vector acceleration is 500K PPS/s。
unsigned short loop1;
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 160000L);
// Set the maximum speed of xyzu axes on module 1 to 160K PPS.
RSM_LINE_3D_INITIAL(hRsm, SlaveAddr, AXIS_X, AXIS_Y, AXIS_Z, sv, v, a);
for (loop1 = 0; loop1 < 10000; loop1++)
{
        RSM_LINE_3D_CONTINUE(hRsm, SlaveAddr, 0, 100, 100, 100);
        RSM_LINE_3D_CONTINUE(hRsm, SlaveAddr, 0, -100, -100, -100);
        //execute X, Y, Z 3-axis linear continuous interpolation on module 1
}
RSM_LINE_3D_CONTINUE (hRsm, 1, 0, 100, 100, 100);
//end X, Y, Z 3-axis linear continuous interpolation on module 1
```

7.4.7 Execute a 3-Axis Continuous Linear Interpolation (Absoult Position)

eRTU RSM_ABS_LINE_3D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*, long *fp3*)

※ eRTU RSM_ABS_MACRO_LINE_3D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nType*, long *fp1*, long *fp2*, long *fp3*)

Description:

This function executes a 3-axis continuous linear interpolation to the absolute position specified.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***nType*:**	**0 → 3-axis linear continuous interpolation 1 → end of 3-axis linear continuous interpolation**
***fp1*:**	**The absolute position of axis 1 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**The absolute position of axis 2 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)**
***fp3*:**	**The absolute position of axis 3 (in Pulses) (-2,000,000,000 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_ABS_LINE_3D_CONTINUE is 0x0AFC.

The Sub_function code of RSM_ABS_MACRO_LINE_3D_CONTINUE is 0x0CFC.

MODBUS example:

```
RSM_ABS_LINE_3D_CONTINUE (hRsm, 1, 0, 100, 100, 100);
//execute X, Y, Z 3-axis linear continuous interpolation on module 1
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 08	10
Register[]	**Value (hex)**	**Remarks**			
0	0A FC/0C FC	*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	00 00	*MSW of fp3*			
7	00 64	*LSW of fp3*			

Related example:

```
BYTE SlaveAddr=1;
unsigned short sv=300; //Set the vector start velocity is 300 PPS。
unsigned short v=18000; //Set the vector velocity is 18000 PPS。
unsigned long a=500000; //Set the vector acceleration is 500K PPS/s。
unsigned short loop1;
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU, 160000L);
// Set the maximum speed of xyzu axes on module 1 to 160K PPS.
RSM_LINE_3D_INITIAL(hRsm, SlaveAddr, AXIS_X, AXIS_Y, AXIS_Z, sv, v, a);
for (loop1 = 0; loop1 < 10000; loop1++)
{
        RSM_ABS_LINE_3D_CONTINUE(hRsm, SlaveAddr, 0, 100, 100, 100);
        RSM_ABS _LINE_3D_CONTINUE(hRsm, SlaveAddr, 0, -100, -100, -
100);
        //execute X, Y, Z 3-axis linear continuous interpolation on module 1
}
RSM_LINE_3D_CONTINUE (hRsm, 1, 0, 100, 100, 100);
//end X, Y, Z 3-axis linear continuous interpolation on module 1
```

7.4.8 Set the Speed of Mixed 2-axis motions in Continuous Interpolation

eRTU RSM_MIX_2D_INITIAL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis1*, BYTE *axis2*, BYTE *nAcc*, DWORD *VSV* , DWORD *VV* , DWORD *VA*)

※ eRTU RSM_MACRO_MIX_2D_INITIAL (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
axis1:	The first axis (axis 1). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).
axis2:	The second axis (axis 2). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 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 chapter 2.2)

Remark:

The Sub_function code of RSM_MIX_2D_INITIAL is 0A 82.
The Sub_function code of RSM_MACRO_MIX_2D_INITIAL is 0C 82.

MODBUS example:

```
RSM_MIX_2D_INITIAL (hRsm, 1, 1, 2, 0, 300, 18000, 500000);
//This function should be defined before the RSM_MIX_2D_CONTINUE()
//function is used. Please refer to the example of this function.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 0A	14
Register[]	**Value (hex)**	**Remarks**			
0	0A 82/0C B2	*Sub_funciton code*			
1	00 01	*axis1 (1 ➔ AXIS_X)*			
2	00 02	*axis2 (2 ➔ AXIS_Y)*			
3	00 00	*nAcc*			
4	00 00	*MSW of VSV*			
5	01 2C	*LSW of VSV (300 = 0x12C)*			
6	00 00	*MSW of VV*			
7	46 50	*LSW of VV (18000 = 0x4650)*			
8	00 07	*MSW of VA*			
9	A1 20	*LSW of VA (500000 = 0x7A120)*			

7.4.9 Execute a Mixed 2-axis motions in Continuous Interpolation (Relative Distance)

eRTU RSM_MIX_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nAcc*, BYTE *nType*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

※ eRTU RSM_MACRO_MIX_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, 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.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***nAcc*:**	**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.**
***nType*:**	**1 → RSM_LINE_2D(BYTE *SlaveAddr*, long *fp1*, long*fp2*) 2 → RSM_ARC_CW(BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*) 3 → RSM_ARC_CCW(BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*) 4 → RSM_CIRCLE_CW(BYTE *SlaveAddr*, long *cp1*, long *cp2*) 5 → RSM_CIRCLE_CCW(BYTE *SlaveAddr*, long *cp1*, long *cp2*)**
***cp1*:**	**It assigns the center point data at axis 1 (-2,000,000,000 ~ +2,000,000,000)**
***cp2*:**	**It assigns the center point data at axis 2 (-2,000,000,000 ~ +2,000,000,000)**
***fp1*:**	**It assigned number of pulses for axis 1 (-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**It assigned number of pulses for axis 2 (-2,000,000,000 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_MIX_2D_CONTINUE is 0A 84.
The Sub_function code of RSM_MACRO_MIX_2D_CONTINUE is 0C 84.

MODBUS example:

RSM_MIX_2D_CONTINUE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 0B	16
Register[]	**Value (hex)**	**Remarks**			
0	0A 84/0C 84	*Sub_funciton code*			
1	00 00	*nAcc*			
2	00 01	*nType*			
3	00 00	*MSW of cp1*			
4	00 00	*LSW of cp1 (for linear motion, set 0)*			
5	00 00	*MSW of cp2*			
6	00 00	*LSW of cp2 (for linear motion, set 0)*			
7	00 00	*MSW of fp1*			
8	00 64	*LSW of fp1 (100 = 0x64)*			
9	00 00	*MSW of fp2*			
10	00 64	*LSW of fp2 (100 = 0x64)*			

7.4.10 Execute a Mixed 2-axis motions in Continuous Interpolation (Absoult Position)

eRTU RSM_ABS_MIX_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nAcc*, BYTE *nType*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

※ eRTU RSM_ABS_MACRO_MIX_2D_CONTINUE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *nAcc*, BYTE *nType*, long *cp1*, long *cp2*, long *fp1*, long *fp2*)

Description:

This function executes mixed linear and circular 2-axis absolute position continuous interpolation motion.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***nAcc*:**	**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.**
***nType*:**	**1 → RSM_ABS_LINE_2D(BYTE *SlaveAddr*, long *fp1*, long*fp2*) 2 → RSM_ABS_ARC_CW(BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*) 3 → RSM_ABS_ARC_CCW(BYTE *SlaveAddr*, long *cp1*, long *cp2*, long *fp1*, long *fp2*) 4 → RSM_ABS_CIRCLE_CW(BYTE *SlaveAddr*, long *cp1*, long *cp2*) 5 → RSM_ABS_CIRCLE_CCW(BYTE *SlaveAddr*, long *cp1*, long *cp2*)**
***cp1*:**	**the absolute center position of axis 1 (-2,000,000,000 ~ +2,000,000,000)**
***cp2*:**	**the absolute center position of axis 2 (-2,000,000,000 ~ +2,000,000,000)**
***fp1*:**	**The absolute end position of axis 1 (-2,000,000,000 ~ +2,000,000,000)**
***fp2*:**	**The absolute end position of axis 2 (-2,000,000,000 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_ABS_MIX_2D_CONTINUE is 0x0AFD.
The Sub_function code of RSM_ABS_MACRO_MIX_2D_CONTINUE is 0x0CFD.

MODBUS example:

RSM_ABS_MIX_2D_CONTINUE (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 0B	16
Register[]	**Value (hex)**	**Remarks**			
0	0A FD/0C FD	*Sub_funciton code*			
1	00 00	*nAcc*			
2	00 01	*nType*			
3	00 00	*MSW of cp1*			
4	00 00	*LSW of cp1 (for linear motion, set 0)*			
5	00 00	*MSW of cp2*			
6	00 00	*LSW of cp2 (for linear motion, set 0)*			
7	00 00	*MSW of fp1*			
8	00 64	*LSW of fp1 (100 = 0x64)*			
9	00 00	*MSW of fp2*			
10	00 64	*LSW of fp2 (100 = 0x64)*			

7.4.11 3-Axis Helical Motion

eRTU RSM_HELIX_3D (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis1*, BYTE *axis2*, BYTE *axis3*, BYTE *nDir*, DWORD *VV* , long *cp1*, long *cp2*, long *cycle*, long *pitch*)

※ eRTU RSM_MACRO_HELIX_3D (HANDLE *hRsm*, BYTE *SlaveAddr*, 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 resources to run this function.

Category:

MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	COM port handle
***SlaveAddr*:**	Modbus RTU Slave Address (valid range: 1 to 247)
***axis1*:**	The first axis (axis 1). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).
***axis2*:**	The second axis (axis 2). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).
***axis3*:**	The third axis (axis 3). Please refer to Table 4. The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).
***nDir*:**	0 → Move in a CW direction. 1 → Move in a CCW direction.
***VV*:**	Vector speed (in PPS)
***cp1*:**	The value of center at axis 1 (-2,000,000,000 ~ +2,000,000,000)
***cp2*:**	The value of center at axis 2 (-2,000,000,000 ~ +2,000,000,000)
***cycle*:**	Number of cycle (-2,000,000,000 ~ +2,000,000,000)
***pitch*:**	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 chapter 2.2)
Use the RSM_GET_ERROR_CODE() function to identify the error.

Remark:

The Sub_function code of RSM_HELIX_3D is 0A 88.
The Sub_function code of RSM_MACRO_HELIX_3D is 0C 88.

MODBUS example:

```
RSM_HELIX_3D (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 0F	1E
Register[]	**Value (hex)**	**Remarks**			
0	0A 88/0C 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)*			

Related example :

```
BYTE SlaveAddr=1; //select module 1.
//===========================================
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU,160000L);
//set maximum speed for all axes to 160K PPS.
long v=50000; //set vector speed to 50K PPS.
RSM_HELIX_3D(hRsm, SlaveAddr, AXIS_Y, AXIS_Z, AXIS_X, 1, v, 0, 1000, 5,
-2000);
//the circular motion is on YZ plane, and the linear motion is
//along the X axis.
//===========================================
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XYZU,160000L);
//set the maximum  speed for all axes to 160K PPS.
long v=100000; //set vector speed to 100K PPS.
RSM_HELIX_3D(hRsm, SlaveAddr, AXIS_Y, AXIS_Z, AXIS_U, 1, v, 0, 25000,
50, 3600);
//the circular motion is on YZ plane, and the linear motion is along
//the U axis.
```

7.4.12 Set Synchronous Line Scan Motion

eRTU RSM_LINE_SCAN (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***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*:**	**Output active logic:** **0 = low active (0V); 1 = high active (5V)**
***PulseWidth*:**	**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 chapter 2.2)

MODBUS example:

```
RSM_LINE_SCAN (hRsm, 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:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 07	0E
Register[]	**Value (hex)**	**Remarks**			
0	0A 90	*Sub_funciton code*			
1	00 01	*axis (1 ➔ AXIS_X)*			
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 = 0xFFFFFFD9)*			

Related example:

```
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XY, 4000000);
//set the maximum speed of X and Y axes to 4M PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_XY, 0);
//set the driving mode to be symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_XY, 2000000);
//set the speed of X and Y axes to 2M PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_XY, 20000000);
//set the acceleration of X and Y axes to 2M PPS/S.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_XY, 2000000);
//set the starting speed of X and Y axes to 2M PPS.
RSM_LINE_SCAN(hRsm, SlaveAddr, 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.
RSM_LINE_SCAN(hRsm, SlaveAddr, AXIS_Y, 0, 0, 0, -79);
//Sets the equal distance Line Scan trigger on axis-Y.
//One trigger will be sent out every 80 (=79+1) pulses.
RSM_LINE_SCAN_START(hRsm, SlaveAddr, AXIS_XY, 0, -4000000);
//Towards the negative direction move 4000K PPS.
```

7.4.13 Start Synchronous Line Scan Motion

eRTU RSM_LINE_SCAN_START (HANDLE *hRsm*, BYTE *SlaveAddr*, 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):
a. All ISR will be stopped.
b. Do not execute the following commands:

- **RSM_READ_bVAR**
- **RSM_READ_VAR**
- **RSM_GET_LP**
- **RSM_GET_EP**
- **RSM_GET_CV**
- **RSM_GET_CA**
- **RSM_GET_DI**
- **RSM_GET_ERROR**
- **RSM_GET_ERROR_CODE**
- **RSM_GET_LATCH**
- **RSM_STOP_WAIT**
- **RSM_CLEAR_STOP**

Category:

MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis number:** **Fixed offset displacement X (1) or Y (2) or XY (3); or unequal offset distance X (1).**
***Type*:**	**0 = equal distance motion (according to LP value)** ***1 = unequal distance motion (according to LP value)** **2 = equal distance motion (according to EP value)** ***3 = unequal distance motion (according to EP value)** ***** reserve**
***Position*:**	**Total pulses, moving distance** **(-2,000,000,000 ~ +2,000,000,000)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_LINE_SCAN_START (hRsm, 1, AXIS_XY, 0, -4000000);

//Towards the negative direction move 4000K PPS.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
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)*			

7.4.14 Get Synchronous Line Scan Motion Status

eRTU RSM_GET_LINE_SCAN_DONE (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE* *LScanState*)

Description:

After calling RSM_LINE_SCAN_START, the RSM_GET_LINE_SCAN_DONE function will determine if the line scan procedure has finished. The line scan done flag will be reset after calling RSM_LINE_SCAN_START.

Category:

MODBUS table；RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***LScanState*:**	**1: line scan process has been completed 0: no interrupt occurred**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
BYTE bState;
RSM_GET_LINE_SCAN_DONE (hRsm, 1, & bState);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01 (Slave Address)	04	00 49 (73)	00 01

MODBUS response:

UID (hex)	FC (hex)	Byte Count(hex)	Register(hex)
01	04	02	00 01 line scan operation has ended

7.5 Stop and Hold Functions

7.5.1 Driving Command Hold

eRTU RSM_DRV_HOLD (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※Δ eRTU RSM_MACRO_DRV_HOLD (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

This command prevents the next motion command from executing but it does not stop the current running command. The held frozen motion command(s) can be activated again by calling RSM_DRV_START(). The purpose of this command is to support the simultaneous start of multi-axis driving. The user may write other commands after the RSM_DRV_HOLD command is issued. The RSM_DRV_START releases the hold command and start executing the motion commands issued after the hold command.

Category:

MODBUS sub_function; RTC, MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

Remark:

The Sub_function code of RSM_DRV_HOLD is 0A B4.
The Sub_function code of RSM_MACRO_DRV_HOLD is 0C B4.

MODBUS example:

```
RSM_DRV_HOLD (hRsm, 1, AXIS_XYU);
//hold the driving command to XYU
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A B4/0C B4	*Sub_funciton code*			
1	00 0B	*axis (B ➔ AXIS_XYU)*			

Related example:

```
BYTE SlaveAddr=1; //select card 1.
RSM_DRV_HOLD(hRsm, SlaveAddr, AXIS_XYU);
//hold the driving command to XYU
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_U, 10000);
//set the maximum speed of U-axis to be 10K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_U, 0);
//set the driving mode to be symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_U, 2000);
//set the speed of U-axis to 2,000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_U,1000);
//set the acceleration of U-axis to 1,000 PPS/S.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_U, 2000);
//set the starting speed to 2,000 PPS.
RSM_SET_AO(hRsm, SlaveAddr, AXIS_U, 9); // set the AO to 9 Pulses.
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XY, 20000);
//set the maximum speed of X and Y axes to 20K PPS.
RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
//set the X-axis as the axis 1 and Y-axis as the axis 2 for a 2-axis
// interpolation.
RSM_VECTOR_SPEED(hRsm, SlaveAddr, 0);
//set constant speed motion.  Therefore, VSV=VV. Only VV is required.
RSM_SET_VV(hRsm, SlaveAddr, 5000);
//set the vector speed for card 1 to 5,000  PPS.
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_U, 5000);
//command U-axis to move  5,000 Pulse.  This command is be held.
RSM_LINE_2D(hRsm, SlaveAddr, 12000, 10000);
//command a linear interpolation    motion on the XY planes. It is held, too.
RSM_DRV_START(hRsm, SlaveAddr, AXIS_XYU);
//release the holding status. X,Y , and U axes will start to move
// simultaneously.
```

7.5.2 Release Holding Status and Start Driving

eRTU RSM_DRV_START (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※Δ eRTU RSM_MACRO_DRV_START (HANDLE *hRsm*, BYTE *SlaveAddr*, 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:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

Remark:

The Sub_function code of RSM_DRV_START is 0A B5.
The Sub_function code of RSM_MACRO_DRV_START is 0C B5.

MODBUS example:

```
RSM_DRV_START (hRsm, 1, AXIS_XYU);
//release the holding status. X,Y , and U axes will start to move
// simultaneously.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A B5/0C B5	*Sub_funciton code*			
1	00 0B	*axis (B ➔ AXIS_XYU)*			

Related example:

```
BYTE SlaveAddr=1; //select card 1.
RSM_DRV_HOLD(hRsm, SlaveAddr, AXIS_XYU);
//hold the driving command to XYU
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_U, 10000);
//set the maximum speed of U-axis to be 10K PPS.
RSM_NORMAL_SPEED(hRsm, SlaveAddr, AXIS_U, 0);
//set the driving mode to be symmetric T-curve.
RSM_SET_V(hRsm, SlaveAddr, AXIS_U, 2000);
//set the speed of U-axis to 2,000 PPS.
RSM_SET_A(hRsm, SlaveAddr, AXIS_U,1000);
//set the acceleration of U-axis to 1,000 PPS/S.
RSM_SET_SV(hRsm, SlaveAddr, AXIS_U, 2000);
//set the starting speed to 2,000 PPS.
RSM_SET_AO(hRsm, SlaveAddr, AXIS_U, 9); // set the AO to 9 Pulses.
RSM_SET_MAX_V(hRsm, SlaveAddr, AXIS_XY, 20000);
//set the maximum speed of X and Y axes to 20K PPS.
RSM_AXIS_ASSIGN(hRsm, SlaveAddr, AXIS_X, AXIS_Y, 0);
//set the X-axis as the axis 1 and Y-axis as the axis 2 for a 2-axis
// interpolation.
RSM_VECTOR_SPEED(hRsm, SlaveAddr, 0);
//set constant speed motion.  Therefore, VSV=VV. Only VV is required.
RSM_SET_VV(hRsm, SlaveAddr, 5000);
//set the vector speed for card 1 to 5,000  PPS.
RSM_FIXED_MOVE(hRsm, SlaveAddr, AXIS_U, 5000);
//command U-axis to move  5,000 Pulse.  This command is be held.
RSM_LINE_2D(hRsm, SlaveAddr, 12000, 10000);
//command a linear interpolation    motion on the XY planes. It is held, too.
RSM_DRV_START(hRsm, SlaveAddr, AXIS_XYU);
//release the holding status. X,Y , and U axes will start to move
// simultaneously.
```

7.5.3 Decelerate and Finally Stop Axes

eRTU RSM_STOP_SLOWLY (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_STOP_SLOWLY (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

This function commands to decelerate and finally stops the assigned axes slowly.

Category:

MODBUS sub_function; RTC, and MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

Remark:

The Sub_function code of RSM_STOP_SLOWLY is 0A B7.
The Sub_function code of RSM_MACRO_STOP_SLOWLY is 0C B7.

MODBUS example:

```
RSM_STOP_SLOWLY (hRsm, 1, AXIS_XY);
//decelerate and stop the X and Y axes
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A B7/0C B7	*Sub_funciton code*			
1	00 03	*axis (3 ➔ AXIS_XY)*			

7.5.4 Immediately Stop Axes

eRTU RSM_STOP_SUDDENLY (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_STOP_SUDDENLY (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

This function commands to immediately stop the assigned axes.

Category:

MODBUS sub_function; RTC, and MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

Remark:

The Sub_function code of RSM_STOP_SUDDENLY is 0A B8.
The Sub_function code of RSM_MACRO_STOP_SUDDENLY is 0C B8.

MODBUS example:

```
RSM_STOP_SUDDENLY (hRsm, 1, AXIS_ZU);
//immediately stop the Z and U axes.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A B8/0C B8	*Sub_funciton code*			
1	00 0C	*axis (C ➔ AXIS_ZU)*			

7.5.5 Decelerate and Finally Stop Interpolation Motion

eRTU RSM_VSTOP_SLOWLY (HANDLE *hRsm*, BYTE *SlaveAddr*)

※ eRTU RSM_MACRO_VSTOP_SLOWLY (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function commands to stop interpolation motion of the assigned module in a decelerating way.

Category:

MODBUS sub_function; RTC, and MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_VSTOP_SLOWLY is 0A B9.
The Sub_function code of RSM_MACRO_VSTOP_SLOWLY is 0C B9.

MODBUS example:

RSM_VSTOP_SLOWLY (hRsm, 1);
//stop the interpolation of card 1 in a decelerating way.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A B9/0C B9	***Sub_funciton code***			

7.5.6 Immediately Stop Interpolation Motion

eRTU RSM_VSTOP_SUDDENLY (HANDLE *hRsm*, BYTE *SlaveAddr*)

※ eRTU RSM_MACRO_VSTOP_SUDDENLY (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function commands to stop interpolation motion of the assigned module immediately.

Category:

MODBUS sub_function; RTC, and MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_VSTOP_SUDDENLY is 0A BA.
The Sub_function code of RSM_MACRO_VSTOP_SUDDENLY is 0C BA.

MODBUS example:

```
RSM_VSTOP_SUDDENLY (hRsm, 1);
// stop the interpolation of card 1 immediately.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A BA/0C BA	***Sub_funciton code***			

7.5.7 Clear Axes Stop Status

eRTU RSM_CLEAR_STOP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_CLEAR_STOP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

After using the stop functions "RSM_STOP_SLOWLY" or "RSM_STOP_SUDDENLY", please solve the malfunction, and then issue this function to clear the stop status.

Category:

MODBUS sub_function; RTC, and MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

Remark:

The Sub_function code of RSM_CLEAR_STOP is 0A BB.
The Sub_function code of RSM_MACRO_CLEAR_STOP is 0C BB.

MODBUS example:

```
RSM_CLEAR_STOP (hRsm, 1, AXIS_ZU);
//clear the error status of zu axes.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A BB/0C BB	*Sub_funciton code*			
1	00 0C	*axis (C ➔ AXIS_ZU)*			

7.5.8 Clear Interpolation Motion Stop Status

eRTU RSM_CLEAR_VSTOP (HANDLE *hRsm*, BYTE *SlaveAddr*)

※ eRTU RSM_MACRO_CLEAR_VSTOP (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

After using the stop functions "RSM_VSTOP_SLOWLY" or "RSM_VSTOP_SUDDENLY", please solve the malfunction, and then issue this function to clear the stop status.

Category:

MODBUS sub_function; RTC, and MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_CLEAR_VSTOP is 0A 09.
The Sub_function code of RSM_MACRO_CLEAR_VSTOP is 0C 09.

MODBUS example:

```
RSM_CLEAR_VSTOP (hRsm, 1);
//clear the error status of card 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A 09/0C 09	***Sub_funciton code***			

7.5.9 End of Interpolation

eRTU RSM_INTP_END (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *type*)

※ eRTU RSM_MACRO_INTP_END (HANDLE *hRsm*, BYTE *SlaveAddr*, 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 RSM_INTP_END() function.**

Category:

MODBUS sub_function; RTC, and MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***type*:**	**0 → 2-axis interpolation 1 → 3-axis interpolation**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_INTP_END is 0A BC.
The Sub_function code of RSM_MACRO_INTP_END is 0C BC.

MODBUS example:

```
RSM_INTP_END (hRsm, 1, 0);
//declear the end of a 2-axis interpolation on card 1.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A BC/0C BC	***Sub_funciton code***			
1	00 00	***type (0 → 2-axis interpolation)***			

7.5.10 Emergency Stop

eRTU RSM_EMERGENCY_STOP (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *stopType*)

Description:

Stop all the axes and clear all data in i-8094H command buffer.

Category:

MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***stopType*:**	**Stop type: 0: immediately stop; 1: deceleration stop**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
RSM_EMERGENCY_STOP (hRsm, 1, 0);
// Stop all the axes immediately and clear all data in the i-8094H command
//buffer.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A 04	***Sub_function code***			
1	00 00	***stopType***			

7.5.11 Clear Emergency Stop

eRTU RSM_CLEAR_EMERGENCY_STOP (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

After using the stop functions RSM_EMERGENCY_STOP, please solve the malfunction, and then issue this function to clear the stop status.

Category:

MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_CLEAR_EMERGENCY_STOP (hRsm, 1);
//clear the error status of card 1.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A 05	***Sub_function code***			

8 Initial Parameter Table

The initial table stores the basic settings of the motion control chip in a non-volatile memory. The initialization table includes the following setting (Figure 3):

- **pulse output signal**
- **encoder setting**
- **hardware trigger level**
- **software limits**

Function	Parameter	X-Axis	Y-Axis	Z-Axis	U-Axis
Pulse Output Signal	Pulse Output Mode	0	0	0	0
Max Pulse Output Rate	Data (8000 to 4,000,000 PPS)	8000	8000	8000	8000
Hardware Limit Switch (HLMT)	Active Level (forward)	Low Active	Low Active	Low Active	Low Active
	Active Level (reverse)	Low Active	Low Active	Low Active	Low Active
Hardware Limit Stop Mode	Stop Mode	Abrupt Stop	Abrupt Stop	Abrupt Stop	Abrupt Stop
Near Home Sensor	Trigger Level	Low Active	Low Active	Low Active	Low Active
Home Sensor	Trigger Level	Low Active	Low Active	Low Active	Low Active
Software Limit	Enable Software Limit	Disable	Disable	Disable	Disable
	Software Limit (forward)	100000	100000	100000	100000
	Software Limit (reverse)	-100000	-100000	-100000	-100000
	Position Counter Type	Logic Pos	Logic Pos	Logic Pos	Logic Pos
Set Encoder Parameters	Encoder Input Type	A Quad B	A Quad B	A Quad B	A Quad B
	A Quad B Input Signal Division	1/1	1/1	1/1	1/1
	Trigger Level for Z Phase	Low Active	Low Active	Low Active	Low Active
Servo Driver Setting	On/Off	Off	Off	Off	Off
Servo Alarm Setting	Enable Servo Alarm	Disable	Disable	Disable	Disable
	Trigger Level	Low Active	Low Active	Low Active	Low Active
In-Position Signal	Enable In-Position Input	Disable	Disable	Disable	Disable
	Trigger Level	Low Active	Low Active	Low Active	Low Active
Digital Filter	Input Ports	1	1	1	1
	Filter Time Constant	2	2	2	2
Variable Ring Position Counter	Enable Variable Ring Counter	Disable	Disable	Disable	Disable
	Maximum Value	10000	10000	10000	10000
Triangle Driving Profile Prevention	Enable Triangle Prevention	Disable	Disable	Disable	Disable

Figure 3: Initial table

The factory default settings of the initial table are as follows:

RSM_SET_PULSE_MODE(hRsm, *SlaveAddr*, AXIS_XYZU, 0);
RSM_SET_MAX_V(hRsm, *SlaveAddr*, AXIS_XYZU, 2000000L);
RSM_SET_HLMT(hRsm, *SlaveAddr*, AXIS_XYZU, 0, 0);
RSM_LIMITSTOP_MODE(hRsm, *SlaveAddr*, AXIS_XYZU, 0);
RSM_SET_NHOME(hRsm, *SlaveAddr*, AXIS_XYZU, 0);

RSM_SET_HOME_EDGE(hRsm, *SlaveAddr*, AXIS_XYZU, 0);
RSM_CLEAR_SLMT(hRsm, *SlaveAddr*, AXIS_XYZU);
RSM_SET_ENCODER(hRsm, *SlaveAddr*, AXIS_XYZU, 0, 0, 0);
RSM_SERVO_OFF(hRsm, *SlaveAddr*, AXIS_XYZU);
RSM_SET_ALARM(hRsm, *SlaveAddr*, AXIS_XYZU, 0, 0);
RSM_SET_INPOS(hRsm, *SlaveAddr*, AXIS_XYZU, 0, 0);
RSM_SET_FILTER(hRsm, *SlaveAddr*, AXIS_XYZU, 0, 0);
RSM_VRING_DISABLE(hRsm, *SlaveAddr*, AXIS_XYZU);
RSM_AVTRI_DISABLE(hRsm, *SlaveAddr*, AXIS_XYZU);
RSM_EXD_DISABLE(hRsm, *SlaveAddr*, AXIS_XYZU);

Modify the default initial table by calling the above function with the required settings. The settings may also be changed after start up or later on during the control operations. Always the last setting will be retained. Set the initialization table according to the specific environment in which the RS-M8194H is being used by calling the above functions.

8.1 Calling the Initial Table

eRTU RSM_LOAD_INITIAL (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

In order to initialize the basic hardware and software settings of the RS-M8194, it is required to call the RSM_LOAD_INITIAL function. Directly after power on, the initial table is being automatically called by executing the RSM_LOAD_INITIAL function. The Modbus master is also allowed to call this function at any time.

Category:

MODBUS sub_function; RTC

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
RSM_LOAD_INITIAL (hRsm, 1);
// load the initial setting values of the parameter table into i-8094H
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A C8	***Sub_funciton code***			

Related example:

```
RSM_SET_PULSE_MODE(hRsm, 1, INITIAL_XYZU, 0);
//set the pulse mode of X, Y, Z, and U axes as 0, write into the parameter
//table in module 1.
RSM_SET_MAX_V(hRsm, 1, INITIAL_XY, 200000L);
```

```
//The maximum speed for the X and Y axes of module 1 is 200KPPS.
//Write into the parameter table.
RSM_SET_HLMT(hRsm, 1, INITIAL_XYZU, 0, 0);
//set all the trigger levels as low-active for all limit switches
//on module 1. Write into the parameter table.
RSM_LIMITSTOP_MODE(hRsm, 1, INITIAL_X, 0);
//set X axis to stop immediately if any limit switch on X axis is triggered
//on module 1. Write into the parameter table.
RSM_SET_NHOME(hRsm, 1, INITIAL_XY, 0);
//set the trigger level of NHOME of X and Y axes on module 1 to be
//active low. Write into the parameter table.
RSM_ LOAD_INITIAL(hRsm, 1);
// load the initial setting values of the parameter table into i-8094H
```

9 Macro Programming

Introduction

Macro programs are written to the nonvolatile memory of the i-8094H module. The nonvolatile memory can hold Macro program of different sizes. Two types of Macro tables are provided:

- **MP tables: These tables hold normal motion control Macro commands. MP commands in these tables are being called by the command RSM_MP_CALL or RSM_MACRO_MP_CALL. The RSM_MP_CALL command can be directly called by the host controller, and the RSM_MACRO_MP_CALL is called within a MP table.**
- **ISR tables: ISR in these tables will be called in case of an interrupt generated by the motion control chip. Not all Macro commands can be added to the ISR table in order to guaranty that the execution of ISR is finished in a short time interval.**

The maximum of command lines provided by each Macro table is shown in the following figure (Figure 4).

Macro Type	Number of Macros	Number of Command Lines	Macro Names								
Interrupt Service Routine (ISR)	6	8	ISR1	ISR2	ISR3	ISR4	ISR5	ISR6			
	9	16	ISR7	ISR8	ISR9	ISR10	ISR11	ISR12	ISR13	ISR14	ISR15
	3	32	ISR16	ISR17	ISR18						
	2	64	ISR19	ISR20							
Motion Program (MP)	40	8	MP1	~	MP40						
	50	16	MP41	~	MP90						
	40	32	MP91	~	MP130						
	20	64	MP131	~	MP150						
	5	128	MP151	MP152	MP153	MP154	MP155				
	2	512	MP156	MP157							

Figure 4: Macro table definitions

9.1 Create MP Macro Program Codes

9.1.1 Start a MP Macro Program Codes Programming

eRTU RSM_MP_CREATE(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *mp_No*)

Description:

Each Macro program MP has to start with an RSM_MP_CREATE command and the end of the MP have to be indicated with an RSM_MACRO_MP_CLOSE command.

Category:

MODBUS sub_function; RTC

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
mpNo:	**The following variables are supported: Macro Program number (MP1 ~ MP157)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
RSM_MP_CREATE (hRsm, 1, MP21);
//Write Macro Program into i-8094H.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A C9	***Sub_funciton code***			
1	00 15	***mpNo (MP21 = 21 = 0x15)***			

Related example:

```
RSM_MP_CREATE(hRsm, 1, MP21); //Write #21 Macro Program into i-8094H.
 //=======================================================
 //The following functions will not be executed, but will be written into
 //i-8094H for further execution.
 RSM_MACRO_SET_MAX_V(hRsm, 1, AXIS_XYZU, 20000);
 //The maximum speed of all axes is 20K PPS.
 RSM_MACRO_NORMAL_SPEED(hRsm, 1, AXIS_XYZU, 0);
 //Set symmetric T-curve for XYZU axes on module.
 RSM_MACRO_SET_V(hRsm, 1, AXIS_XYZU, 2000);
 //set the speed of all axes on module 1 to be 2000 PPS.
 RSM_MACRO_SET_A(hRsm, 1, AXIS_XYZU, 1000);
 //set the acceleration of XYZU axes to be 1000 PPS/sec
 RSM_MACRO_SET_SV(hRsm, 1, AXIS_XYZU, 2000);
 //set the start speed of XYZU axes to be 2000 PPS.
 RSM_MACRO_SET_AO(hRsm,1, AXIS_XYZU, 9);
 //set the number of remaining offset pulses for XYZU axes to be 9 pulses.
 RSM_MACRO_FIXED_MOVE(hRsm, 1, AXIS_XYZU, 10000);
 //move XYZU axes to be 10000 pulses.
 RSM_MACRO_MP_CLOSE(hRsm, 1);
 // module 1 finish, write Macro Program into i-8094H
 //=======================================================
```

9.1.2 End a MP Macro Program Codes Programming

※ **eRTU RSM_MACRO_MP_CLOSE (HANDLE *hRsm*, BYTE *SlaveAddr*)**

Description:

Indicate the end of a MP Macro program. This command is only used together with RSM_MP_CREATE to define an MP program. Otherwise, it will be ignored.

Category:

MODBUS sub_function; MP

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_MP_CLOSE (hRsm, 1);
// module 1 finish, write Macro Program into i-8094H

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C CA	***Sub_funciton code***			

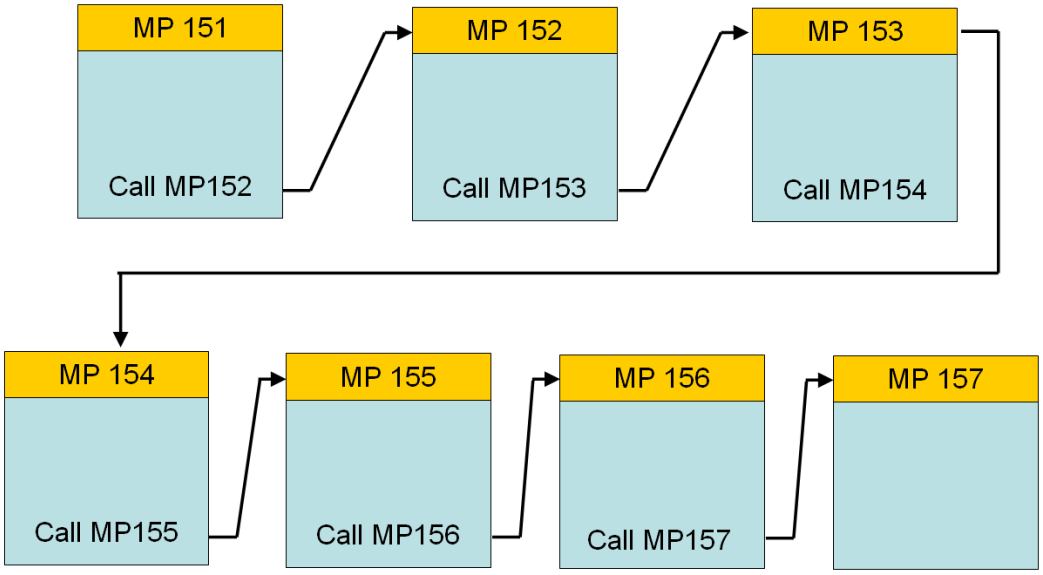
9.1.3 MP Macro Program Execution

eRTU RSM_MP_CALL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *mpNo*)

※ eRTU RSM_MACRO_MP_CALL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *mpNo*)

Description:

Calls and execute a specific Macro Program stored in the non-volatile memory of the i-8094H. Within an MP program, up to 6 nested MP_CALL function calls are allowed. For MP, the maximum nested Macro Program call can be up to 7. For ISR, no nested program call is allowed.



Category:

MODBUS table, MODBUS sub_function; RTC and MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
mpNo:	**The following variables are supported: Macro Program number (MP1 ~ MP157)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_MP_CALL is 0A CB.
The Sub_function code of RSM_MACRO_MP_CALL is 0C CB.

MODBUS example:

```
//The true value of MPn = n; therefore, MP21 = 21 = 0x15.
RSM_MP_CALL (hRsm, 1, MP21); // Execute the MP21 on i-8094H
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A CB/0C CB	*Sub_funciton code*			
1	00 15	*mpNo (MP21 = 21 = 0x15)*			

The other method to call MPn is writing the MPn to the pre-defined holding register 8.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	00 08	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	15	*mpNo (MP21 = 21 = 0x15)*			

Related example:

```
RSM_MP_CREATE(hRsm, 1, MP21); //Write #21 Macro Program into i-8094H.
//=======================================================
//The following functions will not be executed, but will be written into
//i-8094H for further execution.
RSM_MACRO_SET_MAX_V(hRsm, 1, AXIS_XYZU, 20000);
//The maximum speedof the axis is 20K PPS.
RSM_MACRO_NORMAL_SPEED(hRsm, 1, AXIS_XYZU, 0);
//Set symmetric T-curve for XYZU axes on module.
RSM_MACRO_SET_V(hRsm, 1, AXIS_XYZU, 2000);
//set the speed of all axes on module 1 to be 2000 PPS.
RSM_MACRO_SET_A(hRsm, 1, AXIS_XYZU, 1000);
//set the acceleration of XYZU axes to be 1000 PPS/sec.
RSM_MACRO_SET_SV(hRsm, 1, AXIS_XYZU, 2000);
//set the start speed of XYZU axes to be 2000 PPS.
RSM_MACRO_SET_AO(hRsm, 1, AXIS_XYZU, 9);
```

```
//set the number of remaining offset pulses for XYZU axes to be 9 pulses.
RSM_MACRO_FIXED_MOVE(hRsm, 1, AXIS_XYZU, 10000);
//move XYZU axes to be 10000 pulses.
RSM_MACRO_MP_CLOSE(hRsm, 1);
// module 1 finish, write Macro Program into i-8094H
//=====================================================
RSM_MP_CALL(hRsm, 1, MP21); // Execute the MP21 on i-8094H
```

9.2 Create ISR Macro Program Codes

9.2.1 Start a ISR Macro Program Codes Programming

eRTU RSM_MP_ISR_CREATE(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *isr_No*)

Description:

ISR Macro table are being called by hardware interrupts generated by the motion control chip (e.g. exceeding harware or software limit, driving has finished or when a compare condition has been met). To ensure that the API in an ISR are executed completely in a short time, no nested Macro calls, MP_FOR loop and MP_STOP_WAIT are allowed inside an ISR.

Each ISR program has to be created by calling an RSM_MP_ISR_CREATE command and ended by an RSM_MACRO_MP_ISR_CLOSE command.

Category:

MODBUS sub_function; RTC

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
isr_No:	The following variables are supported: ISR Macro Program number (ISR1 ~ ISR20)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MP_ISR_CREATE (hRsm, 1, ISR1);
//Write ISR Macro program into i-8094H.

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0ACD	*Sub_funciton code*			
1	00 01	*isr_No (ISR1 = 1 = 0x01)*			

9.2.2 End a ISR Macro Program Codes Programming

Δ eRTU RSM_MACRO_MP_ISR_CLOSE (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function indicates the end of an ISR program. Every ISR program must contain this command as the last command to indicate the end of the ISR program.

Category:

MODBUS sub_function; ISR

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_MP_ISR_CLOSE (hRsm, 1);
// finish the writing of ISR Macro Program for i-8094H in slot 1

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C CE	***Sub_funciton code***			

9.2.3 ISR Maro Program (ISR) Execution

eRTU RSM_MP_ISR_CALL (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *isrNo*)

Description:

This function is used for calling an ISR Macro program. Usually, ISR Macro tables are called by hardware interrupts. The purpose of this function is to test an ISR program.

Category:

MODBUS table, MODBUS sub_function; ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***isrNo*:**	**The following variables are supported: ISR Macro Program number (ISR1 ~ ISR20)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
//The true value of ISRn = n; therefore, ISR1 = 1 = 0x1.
RSM_MP_ISR_CALL (hRsm, 1, ISR1);      // Execute the ISR1
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A CF	***Sub_funciton code***			
1	00 01	***isrNo (ISR1 = 1 = 0x1)***			

9.3 User Defined Variables

9.3.1 Assign Macro variable a value

※Δ eRTU RSM_MACRO_SET_VAR (HANDLE *hRsm*, BYTE *SlaveAddr*, long *varNo*, long *data*)

Description:

Assign a Macro variable VARn (n = 0, 1, 2 ...) an integer value. It is also possible to assign the variable a value from another variable. This API acts as an assignement operator similar to the following statments:
VARn = VARm
VARn = Value

Category:

MODBUS table, MODBUS sub_function; MP and ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
varNo:	The following variables are supported: MACRO variable (VAR0 ~ VAR511)
data:	The following inputs are supported: Value (-2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
//The address of VARn = 0x7FFF0000 + n; therefore, the address of VAR1 is
// 0x7FFF0001.
RSM_MACRO_SET_VAR(hRsm, 1, VAR1, 100); // VAR1 = 100
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 05	0A
Register[]	**Value (hex)**	**Remarks**			
0	0C D2	*Sub_funciton code*			
1	7F FF	*MSW address of varNo*			
2	00 01	*LSW address of varNo*			
3	00 00	*MSW of data*			
4	00 64	*LSW of data (100 = 0x64)*			

9.3.2 Get Command Return Value

※Δ eRTU RSM_MACRO_SET_RVAR (HANDLE *hRsm*, BYTE *SlaveAddr*, long *varNo*)

Description:

Assign the return value of the preceding function to a Macro variable VARn (n = 0, 1, 2 ...), where VARn = return value of preceding function (the value will be automatically casted to (long).

The command reads the return values of the following commands:

- RSM_MACRO_GET_LP
- RSM_ MACRO_GET_EP
- RSM_MACRO_ABS_GET_POSITION
- RSM_MACRO_GET_HOME_SEARCH_STATE
- RSM_ MACRO_GET_ERROR
- RSM_ MACRO_GET_ERROR_CODE
- RSM_ MACRO_FRNET_IN
- RSM_MACRO_FRNET_READ
- RSM_ MACRO_GET_LATCH
- RSM_ MACRO_GET_DI_ALL
- RSM_ MACRO_GET_DI
- RSM_MACRO_GET_DI_SIGNAL

Category:

MODBUS sub_function; MP and ISR.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
varNo:	The following variables are supported: MACRO variable (VAR0 ~ VAR511)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
// The address of VARn = 0x7FFF0000 + n; therefore, the address of VAR5
// is 0x7FFF0005.
RSM_MACRO_SET_RVAR(hRsm, 1, VAR5);
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0C D3	*Sub_funciton code*			
1	7F FF	*MSW address of varNo*			
2	00 05	*LSW address of varNo*			

Related example:

```
//The following commands show how to use Macro commands to store the
//logic position in a variable.
RSM_MP_CREATE(hRsm, 1, MP1); //Write #1 Macro Program into i-8094H.
RSM_MACRO_GET_LP(hRsm, 1, AXIS_X);
//Reads the LP value of the X axis on module 1.
RSM_MACRO_SET_RVAR(hRsm, 1, VAR5);
//Assign the return value of RSM_MACRO_GET_LP to VAR5
//( VAR5 = LP value)
RSM_MACRO_MP_CLOSE(hRsm, 1);
// module 1 finish, write Macro Program   into i-8094H
```

9.4 Simple Calculations

※Δ eRTU RSM_MACRO_VAR_CALCULATE (HANDLE *hRsm*, BYTE *SlaveAddr*, long *varNo*, BYTE *Operator*, long *varNo1*, long *varNo2*)

Description:

Basic two parameter (value) calculations:

varNo **+**	***varNo1***	**=**	***varNo2***
varNo **-**	***varNo1***	**=**	***varNo2***
varNo *****	***varNo1***	**=**	***varNo2***
varNo **/**	***varNo1***	**=**	***varNo2***
varNo **&**	***varNo1***	**=**	***varNo2***
varNo **|**	***varNo1***	**=**	***varNo2***

Category:

MODBUS sub_function; MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***varNo*:**	**The first operand for this calculation. The following operand inputs are supported: Value (-2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511)**
***Operator*:**	**'+' addition '—' subtraction '∗' multiplication '／' division(rounding down to the nearest integer) '&' AND operator '|' OR operator**
***varNo1*:**	**The second operand of the calculation. The following operand inputs are supported: Value (-2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511)**
***varNo2*:**	**MACRO variable (VAR0 ~ VAR511) which stores the result of the calculation**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

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
'&'	38	26
'|'	124	7C

```
RSM_MACRO_VAR_CALCULATE (hRsm, 1, VAR1, '+', VAR2, VAR3);
//VAR1 + VAR2 = VAR3
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 08	10
Register[]	**Value (hex)**	**Remarks**			
0	0C D4	*Sub_funciton code*			
1	7F FF	*MSW address of varNo*			
2	00 01	*LSW address of varNo*			
3	00 2B	*Operator : ASCII code of '+'*			
4	7F FF	*MSW address of varNo1*			
5	00 02	*LSW address of varNo1*			
6	7F FF	*MSW address of varNo2*			
7	00 03	*LSW address of varNo2*			

Related example:

```
RSM_MP_CREATE(hRsm, 1, MP100); // Write #100 Macro Program into i-8094H.
//=========================================================
//The following functions will not be executed, but will be written into
// i-8094H for further execution.
RSM_MACRO_SET_VAR(hRsm, 1, VAR1, 100); //VAR1 = 100
RSM_MACRO_SET_VAR (hRsm, 1, VAR2, 200); //VAR2 = 200
RSM_MACRO_SET_VAR (hRsm, 1, VAR10, 10); //VAR10 = 10
//---------------------------------------------------------------------------------------------------
RSM_MACRO_FRNET_IN(hRsm, 1, 8);
RSM_MACRO_SET_RVAR(hRsm, 1, VAR6);
//VAR6 = current input of RA8 on   module 1
//---------------------------------------------------------------------------------------------------
```

```
RSM_MACRO_GET_LP(hRsm, 1, AXIS_X);
//Reads the LP value of the X axis on module 1.
RSM_MACRO_SET_RVAR(hRsm, 1, VAR5);
//VAR5 = LP value of the X axis on module 1.
//-------------------------------------------------------------------------------------------------
RSM_MACRO_VAR_CALCULATE(hRsm, 1, VAR1, '+', VAR2, VAR3);
//VAR1 + VAR2 = VAR3
RSM_MACRO_VAR_CALCULATE (hRsm, 1, VAR3, '-', VAR1, VAR3);
//VAR3 – VAR1 = VAR3
RSM_MACRO_VAR_CALCULATE (hRsm, 1, VAR3, '*', VAR10, VAR2);
//VAR3 x VAR10 = VAR2
RSM_MACRO_VAR_CALCULATE (hRsm, 1, VAR3, '/', VAR2, VAR1);
//VAR3 / VAR2 = VAR1
//Results: VAR1 = 10; VAR2 = 200; VAR3 = 2,000; VAR10 = 10
RSM_MACRO_MP_CLOSE(hRsm, 1);
// module 1 finish, write #100 Macro Program into i-8094H
//=======================================================
```

9.5 Command Loop (FOR~NEXT)

9.5.1 Start of FOR Loop Block

※ eRTU RSM_MACRO_FOR (HANDLE *hRsm*, BYTE *SlaveAddr*, long *varNo*)

Description:

The *varNo* specifies how many times the code between the RSM_MACRO_FOR command and the RSM_MACRO_NEXT command will be executed. The *varNo* parameter will be decreased until it reaches zero, and then the loop will be stopped. If the *varNo* is assigned to be a Macro VARn, this value of VARn represents the loop cycles and will be changed each time a loop is finished. An RSM_MACRO_EXIT_FOR command can abort the current loop immediately if it is executed inside the loop. Up to six nested loops are supported: one outer loop and six inner loops.

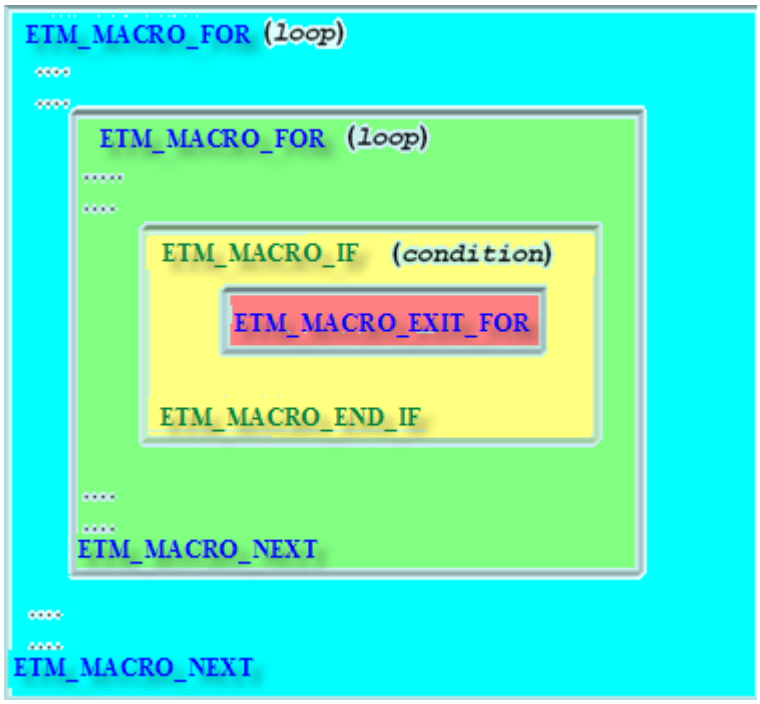


Figure 5: Basic loop application

Category:

MODBUS sub_function; MP.

Parameters:

Paramters	Description
hRsm:	COM port handle
SlaveAddr:	Modbus RTU Slave Address (valid range: 1 to 247)
IValue:	Specifies the cycle number. The following inputs are supported: Value (-2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511)

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
// The address of VARn = 0x7FFF0000 + n;
For example, the address of VAR1 = 0x7FFF0001.
RSM_MACRO_FOR (hRsm, 1, VAR1);
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0C D5	*Sub_funciton code*			
1	7F FF	*MSW address of IValue*			
2	00 01	*LSW address of IValue*			

Related example:

```
RSM_MP_CREATE(hRsm, 1, MP100); // Write #100 Macro Program into i-8094H.
//=======================================================
// The following functions will not be executed, but will be written into
// i-8094H for further execution.
RSM_MACRO_SET_MAX_V(hRsm, 1, AXIS_X, 20000);
//The maximum speed of the axis X is 20K PPS.
RSM_MACRO_NORMAL_SPEED(hRsm, 1, AXIS_X, 0);
//Set symmetric T-curve for axis X
RSM_MACRO_SET_V(hRsm, 1, AXIS_X, 2000);
//set the speed of axis X to be 2000 PPS.
RSM_MACRO_SET_A(hRsm, 1, AXIS_X, 1000);
//set the acceleration of axis X to be 1000 PPS/Sec
RSM_MACRO_SET_SV(hRsm, 1, AXIS_X, 2000);
//set the start speed of axis X to be 2000 PPS.
RSM_MACRO_SET_AO(hRsm, 1, AXIS_X, 0);
```

```
//set the number of remaining offset pulses for axis X to be 0 PPS.
//-------------------------------------------------------------------------------------------------
RSM_MACRO_SET_VAR(hRsm, 1, VAR1, 100); //VAR1 = 100。
RSM_MACRO_FOR(hRsm, 1, VAR1);
//enable axis X to move back and forward 1,000 Pulse, loop for 100 times.
//or input the instant operating value RSM_MACRO_FOR(hRsm, 1, 100).
RSM_MACRO_FIXED_MOVE(hRsm, 1, AXIS_X, 1000);
//move axis X to be 1000 pulses.
RSM_MACRO_FIXED_MOVE(hRsm, 1, AXIS_X, -1000);
//move axis X to be -1000 pulses.
RSM_MACRO_NEXT(hRsm, 1);
RSM_MACRO_MP_CLOSE(hRsm, 1);
// module 1 finish, Macro Program is written into i-8094H
//=======================================================
RSM_MP_CALL(hRsm, 1, MP100);
// call module 1 i-8094H, execute Macro Program #100.
```

9.5.2 End of FOR Loop Block

※ **eRTU RSM_MACRO_NEXT (HANDLE *hRsm*, BYTE *SlaveAddr*)**

Description:

This function indicates the end of the RSM_MACRO_FOR loop. Every loop has to start with RSM_MACRO_FOR statement and end with RSM_MACRO_NEXT statement.

Category:

MODBUS sub_function; MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_NEXT (hRsm, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C D6	***Sub_funciton code***			

9.5.3 FOR Loop Block Exit

※ **eRTU RSM_MACRO_EXIT_FOR(HANDLE *hRsm*, BYTE *SlaveAddr*)**

Description:

An RSM_MACRO_EXIT_FOR command can abort the current loop immediately if it is executed inside the loop.

Category:

MODBUS sub_function; MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_EXIT_FOR (hRsm, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C DE	***Sub_funciton code***			

9.6 Conditional Command (IF~ELSE)

9.6.1 IF Condition

※Δ eRTU RSM_MACRO_IF (HANDLE *hRsm*, BYTE *SlaveAddr*, long *varNo*, WORD *Operator*, long *varNo1*)

Description:

A conditional checking RSM_MACRO_IF has to end the whole block with an RSM_MACRO_END_IF statement. One RSM_MACRO_ELSE statement can be added inside the conditional block and will be executed if the RSM_MACRO_IF statement turns out to be false. Up to 6 nested IF statements are supported (one outer and six inner IF conditions).

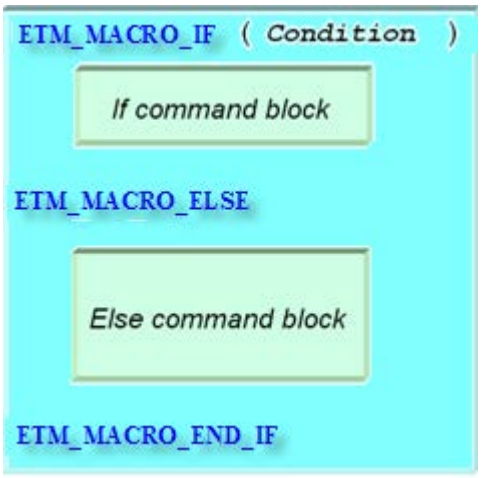


Figure 6: RSM_MACRO_IF application structure

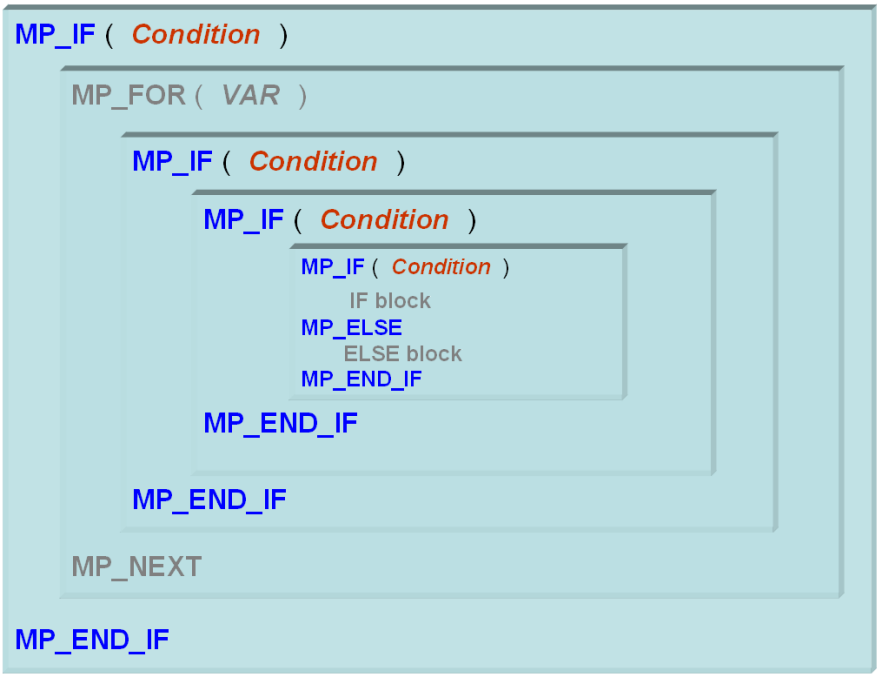


Figure 7: Nested IF statements

Category:

MODBUS sub_function; MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***varNo*:**	**The first variable of the comparison expressions. Can either be a variable of long type (range: -2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511)**
***Operator*:**	**Comparison operators:** **string / constant / Description** **"<" LT Less than (ASCII code = 0x003C)** **">" GT Greater than (ASCII code = 0x003E)** **"<=" LE Less than or equal to (ASCII code = 0x3D3C)** **">=" GE Greater than or equal to (ASCII code = 0x3D3E)** **"==" EQ Equal to (ASCII code = 0x3D3D)** **"!=" NE Not equal to (ASCII code = 0x3D21)**

varNo1:	**The second variable of the comparison expressions. Can either be a variable of long type (range: -2,000,000,000 ~ +2,000,000,000) or MACRO variable (VAR0 ~ VAR511)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

The ASCII code of operator.

Operator	ASCII (dec)	ASCII (hex)
<	60	003C
<=	15676	3D3C
>	62	003E
>=	15678	3D3E
==	15677	3D3D
=	61	003D
!=	15649	3D21
!	33	0021

```
RSM_MACRO_IF (hRsm, 1, VAR1, ”<=”, VAR2);
//The double quotes (“”) is required for Operator parameter.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 06	0C
Register[]	**Value (hex)**	**Remarks**			
0	0C D7	*Sub_funciton code*			
1	7F FF	*MSW address of varNo*			
2	00 01	*LSW address of varNo*			
3	3C 3D	*Operator (0x3C3D ➔ “<=”)*			
4	7F FF	*MSW address of varNo1*			
5	00 02	*LSW address of varNo1*			

Related example:

```
RSM_MP_CREATE(hRsm, 1, MP100); // Write #100 Macro Program into i-8094H.
//=======================================================
// The following functions will not be executed, but will be written into
// i-8094H for further execution.
RSM_MACRO_SET_MAX_V(hRsm, 1, AXIS_X, 20000);
//The maximum speed of the axis X is 20K PPS.
```

```
RSM_MACRO_NORMAL_SPEED(hRsm, 1, AXIS_X, 0);
//Set symmetric T-curve for axis X.
RSM_MACRO_SET_V(hRsm, 1, AXIS_X, 2000);
//set the speed of axis X to be 2000 PPS.
RSM_MACRO_SET_A(hRsm, 1, AXIS_X, 1000);
//set the acceleration of axis X to   be 1000 PPS/sec
RSM_MACRO_SET_SV(hRsm, 1, AXIS_X, 2000);
//set the start speed of axis X to be 2000 PPS.
RSM_MACRO_SET_AO(hRsm, 1, AXIS_X, 0);
//set the number of remaining offset pulses for axis X to be 0 PPS.
//-----------------------------------------------------------------------------------------------
RSM_MACRO_SET_VAR(hRsm, 1, VAR1, 100); //VAR1 = 100。
RSM_MACRO_SET_VAR(hRsm, 1, VAR2, 200); //VAR2 = 200。
RSM_MACRO_IF(hRsm, 1, VAR1, “<”, VAR2);
RSM_MACRO_FIXED_MOVE(hRsm, 1, AXIS_X, 1000);
//command to move axis X for 1000 pulses.
RSM_MACRO_ELSE(hRsm, 1);
RSM_MACRO_FIXED_MOVE(hRsm, 1, AXIS_X, -1000);
// command to move axis X for -1000 pulses.
RSM_MACRO_END_IF(hRsm, 1);
RSM_MACRO_MP_CLOSE(hRsm, 1);
// module 1, Macro Program #100 is written into i-8094H
//=======================================================
RSM_MP_CALL(hRsm, 1, MP100);
// execute  Macro Program #100 of module 1.
```

9.6.2 ELSE Statement

※Δ eRTU RSM_MACRO_ELSE (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

The block under RSM_MACRO_ELSE will be executed if the conditional statement of RSM_MACRO_IF is false; otherwise, it will be ignored.

Category:

MODBUS sub_function; MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_ELSE (hRsm, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C D8	***Sub_funciton code***			

9.6.3 End of IF Statement

※Δ eRTU RSM_MACRO_END_IF (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

This function indicates the end of a conditional block. A conditional block always begins with an RSM_MACRO_IF statement and ends with this function.

Category:

MODBUS sub_function; MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_END_IF (hRsm, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C D9	***Sub_funciton code***			

9.7 Jump Commands (GOTO-LABEL)

9.7.1 GOTO Statement

※ **eRTU RSM_MACRO_GOTO(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *lableNo*)**

Description:

The RSM_MACRO_GOTO Macro statement is a label based statement that transfers the control directly to the labeled location in the Macro program. The goto statement is very useful while working with nested loops. It enables to leave all nested nested loops with one command by jumping from the inner loop to an RSM_MP_LABEL position ouside the nested loop.

Limitations:

- **A Maximum of four RSM_MACRO_LABEL position can be defined.**
- **It is NOT allowed to jump into a loop block. Therefore, the RSM_MACRO_LABEL statement can not be located inside a loop.**
- **Jumping into an MP_IF block is prohibited. RSM_MACRO_LABEL command is not allowed to be located inside an RSM_MACRO_IF block.**
- **You can only jump within a Macro table and it is not allowed to jump from one Macro table (e.g. MP1) to another Macro table (e.g. MP2).**
- **Can not be used inside an ISR Macro table.**

Figure 8: RSM_MACRO_GOTO application

Category:

MODBUS sub_function; only inside MP Macro table allowed.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
labelNo:	**Only four labels are supported for each MP macro table: 0, 1, 2, 3 (LABEL0, LABEL1, LABEL2, LABEL3)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_GOTO (hRsm, 1, 3);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C DF	***Sub_funciton code***			
1	00 03	***labelNo (3 ➔0x03)***			

9.7.2 LABEL Statement

※ **eRTU RSM_MACRO_LABEL(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *lableNo*)**

Description:

This function indicates the target position of a jump statement. The RSM_MACRO_LABEL can be positioned before or after RSM_MACRO_GOTO Macro statement. In a Macro only one RSM_MACRO_LABEL with the same Label number can be used but more than one RSM_MACRO_GOTO with the same Label number is allowed.

Note:

RSM_MACRO_LABEL command is not allowed inside a MP_IF or MP_FOR block.

Category:

MODBUS sub_function; only inside MP Macro table allowed.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
labelNo:	**Only four labels are supported for each MP macro table: 0, 1, 2, 3 (LABEL0, LABEL1, LABEL2, LABEL3)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_LABEL (hRsm, 1, 3);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0C E1	***Sub_funciton code***			
1	00 03	***labelNo (3 ➔0x03)***			

9.8 Timer

※ eRTU RSM_MACRO_TIMER (HANDLE *hRsm*, BYTE *SlaveAddr*, long *varNo*)

Description:

Delay the execution of the following Macro commands. The execution of the following Macro commands is held until the specified time has elapsed.

Category:

MODBUS sub_function; for MP Macro table.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
varNo:	**Time in milliseconds. The following inputs are supported: Value (0 ~ +2,000,000,000) ms, or MACRO variables (VAR0 ~ VAR511)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
RSM_MACRO_TIMER (hRsm, 1, 200);
//delay the execution of the function for 200ms.
```

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 03	06
Register[]	**Value (hex)**	**Remarks**			
0	0C DA	***Sub_funciton code***			
1	00 00	***MSW of varNo***			
2	00 C8	***LSW of varNo (200 = 0xC8)***			

9.9 Wait Until Motion Command has been Executed

eRTU RSM_STOP_WAIT(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

※ eRTU RSM_MACRO_STOP_WAIT(HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*)

Description:

The command ensures that the motion command for the assigned axis is completed before the next macro command is being executed. The command waits until the motion control chip has finished outputting pulses for the designated axis then continues to process the following command.

Category:

MODBUS sub_function; RTC, MP.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
axis:	**Axis or axes (Please refer to Table 4) 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 chapter 2.2)

Remark:

The Sub_function code of RSM_STOP_WAIT is 0A DB.
The Sub_function code of RSM_MACRO_STOP_WAIT is 0C DB.

MODBUS example:

RSM_STOP_WAIT (hRsm, 1, *AXIS_Y*);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 02	04
Register[]	**Value (hex)**	**Remarks**			
0	0A DB/0C DB	***Sub_funciton code***			
1	00 02	***axis (2 ➔ AXIS_Y)***			

9.10 Exit Macro Program

※Δ eRTU RSM_MACRO_EXIT_MACRO (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

The execution of a MP or ISR Macro table will be stopped and the table will be exited as soon as the Macro interpreter encounters this command. If a nested Macro call is being executed, this command causes the command interpreter to jump to calling Macro layer one level up (Figure 9). If the top level Macro layer encounters RSM_MACRO_EXIT_MACRO, the execution of Macro commands terminates all together. Only in case of a new harware interrupt occurs, the relevant ISR Macro will be executed. This command behaves similarly to the RSM_MACRO_MP_CLOSE command except that it can be called at any place inside a macro table, whereas the RSM_MACRO_MP_CLOSE command has to be placed at the end of a macro table.

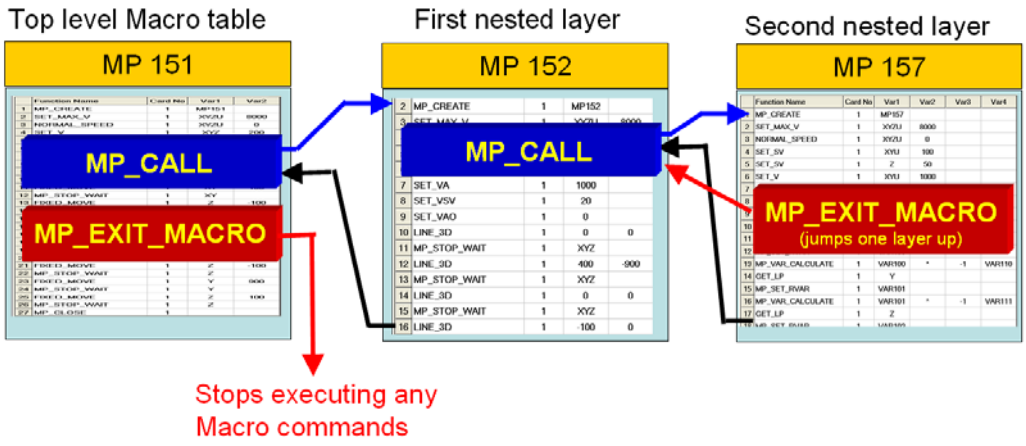


Figure 9: Exiting macro tables

Category:

MODBUS sub_function; MP and ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MACRO_EXIT_MACRO (hRsm, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0C DD	***Sub_funciton code***			

9.11 Terminate all Macro Executions

eRTU RSM_MP_TERMINATE (HANDLE *hRsm*, BYTE *SlaveAddr*)

※Δ eRTU RSM_MACRO_MP_TERMINATE (HANDLE *hRsm*, BYTE *SlaveAddr*)

Description:

All currently running MP or ISR Macro program will be terminated immediately. After all the Macro programs have been terminated successfully, the command automatically resets itself so that new Macro programs can be executed. This command is intended for emergent situation where any Macro execution needs to be stopped immediately. This command can be called both inside a Macro program and outside a Macro program, for example, by the host controller or a remote HMI controller.

Warning:

This command does not stop the motion chip from continuing executing the motion command received prior to the RSM_MP_TERMINATE command. In order to stop the motion chip from outputting pulse signals, one of the following commands has to be called:

- **RSM_STOP_SUDDENLY**
- **RSM_STOP_SLOWLY**
- **RSM_VSTOP_SUDDENLY**
- **RSM_VSTOP_SLOWLY**

Therefore, if it required stopping motion and Macro commands from execution then one of the above four STOP commands as well as the RSM_MP_TERMINATE has to be called.

Category:

MODBUS sub_function; RTC, MP, ISR.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

RSM_MP_TERMINATE (hRsm, 1);

The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 01	02
Register[]	**Value (hex)**	**Remarks**			
0	0A E2/0C E2	***Sub_funciton code***			

9.12 Get Macro Download Status

```
eRTU RSM_GET_MP_DOWNLOAD_STATUS(HANDLE hRsm, BYTE
      SlaveAddr, MpDownloadInfo *pInfo)

typedef struct
{
     WORD MpNo;
     WORD IsrNo;
     WORD wCmd;
     WORD wLineNo;
     WORD wErrCode;
}MpDownloadInfo;
```

Description:

This function confirms whether the previous MP or ISR macro download procedure was successful. It indicate which Macro type has been downloaded (MP or ISR) and its number. In case of a download error the type of error, the command and Macro line number will be indicated.

Category:

MODBUS table; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
pInfo:	***MpNo*: indicates the MP Macro number (1-MP1, 2 - MP2 ...). If no MP Macro has been downloaded MpNo will be zero (MpNo=0).** ***IsrNo*: indicates the latest ISR Macro number downloaded (1-ISR1, 2 -ISR2 ...). If no ISR Macro has been downloaded IsrNo will be zero (IsrNo=0).** ***wCmd*: code of the command in the Macro program which caused the error (see command code chapter 12.3).** ***wLineNo*: the line number in the Macro program. The first line has the number zero.** ***wErrCode*: error code describing the cause of macro download failure:**

	– ***0x00*: no error occurred. Download was succesful.** – ***0x01*: number of commands lines exceeds the quantity of line provide by the corresponding Macro table.** **<u>IF command errors:</u>** – **0x02: Number of nested MP_IF statements exceeds the limit of six inner nested MP_IFs.** – ***0x03*: MP_ELSE command outside MP_IF block** – ***0x04*: MP_END_IF command outside MP_IF block. The MP_END_IF statement has not got a corresponding MP_IF statement.** – ***0x05*: MP_IF block has not been closed by MP_END_IF command.** – ***0x06*: Wrong MP_IF conditional operator.** **<u>FOR loop command errors:</u>** – **0x07: Number of nested MP_FOR exceeds the limit.** – ***0x08*: MP_EXIT_FOR command outside MP_FOR loop. The corresponding MP_FOR command is missing.** – ***0x09*: MP_NEXT command outside MP_FOR block. The corresponding MP_FOR command is missing.** – ***0x0A*: MP_FOR block has not been closed with a MP_NEXT command.** **<u>GOTO command errors:</u>** – ***0x0B*: Label number is being used more than once.** – ***0x0C*: MP_LABEL inside a MP_IF block. Jumping into a MP_IF block is prohibited.** – ***0x0D*: MP_LABEL inside a MP_FOR block. Jumping into a MP_FOR block is prohibited.** – **0x0E: MP_GOTO command has NO corresponding MP_LABEL command.** – ***0x0F*: MACRO (MP or ISR) command not defined: Command will not be written to the FRAM.** – ***0x10*: Wrong MP_VAR_CALCULATE operator.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

MODBUS example:

```
MpDownloadInfo MpInfo;
RSM_GET_MP_DOWNLOAD_STATUS(hRsm, 1, &MpInfo);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01 (Slave Address)	04	00 6E (110)	00 05

MODBUS response:

UID (hex)	FC (hex)	Byte Count(hex)	Registers (hex)
01	04	0A	See next table

Register []	Value (hex)	Remarks
0	00 02	*MpNo:* MP number (e.g. MP2)
1	00 00	*IsrNo:* ISR number (no ISR Macro has been downloded)
2	0A D6	*wCmd:* command code (here MP_NEXT)
3	00 09	*wLineNo*: command line within Macro table (line number 9)
4	00 09	*wErrCode:* error code describing the kind of error (here: MP_NEXT command outside MP_FOR block. The corresponding MP_FOR command is missing.)

9.13 Change Velocity on the Fly

eRTU RSM_CHANGE_VEL_ON_FLY (HANDLE *hRsm*, BYTE *SlaveAddr*, BYTE *axis*, DWORD *data*)

Description:

If a Macro program is running inside the i-8094H, the remote host can call this function to change the drive speed set by the Macro program. Once the new drive speed has been set the drive will accelerate to the specified velocity.

Category:

MODBUS sub_function; RTC.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
***axis*:**	**Axis or axes (Please refer to Table 4) The axis can be either X, Y, Z or U (1 or 2 or 4 or 8).**
***data*:**	**Drive speed (Please refer to 2.5 for maximm speed) PPS.**

Return:

0: Success; Others: Fail (Please refer to chapter 2.2)

Remark:

The Sub_function code of RSM_CHANGE_VEL_ON_FLY is 0x0A43.

MODBUS example:

RSM_CHANGE_VEL_ON_FLY (hRsm, 1, AXIS_X, 120000L);

//set the speed for the X axis on module 1 to 120000 PPS.

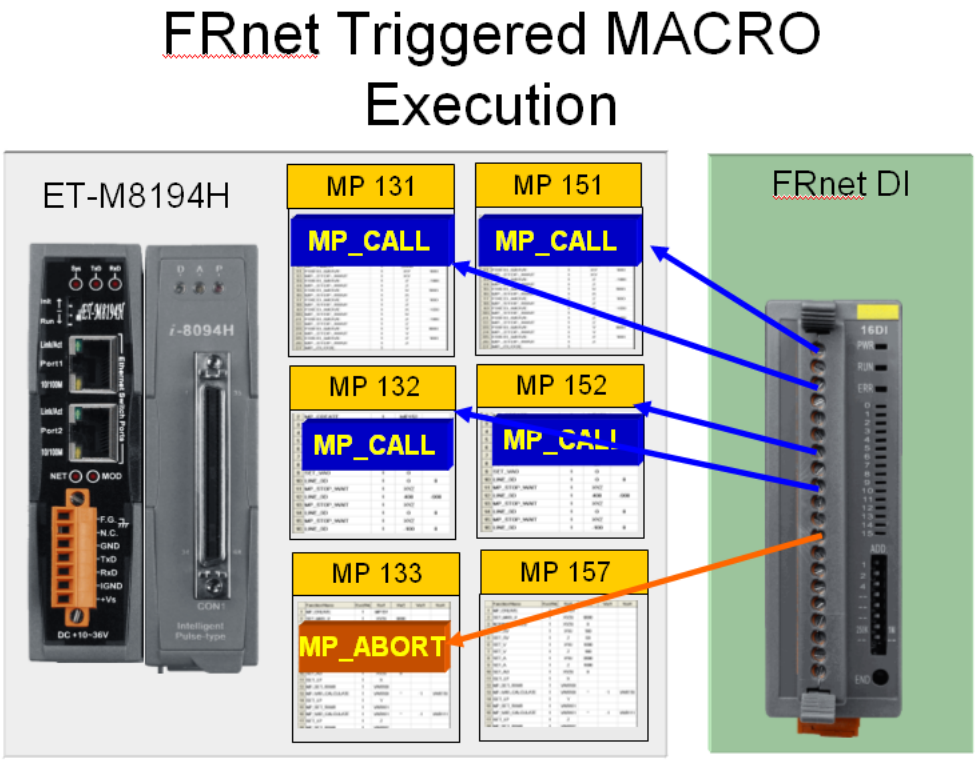
The MODBUS command is listed as follows:

UID (hex)		FC (hex)	St_Addr. (hex)	Word Count (hex)	Byte Count (hex)
01		10	1F 40	00 04	08
Register[]	**Value (hex)**	**Remarks**			
0	0A 43	***Sub_function code***			
1	00 01	***axis (1 = AXIS_X)***			
2	00 01	***MSW of data***			
3	D4 C0	***LSW of data (120000 = 0x1D4C0)***			

10 Stand Alone Controller

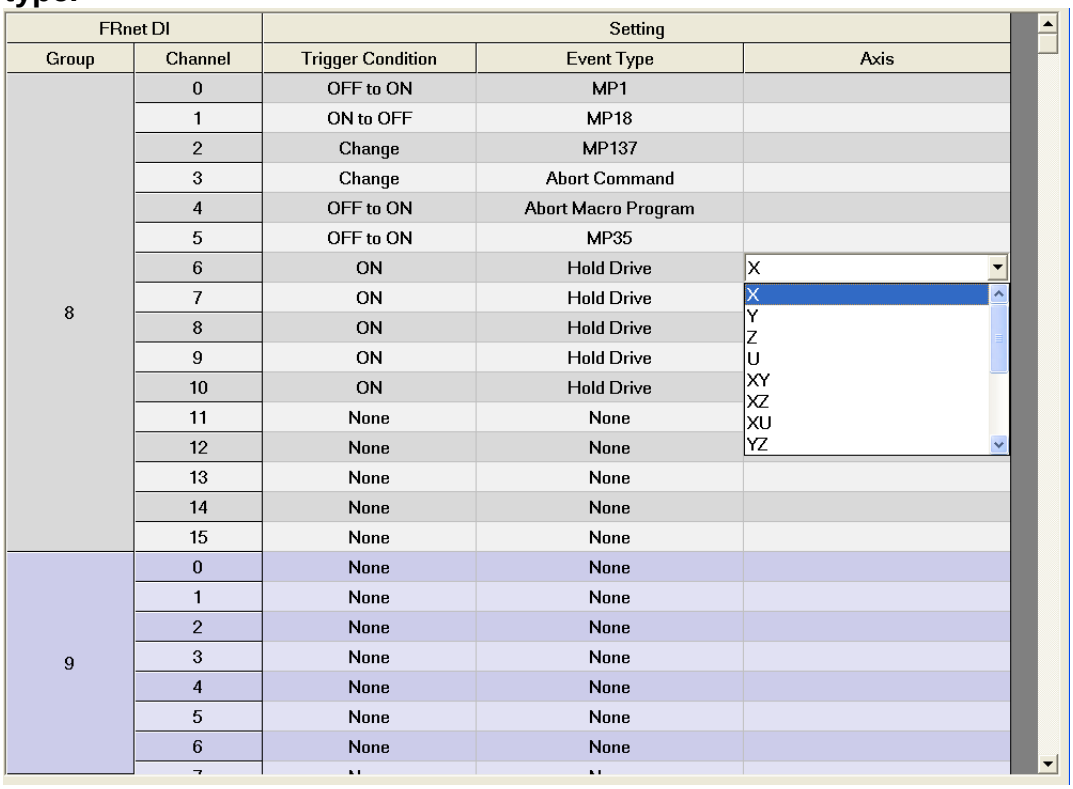
The ET-8194H together with an FRnet DI can operate as a standalone controller without being connected to a Modbus master. Each FRnet DI can be assigned to a motion program stored in the RS-M8194H as a Macro program. By triggering an FRnet DI event the corresponding Macro program starts executing. Only one Macro program can be triggered at a time.

In the following the procedure for configurating the RS-M8194H to support FRnet DI triggering will be described:



- **Step 1:**
  Write a Macro program and download it to the RS-M8194H (see chapter 9).
- **Step 2:**
  Use the EzMove Utility to select which DI channel to use for FRnet

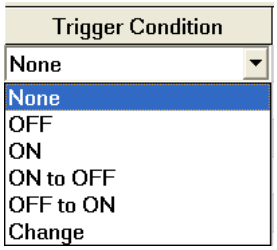
triggered Macro execution. Connect to the RS-M8194H and open the "FRnet DI Setting" window (Setting → FRnet DI...). For each DI channel select the trigger condition and the corresponding event type.



FRnet DI		Setting		
Group	Channel	Trigger Condition	Event Type	Axis
8	0	OFF to ON	MP1	
	1	ON to OFF	MP18	
	2	Change	MP137	
	3	Change	Abort Command	
	4	OFF to ON	Abort Macro Program	
	5	OFF to ON	MP35	
	6	ON	Hold Drive	X
	7	ON	Hold Drive	
	8	ON	Hold Drive	
	9	ON	Hold Drive	
	10	ON	Hold Drive	
	11	None	None	
	12	None	None	
	13	None	None	
	14	None	None	
	15	None	None	
9	0	None	None	
	1	None	None	
	2	None	None	
	3	None	None	
	4	None	None	
	5	None	None	
	6	None	None	

X
Y
Z
U
XY
XZ
XU
YZ

The RS-M8194H defines the following DI trigger conditions:

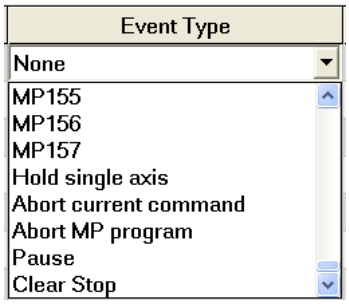


Trigger Condition

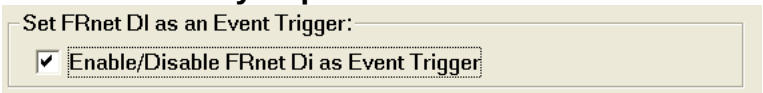
None

None
OFF
ON
ON to OFF
OFF to ON
Change

If the DI channel is not being used for event triggering then set it to "None". The DI triggered event can be a Macro execution, pause or abort of the current Macro execution execution, or stop of motion execution



- **Step 3:**
  Download the DI trigger setting to the RS-M8194H by clicking "Send Table"
- **Step 4:**
  Enable FRnet DI triggering by clicking the check box. Now the RS-M8194H is ready to process FRnet DI events



It is important to notice that only one Macro program can run at a time. Therefore if an FRnet DI event is generated to execute a new Macro program while a Macro program is already running then the event first waits until the current Macro program has finished executing before starting the next one.

11 Library Version

11.1 Modbus P824 Library

eRTU RSM_GET_RSM_FIRMWARE_VERSION (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD* *RSM_Ver*);

Description:

This function reads the Modbus library version of the RS-M8194H.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
RSM_Ver	**Version number Example: 0x03000000 → Version 3.0**

MODBUS example:

```
DWORD RSM_Ver;
HANDLE hRsm;
RSM_GET_RSM_FIRMWARE_VERSION(hRsm, &RSM_Ver);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 52	00 02

MODBUS response:

UID (hex)	FC (hex)	Byte Count (hex)	*Version* (hex) High Word	*Version*(hex) Low Word
01	04	04	02 00	00 00

11.2 i-8094H Library Version

eRTU RSM_GET_i8094H_FIRMWARE_VERSION (HANDLE *hRsm*, BYTE *SlaveAddr*, DWORD* *i8094H_Ver*);

Description:

This function reads the i-8094H library version of the RS-M8194H.

Parameters:

Paramters	Description
hRsm:	**COM port handle**
***SlaveAddr*:**	**Modbus RTU Slave Address (valid range: 1 to 247)**
i8094H_Ver	**Version number** **Example: 0x02210300** ▪ **PCB 2.21** ▪ **The Low WORD (last two hex number) represents the i-8094H firmware Version: 3.00**

MODBUS example:

```
DWORD i8094H_Ver;
HANDLE hRsm;
RSM_GET_i8094H_FIRMWARE_VERSION(hRsm, &i8094H_Ver);
```

MODBUS request:

UID (hex)	FC (hex)	St_Addr. (hex)	Word Count (hex)
01	04	00 54	00 02

MODBUS response:

UID (hex)	FC (hex)	Byte Count (hex)	*Version*(hex) High Word	*Version*(hex) Low Word
01	04	04	02 21	02 01

11.3 RS-M8194H DLL Version

```
WORD RSM_GET_DLL_VERSION (void);
```

Description:

This function reads the read RS-M8194H API Library version. This function is an internal function and can be called without connecting to the ET-M8194H.

0x0100 means the version number is 1.00. The high byte is the major version number and the low byte is the minor version number.

12 Appendix

12.1 MODBUS Input Registers

Use Modbus function code 4 to read the input register table. The table addresses defined below is zero-based but for some PLCs and HMIs the addresses are one-based. This means the one based device use the address one to access the register table at index zero. The format of the one-based input register table is 3xxxx, x is digital number, '3' represents the table and is defined for Input registers; and xxxx is the one-based address of a register.

In PLC, the first input register defined at address 30001 is the same as the first register at address 0 shown in the table below. Most PLC systems which addresses do not start from zero can be referred as one-based system. The following tables will only show zero-based addresses.

When a value is represented by two registers data, the order of most significant WORD (MSW) and least significant WORD (LSW) can be determined by reading the current WORD order setting. The WORD order setting is defined at address 14 (zero-based) in holding register table. If WORD order is set to 0, the first register (lower address) represents the high-word (**MSW**) and the second register is low-word (**LSW**).

12.1.1 Read FRnet DI/O

Appendix table 1: FRnet DI/O (PLC address: 30001 to 30016)

Read FRnet DIO				
Variable Name		**Address (zero based)**	**Type**	**Comment**
FRnet DO	Group 0	0	R	Read the FRnet DO group 0 (16-bit)
	Group 1	1	R	Read the FRnet DO group 1 (16-bit)
	Group 2	2	R	Read the FRnet DO group 2 (16-bit)
	Group 3	3	R	Read the FRnet DO group 3 (16-bit)
	Group 4	4	R	Read the FRnet DO group 4 (16-bit)
	Group 5	5	R	Read the FRnet DO group 5 (16-bit)
	Group 6	6	R	Read the FRnet DO group 6 (16-bit)
	Group 7	7	R	Read the FRnet DO group 7 (16-bit)
FRnet DI	Group 8	8	R	Read the FRnet DI group 8 (16-bit)
	Group 9	9	R	Read the FRnet DI group 9 (16-bit)
	Group 10	10	R	Read the FRnet DI group 10 (16-bit)
	Group 11	11	R	Read the FRnet DI group 11 (16-bit)
	Group 12	12	R	Read the FRnet DI group 12 (16-bit)
	Group 13	13	R	Read the FRnet DI group 13 (16-bit)
	Group 14	14	R	Read the FRnet DI group 14 (16-bit)
	Group 15	15	R	Read the FRnet DI group 15 (16-bit)

12.1.2 Read Motion Chip DI Status of Daughterboard

Appendix table 2: DI Status of Daughterboard (PLC address: 30017 to 30020)

Read DI Status (of Daughterboard)				
Variable Name		**Address (zero based)**	**Type**	**Comment**
DI ALL	X-Axis	16	R	All DI status of AXIS-X, If the individual DI is needed, read X_DI_0 ~ X_DI_9.
	Y-Axis	17	R	All DI status of AXIS-Y, If the individual DI is needed, please read Y_DI_0 ~ Y_DI_9.
	Z-Axis	18	R	All DI status of AXIS-Z, If the individual DI is needed, please read Z_DI_0 ~ Z_DI_9.
	U-Axis	19	R	All DI status of AXIS-Z, If the individual DI is needed, please read U_DI_0 ~ U_DI_9.

12.1.3 Read Error Code

Appendix table 3: Error Code (PLC address: 30021 to 30026)

Read Error Code				
Variable Name		**Address (zero based)**	**Type**	**Comment**
ERROR CODE	X-Axis	20	R	The error occurred on AXIS-X. Please refer to RSM_GET_ERROR_CODE() for detailed information.
	Y-Axis	21	R	The error occurred on AXIS-Y. Please refer to RSM_GET_ERROR_CODE() for detailed information.
	Z-Axis	22	R	The error occurred on AXIS-Z. Please refer to RSM_GET_ERROR_CODE() for detailed information.
	U-Axis	23	R	The error occurred on AXIS-U. Please refer to RSM_GET_ERROR_CODE() for detailed information.
MP Call number		24	R	Current running MP or ISR number
EMERGENCY_STOP state		25	R	Shows the current emergency state 1- emergency stop enabled 0 – no emergency disabled

12.1.4 Read Logic and Encoder Position, Acceleration, Velocity

Appendix table 4: Logic and Encoder Position, Acceleration, Velocity (PLC address: 30027 to 30058)

Motion Status				
Variable Name		**Address (zero based)**	**Type**	**Comment**
Logic Position	X	26-27	R	Logical position of X-axis.
	Y	28-29	R	Logical position of Y-axis.
	Z	30-31	R	Logical position of Z-axis.
	U	32-33	R	Logical position of U-axis.
Encoder Position	X	34-35	R	Encoder feedback position of X-axis. It takes two registers.
	Y	36-37	R	Encoder feedback position of Y-axis. It takes two registers.
	Z	38-39	R	Encoder feedback position of Z-axis. It takes two registers.
	U	40-41	R	Encoder feedback position of U-axis. It takes two registers.
Current velocity	X	42-43	R	Current velocity of X-axis. It takes two registers.
	Y	44-45	R	Current velocity of Y-axis. It takes two registers.
	Z	46-47	R	Current velocity of Z-axis. It takes two registers.
	U	48-49	R	Current velocity of U-axis. It takes two registers.
Current Acceleration	X	50-51	R	Current acceleration of X-axis. It takes two registers.
	Y	52-53	R	Current acceleration of Y-axis. It takes two registers.
	Z	54-55	R	Current acceleration of Z-axis. It takes two registers.
	U	56-57	R	Current acceleration of U-axis. It takes two registers.

12.1.5 Read Stop Status

Appendix table 5: Stop Status (PLC address: 30059 to 30062)

Variable Name		Address (zero based)	Type	Comment
Stop Status	X	58	R	1:stop, 0: moving
	Y	59	R	1:stop, 0: moving
	Z	60	R	1:stop, 0: moving
	U	61	R	1:stop, 0: moving

12.1.6 Read Latch

Appendix table 6: Read Latch (PLC address: 30063 to 30070)

Variable Name		Address (zero based)	Type	Comment
Latch	X	62-63	R	Value in X-axis latch register long type
	Y	64-65	R	Value in Y-axis latch register long type
	Z	66-67	R	Value in Z-axis latch register long type
	U	68-69	R	Value in U-axis latch register long type

12.1.7 Read Error State and Free Buffer Size

Appendix table 7: Error State and Free Buffer Size (PLC address: 30071 to 30072)

Variable Name	Address (zero based)	Type	Comment
ERROR_STATE	70	R	1: error occurred; 0: no error
FREE_BUFFER_SIZE	71	R	Get the available block-numbers in Buffer. The maximum block is 30.

12.1.8 Read i-8094H Interrupt

Appendix table 8: i-8094H Interrupt (PLC address: 30073 to 30081)

i-8094H Interrupt RS-M8194H (Return Interrupt RINT)			
Variable Name	**Address (zero based)**	**Type**	**Comment**
RINT_STATE_ALL	72	R	Read the RINT state. For detailed, please refer to the descriptions of **RSM_RINT_WAIT()** and **RSM_RINT_ENABLE()**.
Line_Scan_Completed	73	R	Bit0 of RINT_STATE
MP_Completed	74	R	Bit1 of RINT_STATE
User-Defined_RINT	75	R	Bit2 of RINT_STATE
Continuous_Inp_Interrupt	76	R	Bit3 of RINT_STATE
…..Undefined	77	--	0
…..Undefined	78	--	0
Axes_Error	79	R	Bit6 of RINT_STATE
Module_Error	80	R	Bit7 of RINT_STATE

12.1.9 Read Firmware Version

Appendix table 9: Firmware Version (PLC address: 30082 to 30087)

Others			
Variable Name	**Address (zero based)**	**Type**	**Comment**
i-8094H Module ID	81	R	i-8094H：0x**44** i-8094A：0x**55**
RS-M8194H Firmware Version	82-83	R	0x**0100**0000 means Ver：1.00
i-8094H Firmware Version	84-85	R	0x02210201 ▪ High Word for PCB version ▪ Low WORD represents the i-8094H firmware Version: // Example: ▪ High WORD o PCB Version 2.21 ▪ Low WORD o 0201->02 major version, 01-> minor version number

12.1.10 Read DI Signal of Daughterboard

Appendix Table 10: DI Status of Daughterboard (PLC address: 30017 to 30020)

Read DI Signal (of Daughterboard)				
Variable Name		**Address (zero based)**	**Type**	**Comment**
DI ALL	X-Axis	88	R	All DI signal of AXIS-X, If the individual DI is needed, read X_DI_SIG_0 ~ X_DI_SIG _7.
	Y-Axis	89	R	All DI signal of AXIS-Y, If the individual DI is needed, please read Y_DI_SIG _0 ~ Y_DI_SIG _7.
	Z-Axis	90	R	All DI signal of AXIS-Z, If the individual DI is needed, please read Z_DI_SIG _0 ~ Z_DI_SIG _7.
	U-Axis	91	R	All DI signal of AXIS-Z, If the individual DI is needed, please read U_DI_SIG _0 ~ U_DI_SIG _7.

12.1.11 Read Absolute Logic Position

Appendix table 11: Absolute Logic Position (PLC address: 30093 to 30100)

Motion Status				
Variable Name		**Address (zero based)**	**Type**	**Comment**
Absolute Logic Position	X	160-161	R	Absolute logical position of X-axis.
	Y	162-163	R	Absolute logical position of Y-axis.
	Z	164-165	R	Absolute logical position of Z-axis.
	U	166-167	R	Absolute logical position of U-axis.

12.1.12 Read RS-M8194H and i-8094H State

Appendix table 12: RS-M8194H and i-8094H State (PLC address: 30101 to 30103)

Current state of the RS-M8194H and i-8094H			
Variable Name	**Address (zero baed)**	**Type**	**Comment**
RS-M8194H state	100	R	The current state of the RS-M8194H module: 0- ready for receiving new RTC and Macro commands 1- RSM is in an MACRO download state (MPn) 2- RSM is in an MACRO download state (ISRn)
i-8094H state	101	R	The current state of the i-8094H module: 3- ready for receiving new RTC commands 4- Calling the initialization table 5- Initialization finished 6- downloading MACRO commands (MP or ISR commands) to the FRAM or is waiting for the next MACRO command 7- Macro download finished 8- MACRO program (MP) is being executed 9- MACRO program (MP) execution finished 10- Interrupt MACRO (ISR) program is being executed 11- Interrupt MACRO (ISR) execution finished 255- firmware is booting
Illegal Modbus function	102	R	Cause of the ILLIGAL FUNCTION Modbus exception response: 0- no error 1- function code not supported 2- Parameter value in NOT within defined range 3- No such command

			4- More than the avalaible command lines are used for the specific MACRO program (MPn or ISRn) 5- A MP command is being used inside ISR MACRO. 6- A ISR command is being used inside MP MACRO 7- A RTC command is being used inside a MP or ISR MACRO 8- A MP or ISR MACRO command is being used outside a MACRO program as real time command **0xB1** - The max number of nested MP_IF commands has been exceeded **0xB2** - MP_ELSE command outside MP_IF block **0xB3** - MP_END_IF command outside MP_IF block **0xB4** - MP_IF block has not been closed by MP_END_IF command **0xB5** - Number of nested MP_FOR exceeds the limit **0xB6** -MP_EXIT_FOR command outside MP_FOR loop **0xB7** - MP_NEXT command outside MP_FOR block **0xB8** - MP_FOR block has not been closed with a MP_NEXT command **0xB9** - Label number is being used more than once **0xBA** - MP_LABEL inside a MP_IF block. Jumping into a MP_IF block is prohibited **0xBB** - MP_LABEL inside a MP_FOR block. Jumping into a MP_FOR block is prohibited **0xBC** - MP_GOTO command has NO corresponding MP_LABEL command **0xC0** - A DWORD is required but only one register has been received **0xC1** -DPRAM of i-8094H is full **0xC2** -Module not in slot **0xC3** - A Macro program is currently being executed inside the i-8094H

12.1.13 Macro Program Download Error Messages

Appendix table 13: Program Download Error Message (PLC address: 30111 to 30115)

Macro program download error message			
Variable Name	**Address (zero based)**	**Type**	**Comment**
Macro MP number (MPxx)	110	R	The last downloaded Macro MP program number since RS-M8194H power on.
Macro ISR number (ISRxx)	111	R	The last downloaded Macro ISR program number since RS-M8194H power on.
Command type	112	R	The number identifying the command. See sub function code. The register following the 8000 sub function register identifies the command.
Line number	113	R	The line number of the Macro program at which the error occurred
Error type	114	R	Describes the type of error

12.1.14 Macro Program Execution Error Messages

Appendix table 14: Macro Program Execution Error Messages (PLC address: 30116 to 30120)

Macro program execution error message			
Variable Name	**Address (zero based)**	**Type**	**Comment**
Macro MP number (MPxx)	115	R	The last downloaded Macro MP program number since RS-M8194H power on.
Macro ISR number (ISRxx)	116	R	The last downloaded Macro ISR program number since RS-M8194H power on.
Command type	117	R	The number identifying the command at which the error occurred. See sub function code. The register following the 8000 sub function register identifies the command.
Line number	118	R	The line number of the Macro program at which the error occurred
Error type	119	R	Describes the type of error

12.1.15 Motion Chip Triggered Interrupt

Appendix table 15: Motion Chip Triggered Interrupt (PLC address: 30131 to 30140)

Motion chip triggered interrupt				
nINT	**Symbol**	**Address (zero based)**	**Type**	**Description**
0	PULSE	130	R	Interupt occurs when pulse is up.
1	P>=C-	131	R	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP- register.
2	P<C-	132	R	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP- register.
3	P<C+	133	R	Interrupt occurs when the value of logical / real position counter is smaller than that of COMP+ register.
4	P>=C+	134	R	Interrupt occurs when the value of logical / real position counter is larger than or equal to that of COMP+ register.
5	C-END	135	R	Interrupt occurs at the end of the constant speed drive or completion of Acceleration Offset Pulse output
6	C-STA	136	R	Interrupt occurs at the start of the constant speed drive or begin of Acceleration Offset Pulse output
7	D-END	137	R	Interrupt occurs when the driving is finished
8	HMEND	138	R	Automatic home search terminated
9	SYNC	139	R	Synchronous action was activated

12.2 Holding Registers

Use Modbus function code 6 or 16 to write to the holding register(s) and function code 3 to read the registers. The table addresses defined below is zero-based. For some PLC or HMI, the addresses maybe show on their documents as 4xxxx, x is digital number. For example, 40001 to 49999. '4' represents the table is defined for holding registers; and xxxx is the address of a register.

In PLC, the first holding register defined at address 40001 is the same as the first register at address 0 shown in the following table. Most PLC systems which addresses do not start from zero can be referred as one-based system. The following tables will only show zero-based addresses. When PLC or HMI users configure their registers, they have to consider register number mapping problem carefully.

When a parameter needs two registers to store, such as DWORD, long, float, the order of MSW and LSW can be determined by setting a flag (WORD Order). WORD Order is defined at address 14 (zero-based) in holding register table. If WORD Order is set to 0, the first register (lower address) is high-word (**MSW**) and the second register is low-word (**LSW**).

R/W – read/write

12.2.1 FRnet DO

Appendix table 16: FRnet DO (PLC address: 40001 to 40008)

FRnet DO				
Variable Name		**Address (zero base)**	**Type**	**Comment**
FRnet DO	Group 0	0	R/W	Set/Read the FRnet DO group 0
	Group 1	1	R/W	Set/Read the FRnet DO group 1
	Group 2	2	R/W	Set/Read the FRnet DO group 2
	Group 3	3	R/W	Set/Read the FRnet DO group 3
	Group 4	4	R/W	Set/Read the FRnet DO group 4
	Group 5	5	R/W	Set/Read the FRnet DO group 5
	Group 6	6	R/W	Set/Read the FRnet DO group 6
	Group 7	7	R/W	Set/Read the FRnet DO group 7

12.2.2 Macro Call, FRnet Event and WORD Order Setting

Appendix table 17: Macro Call, FRnet Event and WORD Order Setting (PLC address: 40009 to 40015)

Variable Name	Address (zero based)	Type	Comment
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. n=1-157
FRNet Event	9	--	Enable FRnet DI to generate events 0 – disable FRnet event trigger 1 – enable FRnet event trigger
Reserved	10-13	--	Not defined
WORD Order	14	R/W	This setting is relevant for DWORD, long, float variables: 0: Big endian WORD (reverse type). First register represents the most significant word the second register represents the least significant word 1: Small Endian WORD Most significant word is the second register

12.2.3 Read/ Write Logic and Encoder Position, Acceleration, Velocity

Appendix table 18: Logic and Encoder Position, Acceleration, Velocity (PLC address: 40091 to 40122)

Motion Status				
Variable Name		**Address (zero based)**	**Type**	**Comment**
Absolute Position	X-Axis	82-83	R/W	Absolute position of X-axis.
	Y-Axis	84-85	R/W	Absolute position of Y-axis.
	Z-Axis	86-87	R/W	Absolute position of Z-axis.
	U-Axis	88-89	R/W	Absolute position of U-axis.
Logic Position (LP)	X-Axis	90-91	R/W	Logical position of X-axis.
	Y-Axis	92-93	R/W	Logical position of Y-axis.
	Z-Axis	94-95	R/W	Logical position of Z-axis.
	U-Axis	96-97	R/W	Logical position of U-axis.
Encoder Position (EP)	X-Axis	98-99	R/W	Encoder feedback position of X-axis.
	Y-Axis	100-101	R/W	Encoder feedback position of Y-axis.
	Z-Axis	102-103	R/W	Encoder feedback position of Z-axis.
	U-Axis	104-105	R/W	Encoder feedback position of U-axis.
Current Velocity (CV)	X-Axis	106-107	R	Current velocity of X-axis.
	Y-Axis	108-109	R	Current velocity of Y-axis.
	Z-Axis	110-111	R	Current velocity of Z-axis.
	U-Axis	112-113	R	Current velocity of U-axis.
Current Acceleration (CA)	X-Axis	114-115	R	Current acceleration of X-axis.
	Y-Axis	116-117	R	Current acceleration of Y-axis.
	Z-Axis	118-119	R	Current acceleration of Z-axis.
	U-Axis	120-121	R	Current acceleration of U-axis.

12.2.4 Read/ Write VAR Variables

Appendix table 19: VAR Variables (PLC address: 40301 to 41324)

Macro Variables			
Variable Name	**Address (zero based)**	**Type**	**Comment**
VAR0	300-301	R/W	Value of VAR0.
VAR1	302-303	R/W	Value of VAR1.
…	…		VARn = 300+2*n to 300+2*n+1
VAR510	1320-1321	R/W	Value of VAR510.
VAR511	1322-1323	R/W	Value of VAR511.

12.2.5 Sub Function Code Definition

Appendix table 20: Sub Function Code Definition (PLC address: 48001 to 48101)

Motion Commands Sub Function Code 16			
Variable Name	**Address (zero based)**	**Type**	**Comment**
Sub_Function_Code	8000	W	The mapping of sub function code can refer to table in **section 12.3**.
Reg1	8001	W	The definitions of parameters depend on the sub functions. Please refer to the table in **section 12.3**.
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 **section 12.1 and 12.2** are valuable.
Reg4	8004	W	Users can refer to examples in the functions description above to know how to set the parameters.
…			The basic rules of parameter definitions are : BYTE ➔ one register (low-byte) WORD ➔ one register DWORD ➔ two registers long ➔ two registers float ➔ two registers
Reg100	8100	W	

12.3 Sub-Function Code Mapping Table

Please use the FC=16 to call the sub-functions; the related starting address is 8000 (zero-based). 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 section 12.1 and 12.2.

Appendix table 21: Sub Function Code mapping table

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
2.1.1	RSM_OPEN_COM_PORT	xxxxxx	xxxxxx	x
2.1.2	RSM_CLOSE_COM_PORT	xxxxxx	xxxxxx	x
2.1.3	RSM_CONNECTION_STATE	xxxxxx	xxxxxx	x
2.1.4	RSM_SET_TIMEOUT	xxxxxx	xxxxxx	x
2.1.5	RSM_GET_TIMEOUT	xxxxxx	xxxxxx	x
2.1.6	RSM_SET_PACKET_DELAY	xxxxxx	xxxxxx	x
2.1.7	RSM_GET_PACKET_DELAY	xxxxxx	xxxxxx	x
2.1.8	RSM_SET_WORD_ORDER	xxxxxx	xxxxxx	x
2.1.9	RSM_GET_WORD_ORDER	xxxxxx	xxxxxx	x
3.2.1	RSM_RESET_CARD	0x0A0A	2570	1
3.2.2	RSM_CLEAR_CARD_BUFFER	0x0A0B	2571	1
3.3	RSM_SET_PULSE_MODE	0x0A0C	2572	3
3.3	RSM_MACRO_SET_PULSE_MODE	0x0C0C	3084	3
3.4	RSM_SET_MAX_V	0x0A0D	2573	4
3.4	RSM_MACRO_SET_MAX_V	0x0C0D	3085	4
3.5	RSM_SET_HLMT	0x0A0E	2574	4
3.5	RSM_MACRO_SET_HLMT	0x0C0E	3086	4
3.6	RSM_LIMITSTOP_MODE	0x0A0F	2575	3
3.6	RSM_MACRO_LIMITSTOP_MODE	0x0C0F	3087	3
3.7	RSM_SET_NHOME	0x0A10	2576	3
3.7	RSM_MACRO_SET_NHOME	0x0C10	3088	3
3.8	RSM_SET_HOME_EDGE	0x0A11	2577	3
3.8	RSM_MACRO_SET_HOME_EDGE	0x0C11	3089	3
3.9.1	RSM_SET_SLMT	0x0A12	2578	7

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
3.9.1	RSM_MACRO_SET_SLMT	0x0C12	3090	7
3.9.2	RSM_CLEAR_SLMT	0x0A13	2579	2
3.9.2	RSM_MACRO_CLEAR_SLMT	0x0C13	3091	2
3.10	RSM_SET_ENCODER	0x0A14	2580	5
3.10	RSM_MACRO_SET_ENCODER	0x0C14	3092	5
3.11.1	RSM_SERVO_ON	0x0A15	2581	2
3.11.1	RSM_MACRO_SERVO_ON	0x0C15	3093	2
3.11.2	RSM_SERVO_OFF	0x0A16	2582	2
3.11.2	RSM_MACRO_SERVO_OFF	0x0C16	3094	2
3.12	RSM_SET_ALARM	0x0A17	2583	4
3.12	RSM_MACRO_SET_ALARM	0x0C17	3095	4
3.13	RSM_SET_INPOS	0x0A18	2584	4
3.13	RSM_MACRO_SET_INPOS	0x0C18	3096	4
3.14	RSM_SET_FILTER	0x0A19	2585	4
3.14	RSM_MACRO_SET_FILTER	0x0C19	3097	4
3.15.1	RSM_VRING_ENABLE	0x0A1A	2586	4
3.15.1	RSM_MACRO_VRING_ENABLE	0x0C1A	3098	4
3.15.2	RSM_VRING_DISABLE	0x0A1B	2587	2
3.15.2	RSM_MACRO_VRING_DISABLE	0x0C1B	3099	2
3.16.1	RSM_AVTRI_ENABLE	0x0A1C	2588	2
3.16.1	RSM_MACRO_AVTRI_ENABLE	0x0C1C	3100	2
3.16.2	RSM_AVTRI_DISABLE	0x0A1D	2589	2
3.16.2	RSM_MACRO_AVTRI_DISABLE	0x0C1D	3101	2
3.17.1	RSM_EXD_MP	0x0A1E	2590	4
3.17.2	RSM_EXD_FP	0x0A1F	2591	4
3.17.3	RSM_EXD_CP	0x0A20	2592	4
3.17.4	RSM_EXD_DISABLE	0x0A21	2593	2
3.18.1	RSM_READ_bVAR	xxxxxx	xxxxxx	x
3.18.2	RSM_WRITE_bVAR	0x0A23	2595	3
3.18.3	RSM_READ_VAR	xxxxxx	xxxxxx	x
3.18.4	RSM_WRITE_VAR	0x0A25	2597	5
3.19.1	RSM_READ_MD	xxxxxx	xxxxxx	x
3.19.2	RSM_WRITE_MD	0x0A27	2599	7
4.1.1	RSM_SET_LP	0x0A28	2600	4
4.1.1	RSM_MACRO_SET_LP	0x0C28	3112	4
4.1.2	RSM_GET_LP	xxxxxx	xxxxxx	x

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
4.1.2	RSM_MACRO_GET_LP	0x0C29	3113	2
4.1.2	RSM_GET_LP_4_AXIS	xxxxxx	xxxxxx	x
4.2.1	RSM_SET_EP	0x0A2A	2602	4
4.2.1	RSM_MACRO_SET_EP	0x0C2A	3114	4
4.2.2	RSM_GET_EP	xxxxxx	xxxxxx	x
4.2.2	RSM_MACRO_GET_EP	0x0C2B	3115	2
4.2.2	RSM_GET_EP_4_AXIS	xxxxxx	xxxxxx	x
4.3.1	RSM_ABS_SET_POSITION	0x0AF4	2804	4
4.3.1	RSM_MACRO_ABS_SET_POSITION	0x0CF4	3316	4
4.3.2	RSM_ABS_GET_POSITION	xxxxxx	xxxxxx	x
4.3.2	RSM_MACRO_ABS_GET_POSITION	0x0CF5	3317	x
4.3.2	RSM_ABS_GET_POSITION_4_AXIS	xxxxxx	xxxxxx	x
4.4	RSM_GET_CV	xxxxxx	xxxxxx	x
4.4	RSM_GET_CV_4_AXIS	xxxxxx	xxxxxx	x
4.5	RSM_GET_CA	xxxxxx	xxxxxx	x
4.5	RSM_GET_CA_4_AXIS	xxxxxx	xxxxxx	x
4.6.1	RSM_MACRO_GET_DI	0x0C2E	3118	3
4.6.2	RSM_GET_DI_ALL	xxxxxx	xxxxxx	x
4.6.2	RSM_MACRO_GET_DI_ALL	0x0C31	3121	2
4.6.2	RSM_GET_DI_ALL_4_AXIS	xxxxxx	xxxxxx	x
4.7.1	RSM_MACRO_GET_DI_SIGNAL	0x0C40	3136	3
4.7.2	RSM_GET_DI_SIGNAL_ALL	xxxxxx	xxxxxx	x
4.7.2	RSM_MACRO_GET_DI_SIGNAL _ALL	0x0C41	3137	2
4.7.2	RSM_GET_DI_SIGNAL_ALL_4_AXIS	xxxxxx	xxxxxx	x
4.8	RSM_GET_HOME_SEARCH_STATE	xxxxxx	xxxxxx	x
4.8	RSM_MACRO_ GET_HOME_SEARCH_STATE	0x0C42	3138	2
4.8	RSM_GET_HOME_SEARCH_STATE _4_AXIS	xxxxxx	xxxxxx	x
4.9.1	RSM_GET_ERROR_STATE	xxxxxx	xxxxxx	x
4.9.1	RSM_MACRO_GET_ERROR	0x0C2F	3119	1
4.9.2	RSM_GET_ERROR_CODE	xxxxxx	xxxxxx	x
4.9.2	RSM_MACRO_GET_ERROR_CODE	0x0C30	3120	2
4.9.2	RSM_GET_ERROR_CODE_4_AXIS	xxxxxx	xxxxxx	x
4.10	RSM_GET_FREE_BUFFER	xxxxxx	xxxxxx	x
4.11	RSM_GET_STOP_STATUS	xxxxxx	xxxxxx	x

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
4.11	RSM_GET_STOP_STATUS_4_AXIS	xxxxxx	xxxxxx	x
4.12	RSM_GET_EMERGENCY_STATE	xxxxxx	xxxxxx	x
4.13	RSM_GET_DRIVING_AXIS	xxxxxx	xxxxxx	x
4.14	RSM_GET_INTERPOL_RDY_FLAG	xxxxxx	xxxxxx	x
4.15	RSM_GET_TRIG_INTFACTOR	xxxxxx	xxxxxx	x
4.16.2	RSM_GET_DEVICE_STATE	xxxxxx	xxxxxx	x
5.1.1	RSM_MACRO_FRNET_IN	0x0C32	3122	2
5.1.2	RSM_MACRO_FRNET_READ	0x0C34	3124	5
5.1.3	RSM_FRNET_READ_SINGLE_DIO	xxxxxx	xxxxxx	x
5.1.4	RSM_FRNET_READ_GROUP_DIO	xxxxxx	xxxxxx	x
5.1.5	RSM_FRNET_READ_MULTI_GROUP_DIO	xxxxxx	xxxxxx	x
5.2.1	RSM_MACRO_FRNET_OUT	0x0C33	3123	4
5.2.2	RSM_MACRO_FRNET_WRITE	0x0C35	3125	7
5.2.3	RSM_FRNET_WRITE_SINGLE_DO	xxxxxx	xxxxxx	x
5.2.4	RSM_FRNET_WRITE_GROUP_DO	xxxxxx	xxxxxx	x
5.2.5	RSM_FRNET_WRITE_MULTI_GROUP_DO	xxxxxx	xxxxxx	x
5.3	RSM_MACRO_FRNET_WAIT	0x0C36	3126	7
5.4	RSM_SET_FRNET_TRIGGER_EVENT	xxxxxx	xxxxxx	x
5.5	RSM_GET_FRNET_TRIGGER_EVENT_SETTING	xxxxxx	xxxxxx	x
6.1	RSM_SET_HV	0x0A3C	2620	4
6.1	RSM_MACRO_SET_HV	0x0C3C	3132	4
6.2	RSM_HOME_LIMIT	0x0A3D	2621	3
6.2	RSM_MACRO_HOME_LIMIT	0x0C3D	3133	3
6.3	RSM_SET_HOME_MODE	0x0A3E	2622	8
6.3	RSM_MACRO_SET_HOME_MODE	0x0C3E	3134	8
6.4	RSM_HOME_START	0x0A3F	2623	2
6.4	RSM_MACRO_HOME_START	0x0C3F	3135	2
7.1.1	RSM_NORMAL_SPEED	0x0A46	2630	3
7.1.1	RSM_MACRO_NORMAL_SPEED	0x0C46	3142	3
7.1.2	RSM_SET_SV	0x0A47	2631	4
7.1.2	RSM_MACRO_SET_SV	0x0C47	3143	4
7.1.3	RSM_SET_V	0x0A48	2632	4
7.1.3	RSM_MACRO_SET_V	0x0C48	3144	4
7.1.4	RSM_SET_A	0x0A49	2633	4

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
7.1.4	RSM_MACRO_SET_A	0x0C49	3145	4
7.1.5	RSM_SET_D	0x0A4A	2634	4
7.1.5	RSM_MACRO_SET_D	0x0C4A	3146	4
7.1.6	RSM_SET_K	0x0A4B	2635	4
7.1.6	RSM_MACRO_SET_K	0x0C4B	3147	4
7.1.7	RSM_SET_L	0x0A4C	2636	4
7.1.7	RSM_MACRO_SET_L	0x0C4C	3148	4
7.1.8	RSM_SET_AO	0x0A4D	2637	4
7.1.8	RSM_MACRO_SET_AO	0x0C4D	3149	4
7.1.9	RSM_FIXED_MOVE	0x0A4E	2638	4
7.1.9	RSM_MACRO_FIXED_MOVE	0x0C4E	3150	4
7.1.10	RSM_SET_PULSE	0x0A4F	2639	4
7.1.10	RSM_MACRO_SET_PULSE	0x0C4F	3151	4
7.1.11	RSM_ABS_FIXED_MOVE	0x0AF6	2806	4
7.1.11	RSM_ABS_MACRO_FIXED_MOVE	0x0CF6	3318	4
7.1.12	RSM_CONTINUE_MOVE	0x0A50	2640	4
7.1.12	RSM_MACRO_CONTINUE_MOVE	0x0C50	3152	4
7.2.1	RSM_AXIS_ASSIGN	0x0A5A	2650	4
7.2.1	RSM_MACRO_AXIS_ASSIGN	0x0C5A	3162	4
7.2.2	RSM_VECTOR_SPEED	0x0A5B	2651	2
7.2.2	RSM_MACRO_VECTOR_SPEED	0x0C5B	3163	2
7.2.3	RSM_SET_VSV	0x0A5C	2652	3
7.2.3	RSM_MACRO_SET_VSV	0x0C5C	3164	3
7.2.4	RSM_SET_VV	0x0A5D	2653	3
7.2.4	RSM_MACRO_SET_VV	0x0C5D	3165	3
7.2.5	RSM_SET_VA	0x0A5E	2654	3
7.2.5	RSM_MACRO_SET_VA	0x0C5E	3166	3
7.2.6	RSM_SET_VD	0x0A5F	2655	3
7.2.6	RSM_MACRO_SET_VD	0x0C5F	3167	3
7.2.7	RSM_SET_VK	0x0A60	2656	3
7.2.7	RSM_MACRO_SET_VK	0x0C60	3168	3
7.2.8	RSM_SET_VL	0x0A61	2657	3
7.2.8	RSM_MACRO_SET_VL	0x0C61	3169	3
7.2.9	RSM_SET_VAO	0x0A62	2658	3
7.2.9	RSM_MACRO_SET_VAO	0x0C62	3170	3
7.2.10	RSM_LINE_2D	0x0A63	2659	5

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
7.2.10	RSM_MACRO_LINE_2D	0x0C63	3171	5
7.2.11	RSM_ABS_LINE_2D	0x0AF7	2807	5
7.2.11	RSM_ABS_MACRO_LINE_2D	0x0CF7	3319	5
7.2.12	RSM_LINE_3D	0x0A64	2660	7
7.2.12	RSM_MACRO_LINE_3D	0x0C64	3172	7
7.2.13	RSM_ABS_LINE_3D	0x0AF8	2808	7
7.2.13	RSM_ABS_MACRO_LINE_3D	0x0CF8	3320	7
7.2.14	RSM_ARC_CW	0x0A65	2661	9
7.2.14	RSM_MACRO_ARC_CW	0x0C65	3173	9
7.2.15	RSM_ARC_CCW	0x0A67	2663	9
7.2.15	RSM_MACRO_ARC_CCW	0x0C67	3175	9
7.2.16	RSM_ABS_ARC_CW	0x0AF9	2809	9
7.2.16	RSM_ABS_MACRO_ARC_CW	0x0CF9	3321	9
7.2.17	RSM_ABS_ARC_CCW	0x0AFA	2810	9
7.2.17	RSM_ABS_MACRO_ARC_CCW	0x0CFA	3322	9
7.2.18	RSM_CIRCLE_CW	0x0A69	2665	5
7.2.18	RSM_MACRO_CIRCLE_CW	0x0C69	3177	5
7.2.19	RSM_CIRCLE_CCW	0x0A6A	2666	5
7.2.19	RSM_MACRO_CIRCLE_CCW	0x0C6A	3178	5
7.3.1	RSM_SYNC_ACTION	0x0A6E	2670	14
7.3.1	RSM_MACRO_SYNC_ACTION	0x0C6E	3182	14
7.3.2	RSM_CLEAR_SYNC_ACTION	0x0A95	2709	2
7.3.2	RSM_MACRO_CLEAR_SYNC_ACTION	0x0C95	3221	2
7.3.3	RSM_SET_ACTIVATION_FACTORS	0x0A96	2710	4
7.3.3	RSM_MACRO_SET_ACTIVATION_FACTORS	0x0C96	3222	4
7.3.4	RSM_SET_ACTIVATION_AXIS	0x0A97	2711	3
7.3.4	RSM_MACRO_SET_ACTIVATION_AXIS	0x0C97	3223	3
7.3.5	RSM_SET_ACTION	0x0A98	2712	11
7.3.5	RSM_MACRO_SET_ACTION	0x0C98	3224	11
7.3.6	RSM_SET_COMPARE	0x0A70	2672	6
7.3.6	RSM_MACRO_SET_COMPARE	0x0C70	3184	6
7.3.7	RSM_GET_LATCH	xxxxxx	xxxxxx	x
7.3.7	RSM_MACRO_GET_LATCH	0x0C71	3185	2
7.3.7	RSM_GET_LATCH_4_AXIS	xxxxxx	xxxxxx	x
7.3.8	RSM_SET_PRESET	0x0A72	2674	4

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
7.3.8	RSM_MACRO_SET_PRESET	0x0C72	3186	4
7.3.9	RSM_SET_OUT	0x0A73	2675	4
7.3.9	RSM_MACRO_SET_OUT	0x0C73	3187	4
7.3.10	RSM_ENABLE_INT	0x0AAA	2730	1
7.3.10	RSM_MACRO_ENABLE_INT	0x0CAA	3242	1
7.3.11	RSM_DISABLE_INT	0x0AAB	2731	1
7.3.11	RSM_MACRO_DISABLE_INT	0x0CAB	3243	1
7.3.12	RSM_INTFACTOR_ENABLE	0x0AAC	2732	4
7.3.12	RSM_MACRO_INTFACTOR_ENABLE	0x0CAC	3244	4
7.3.13	RSM_INTFACTOR_DISABLE	0x0AAD	2733	3
7.3.13	RSM_MACRO_INTFACTOR_DISABLE	0x0CAD	3245	3
7.4.1	RSM_RECTANGLE	0x0A78	2680	20
7.4.1	RSM_MACRO_RECTANGLE	0x0C78	3192	20
7.4.2	RSM_LINE_2D_INITIAL	0x0A7C	2684	9
7.4.2	RSM_MACRO_LINE_2D_INITIAL	0x0C7C	3196	9
7.4.3	RSM_LINE_2D_CONTINUE	0x0A7E	2686	6
7.4.3	RSM_MACRO_LINE_2D_CONTINUE	0x0C7E	3198	6
7.4.4	RSM_ABS_LINE_2D_CONTINUE	0x0AFB	2811	6
7.4.4	RSM_ABS_MACRO_LINE_2D_CONTINUE	0x0CFB	3323	6
7.4.5	RSM_LINE_3D_INITIAL	0x0A7F	2687	10
7.4.5	RSM_MACRO_LINE_3D_INITIAL	0x0C7F	3199	10
7.4.6	RSM_LINE_3D_CONTINUE	0x0A81	2689	8
7.4.6	RSM_MACRO_LINE_3D_CONTINUE	0x0CFC	3324	8
7.4.7	RSM_ABS_LINE_3D_CONTINUE	0x0AFC	2812	8
7.4.7	RSM_ABS_MACRO_LINE_3D_CONTINUE	0x0C81	3201	8
7.4.8	RSM_MIX_2D_INITIAL	0x0A82	2690	10
7.4.8	RSM_MACRO_MIX_2D_INITIAL	0x0C82	3202	10
7.4.9	RSM_MIX_2D_CONTINUE	0x0AFD	2813	11
7.4.9	RSM_MACRO_MIX_2D_CONTINUE	0x0CFD	3325	11
7.4.10	RSM_ABS_MIX_2D_CONTINUE	0x0A84	2692	11
7.4.10	RSM_ABS_MACRO_MIX_2D_CONTINUE	0x0C84	3204	11
7.4.11	RSM_HELIX_3D	0x0A88	2696	15
7.4.11	RSM_MACRO_HELIX_3D	0x0C88	3208	15
7.4.12	RSM_LINE_SCAN	0x0A90	2704	7
7.4.13	RSM_LINE_SCAN_START	0x0A91	2705	5

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
7.4.14	RSM_GET_LINE_SCAN_DONE	xxxxxx	xxxxxx	x
7.5.1	RSM_DRV_HOLD	0x0AB4	2740	2
7.5.1	RSM_MACRO_DRV_HOLD	0x0CB4	3252	2
7.5.2	RSM_DRV_START	0x0AB5	2741	2
7.5.2	RSM_MACRO_DRV_START	0x0CB5	3253	2
7.5.3	RSM_STOP_SLOWLY	0x0AB7	2743	2
7.5.3	RSM_MACRO_STOP_SLOWLY	0x0CB7	3255	2
7.5.4	RSM_STOP_SUDDENLY	0x0AB8	2744	2
7.5.4	RSM_MACRO_STOP_SUDDENLY	0x0CB8	3256	2
7.5.5	RSM_VSTOP_SLOWLY	0x0AB9	2745	1
7.5.5	RSM_MACRO_VSTOP_SLOWLY	0x0CB9	3257	1
7.5.6	RSM_VSTOP_SUDDENLY	0x0ABA	2746	1
7.5.6	RSM_MACRO_VSTOP_SUDDENLY	0x0CBA	3258	1
7.5.7	RSM_CLEAR_STOP	0x0ABB	2747	2
7.5.7	RSM_MACRO_CLEAR_STOP	0x0CBB	3259	2
7.5.8	RSM_CLEAR_VSTOP	0x0A09	2569	1
7.5.8	RSM_MACRO_CLEAR_VSTOP	0x0C09	3081	1
7.5.9	RSM_INTP_END	0x0ABC	2748	2
7.5.9	RSM_MACRO_INTP_END	0x0CBC	3260	2
7.5.10	RSM_EMERGENCY_STOP	0x0A04	2564	2
7.5.11	RSM_CLEAR_EMERGENCY_STOP	0x0A05	2565	1
8.1	RSM_LOAD_INITIAL	0x0AC8	2760	1
9.1.1	RSM_MP_CREATE	0x0AC9	2761	2
9.1.2	RSM_MACRO_MP_CLOSE	0x0CCA	3274	1
9.1.3	RSM_MP_CALL	0x0ACB	2763	2
9.1.3	RSM_MACRO_MP_CALL	0x0CCB	3275	2
9.2.1	RSM_MP_ISR_CREATE	0x0ACD	2765	2
9.2.2	RSM_MACRO_MP_ISR_CLOSE	0x0CCE	3278	1
9.2.3	RSM_MP_ISR_CALL	0x0ACF	2767	2
9.3.1	RSM_MACRO_SET_VAR	0x0CD2	3282	5
9.3.2	RSM_MACRO_SET_RVAR	0x0CD3	3283	3
9.4	RSM_MACRO_VAR_CALCULATE	0x0CD4	3284	8
9.5.1	RSM_MACRO_FOR	0x0CD5	3285	3
9.5.2	RSM_MACRO_NEXT	0x0CD6	3286	1
9.5.3	RSM_MACRO_EXIT_FOR	0x0CDE	3294	1
9.6.1	RSM_MACRO_IF	0x0CD7	3287	6

Section	Sub_Function	Code (Hex)	Code (Dec)	No. of registers
		reg(0)	reg(0)	
9.6.2	RSM_MACRO_ELSE	0x0CD8	3288	1
9.6.3	RSM_MACRO_END_IF	0x0CD9	3289	1
9.7.1	RSM_MACRO_GOTO	0x0CDF	3295	2
9.7.2	RSM_MACRO_LABEL	0x0CE1	3297	2
9.8	RSM_MACRO_TIMER	0x0CDA	3290	3
9.9	RSM_STOP_WAIT	0x0ADB	2779	2
9.9	RSM_MACRO_STOP_WAIT	0x0CDB	3291	2
9.10	RSM_MACRO_EXIT_MACRO	0x0CDD	3293	1
9.11	RSM_MP_TERMINATE	0x0AE2	2786	1
9.11	RSM_MACRO_MP_TERMINATE	0x0CE2	3298	1
9.12	RSM_GET_MP_DOWNLOAD_STATUS	xxxxxx	xxxxxx	x
9.13	RSM_CHANGE_VEL_ON_FLY	0x0A43	2627	4
11.1	RSM_GET_RSM_FIRMWARE_VERSION	xxxxxx	xxxxxx	x
11.2	RSM_GET_i8094H_FIRMWARE_VERSION	xxxxxx	xxxxxx	x
11.3	RSM_GET_DLL_VERSION	xxxxxx	xxxxxx	x

12.4 MODBUS Coil Table

The coil table can be accessed with the following Modbus functions:
- Read Coils (FC 01)
- Write Single Coil (FC 05)
- Write Multiple Coils (FC 15)

The table addresses defined below is zero-based. For some PLC or HMI, the addresses are shown on their documents as 0xxxx; x is digital number (for example 00001 to 09999). The first '0' represents the table is defined for coils; and the following xxxx is the address of a coil.

Most PLC systems use the one-based system which means the Modbus table starts at address 1. The tables defined in this manual are zero-based addresses, which means the table address starts at zero. Therefore the one-based system needs to use the address 1 to access the table at address 0.

12.4.1 FRnet Digital Output

The starting address is 0 (zero-based). Please use FC=05, 15 to write the FRnet DO status:

(1) FC=05. It can be used to write single point FRnet DO (0~127).

(2) FC=15. It can be used to write multi-point FRnet DO. These DO do not have to be limited to the same group. For example: the user can choose to write DO points from 5~45 (total 41 points).

Appendix table 22: FRnet Digital Output (PLC addresses: 00001 to 00128)

FRnet Digital Output				
Group	Channel	Address (zero-based)	Type	Comment
0	0	0	R/W	
	1	1	R/W	
	2	2	R/W	
	3	3	R/W	
	4	4	R/W	
	5	5	R/W	
	6	6	R/W	
	7	7	R/W	
	8	8	R/W	
	9	9	R/W	
	10	10	R/W	
	11	11	R/W	
	12	12	R/W	
	13	13	R/W	
	14	14	R/W	
	15	15	R/W	
1	0	16	R/W	
	1	17	R/W	
	2	18	R/W	
	3	19	R/W	
	4	20	R/W	
	5	21	R/W	
	6	22	R/W	
	7	23	R/W	

	8	24	R/W	
	9	25	R/W	
	10	26	R/W	
	11	27	R/W	
	12	28	R/W	
	13	29	R/W	
	14	30	R/W	
	15	31	R/W	
2	0	32	R/W	
	1	33	R/W	
	2	34	R/W	
	3	35	R/W	
	4	36	R/W	
	5	37	R/W	
	6	38	R/W	
	7	39	R/W	
	8	40	R/W	
	9	41	R/W	
	10	42	R/W	
	11	43	R/W	
	12	44	R/W	
	13	45	R/W	
	14	46	R/W	
	15	47	R/W	
3	0	48	R/W	
	1	49	R/W	
	2	50	R/W	
	3	51	R/W	
	4	52	R/W	
	5	53	R/W	
	6	54	R/W	
	7	55	R/W	
	8	56	R/W	
	9	57	R/W	
	10	58	R/W	
	11	59	R/W	
	12	60	R/W	

	13	61	R/W	
	14	62	R/W	
	15	63	R/W	
4	0	64	R/W	
	1	65	R/W	
	2	66	R/W	
	3	67	R/W	
	4	68	R/W	
	5	69	R/W	
	6	70	R/W	
	7	71	R/W	
	8	72	R/W	
	9	73	R/W	
	10	74	R/W	
	11	75	R/W	
	12	76	R/W	
	13	77	R/W	
	14	78	R/W	
	15	79	R/W	
5	0	80	R/W	
	1	81	R/W	
	2	82	R/W	
	3	83	R/W	
	4	84	R/W	
	5	85	R/W	
	6	86	R/W	
	7	87	R/W	
	8	88	R/W	
	9	89	R/W	
	10	90	R/W	
	11	91	R/W	
	12	92	R/W	
	13	93	R/W	
	14	94	R/W	
	15	95	R/W	
6	0	96	R/W	
	1	97	R/W	

	2	98	R/W	
	3	99	R/W	
	4	100	R/W	
	5	101	R/W	
	6	102	R/W	
	7	103	R/W	
	8	104	R/W	
	9	105	R/W	
	10	106	R/W	
	11	107	R/W	
	12	108	R/W	
	13	109	R/W	
	14	110	R/W	
	15	111	R/W	
7	0	112	R/W	
	1	113	R/W	
	2	114	R/W	
	3	115	R/W	
	4	116	R/W	
	5	117	R/W	
	6	118	R/W	
	7	119	R/W	
	8	120	R/W	
	9	121	R/W	
	10	122	R/W	
	11	123	R/W	
	12	124	R/W	
	13	125	R/W	
	14	126	R/W	
	15	127	R/W	

12.4.2 Servo On/Off

Appendix table 23: Servo On/Off (PLC addresses: 00129 to 00132)

Servo State			
Axis	Address (zero based)	Type	Comment
X	128	R	1: Servo on 0: Servo off
Y	129	R	1: Servo on 0: Servo off
Z	130	R	1: Servo on 0: Servo off
U	131	R	1: Servo on 0: Servo off

12.5 MODBUS Discrete Input Table

The discrete input table can be accessed by the following Modbus functions:
– Read discrete input (FC 02)

The table addresses defined below is zero-based. For some PLC or HMI, the addresses maybe show on their documents as 1xxxx, x is digital number. For example: 10001 to 19999. The first '1' represents the table is defined for discrete input; and the following xxxx is the address of a discrete input.

In PLC, the first discrete input defined at address 10001 is the same as the first register at address 0 shown in the following table. Most PLC systems which addresses do not start from zero can be referred as one-based system. The following tables will only show zero-based addresses. When PLC or HMI users configure their discrete input numbers, they have to consider discrete input number mapping problem carefully.

12.5.1 FRnet Digital Input

Appendix table 24: FRnet Digital Inputs (PLC addresses: 10001 to 10128)

FRnet Digital Input				
Group	Channel	Address (zero based)	Type	Comment
8	0	0	R	
	1	1	R	
	2	2	R	
	3	3	R	
	4	4	R	
	5	5	R	
	6	6	R	
	7	7	R	
	8	8	R	
	9	9	R	
	10	10	R	
	11	11	R	
	12	12	R	
	13	13	R	
	14	14	R	
	15	15	R	
9	0	16	R	
	1	17	R	
	2	18	R	
	3	19	R	
	4	20	R	
	5	21	R	
	6	22	R	
	7	23	R	
	8	24	R	
	9	25	R	
	10	26	R	
	11	27	R	
	12	28	R	

	13	29	R	
	14	30	R	
	15	31	R	
10	0	32	R	
	1	33	R	
	2	34	R	
	3	35	R	
	4	36	R	
	5	37	R	
	6	38	R	
	7	39	R	
	8	40	R	
	9	41	R	
	10	42	R	
	11	43	R	
	12	44	R	
	13	45	R	
	14	46	R	
	15	47	R	
11	0	48	R	
	1	49	R	
	2	50	R	
	3	51	R	
	4	52	R	
	5	53	R	
	6	54	R	
	7	55	R	
	8	56	R	
	9	57	R	
	10	58	R	
	11	59	R	
	12	60	R	
	13	61	R	
	14	62	R	
	15	63	R	

12	0	64	R	
	1	65	R	
	2	66	R	
	3	67	R	
	4	68	R	
	5	69	R	
	6	70	R	
	7	71	R	
	8	72	R	
	9	73	R	
	10	74	R	
	11	75	R	
	12	76	R	
	13	77	R	
	14	78	R	
	15	79	R	
13	0	80	R	
	1	81	R	
	2	82	R	
	3	83	R	
	4	84	R	
	5	85	R	
	6	86	R	
	7	87	R	
	8	88	R	
	9	89	R	
	10	90	R	
	11	91	R	
	12	92	R	
	13	93	R	
	14	94	R	
	15	95	R	
14	0	96	R	
	1	97	R	
	2	98	R	

	3	99	R	
	4	100	R	
	5	101	R	
	6	102	R	
	7	103	R	
	8	104	R	
	9	105	R	
	10	106	R	
	11	107	R	
	12	108	R	
	13	109	R	
	14	110	R	
	15	111	R	
15	0	112	R	
	1	113	R	
	2	114	R	
	3	115	R	
	4	116	R	
	5	117	R	
	6	118	R	
	7	119	R	
	8	120	R	
	9	121	R	
	10	122	R	
	11	123	R	
	12	124	R	
	13	125	R	
	14	126	R	
	15	127	R	

12.5.2 DI or Status of Control Board

Appendix table 25: Motion chip status triggered by DI of Control Board (PLC addresses: 10129 to 10192)

Read DI or Status of Control Board				
Axis	DI	Address (zero based)	Type	Comment
X	DRIVING	128	R	The driving state of AXIS-X: 1: driving, 0: stop
	LIMIT+	129	R	The hardware limit (LMT+) state of AXIS-X: 0: off, 1: on
	LIMIT-	130	R	The hardware limit (LMT-) state of AXIS-X: 0: off, 1: on
	EMERGENCY	131	R	The emergency (EMG) state of AXIS-X: 0: off, 1: on
	ALARM	132	R	The ALARM state of AXIS-X: (please enable Alarm with RSM_SET_ALARM() 0: off, 1: on
	HOME	133	R	The HOME/IN1 state of AXIS-X: 0: on, 1: off
	NHOME	134	R	The NHOME/IN0 state of AXIS-X: 0: on, 1: off
	IN3	135	R	The IN3 state of AXIS-X: 0: on, 1: off
	INPOS	136	R	The INPOS or Servo Ready state of AXIS-X: 0: on, 1: off
	INDEX	137	R	The Z-phase/IN2 state of AXIS-X: 0: on, 1: off
		138		undefined
		139		
		140		
		141		
		142		
		143		
Y	DRIVING	144	R	The driving state of AXIS-Y: 1: driving, 0: stop
	LIMIT+	145	R	The hardware limit (LMT+) state of AXIS-Y: 0: off, 1: on
	LIMIT-	146	R	The hardware limit (LMT-) state of AXIS-Y: 0: off, 1: on
	EMERGENCY	147	R	The emergency (EMG) state of AXIS-Y: 0: off, 1: on

	ALARM	148	R	The ALARM state of AXIS- Y: (please enable Alarm with RSM_SET_ALARM() 0: off, 1: on
	HOME	149	R	The HOME/IN1 state of AXIS- Y: 0: on, 1: off
	NHOME	150	R	The NHOME/IN0 state of AXIS- Y: 0: on, 1: off
	IN3	151	R	The IN3 state of AXIS- Y: 0: on, 1: off
	INPOS	152	R	The INPOS or Servo Ready state of AXIS-X: 0: on, 1: off
	INDEX	153	R	The Z-phase/IN2 state of AXIS- Y: 0: on, 1: off
		154		undefined
		155		
		156		
		157		
		158		
		159		
Z	DRIVING	160	R	The driving state of AXIS-Z: 1: driving, 0: stop
	LIMIT+	161	R	The hardware limit (LMT+) state of AXIS-Z: 0: off, 1: on
	LIMIT-	162	R	The hardware limit (LMT-) state of AXIS-Z: 0: off, 1: on
	EMERGENCY	163	R	The emergency (EMG) state of AXIS-Z: 0: off, 1: on
	ALARM	164	R	The ALARM state of AXIS-Z: (please enable Alarm with RSM_SET_ALARM() 0: off, 1: on
	HOME	165	R	The HOME/IN1 state of AXIS-Z: 0: on, 1: off
	NHOME	166	R	The NHOME/IN0 state of AXIS-Z: 0: on, 1: off
	IN3	167	R	The IN3 state of AXIS-Z: 0: on, 1: off
	INPOS	168	R	The INPOS or Servo Ready state of AXIS-Z: 0: on, 1: off
	INDEX	169	R	The Z-phase/IN2 state of AXIS-Z: 0: on, 1: off
		170		undefined

		171		
		172		
		173		
		174		
		175		
U	DRIVING	176	R	The driving state of AXIS-U: 1: driving, 0: stop
	LIMIT+	177	R	The hardware limit (LMT+) state of AXIS-U: 0: off, 1: on
	LIMIT-	178	R	The hardware limit (LMT-) state of AXIS-U: 0: off, 1: on
	EMERGENCY	179	R	The emergency (EMG) state of AXIS-U: 0: off, 1: on
	ALARM	180	R	The ALARM state of AXIS-U: (please enable Alarm with RSM_SET_ALARM() 0: off, 1: on
	HOME	181	R	The HOME/IN1 state of AXIS-U: 0: on, 1: off
	NHOME	182	R	The NHOME/IN0 state of AXIS-U: 0: on, 1: off
	IN3	183	R	The IN3 state of AXIS-U: 0: on, 1: off
	INPOS	184	R	The INPOS or Servo Ready state of AXIS-U: 0: on, 1: off
	INDEX	185	R	The Z-phase/IN2 state of AXIS-U: 0: on, 1: off
		186		undefined
		187		
		188		
		189		
		190		
		191		

12.5.3 Error Stop States

Appendix table 26: Error Stop States (PLC addresses: 10193 to 10256)

Read Motion Error States				
Axis	Cause of stop	Modbus Address	Type	Comment
X	SOFT LIMIT+	192	R	Forward **software** limit of AXIS-X has been encountered
	SOFT LIMIT-	193	R	Reverse **software** limit of AXIS-X has been encountered
	LIMIT+	194	R	Forward **hardware** limit of AXIS-X has been encountered
	LIMIT-	195	R	Reverse **hardware** limit of AXIS-X has been encountered
	ALARM	196	R	The ALARM state of AXIS-X has been activated
	EMERGENCY	197	R	The emergency state of AXIS-X has been activated
	Reserved	198	R	Not defined
	HOME	199	R	Z phase and HOME position of AXIS-X have been reached
	Stop Command	200	R	Motion has been stopped by STOP_SLOWLY STOP_SUDDENLY VSTOP_SLOWLY VSTOP_SUDDENLY
		201		undefined
		202		
		203		
		204		
		205		
		206		
		207		
Y	SOFT LIMIT+	208	R	Forward **software** limit of AXIS-Y has been encountered
	SOFT LIMIT-	209	R	Reverse **software** limit of AXIS-Y has been encountered
	LIMIT+	210	R	Forward **hardware** limit of AXIS-Y has been encountered

	LIMIT-	211	R	Reverse **hardware** limit of AXIS-Y has been encountered
	ALARM	212	R	The ALARM state of AXIS-Y has been activated
	EMERGENCY	213	R	The emergency state of AXIS-Y has been activated
	Reserved	214	R	Not defined
	HOME	215	R	Z phase and HOME position of AXIS-Y have been reached
	Stop Command	216	R	Motion has been stopped by STOP_SLOWLY STOP_SUDDENLY VSTOP_SLOWLY VSTOP_SUDDENLY
		217		undefined
		218		
		219		
		220		
		221		
		222		
		223		
Z	SOFT LIMIT+	224	R	Forward **software** limit of AXIS-Z has been encountered
	SOFT LIMIT-	225	R	Reverse **software** limit of AXIS-Z has been encountered
	LIMIT+	226	R	Forward **hardware** limit of AXIS-Z has been encountered
	LIMIT-	227	R	Reverse **hardware** limit of AXIS-Z has been encountered
	ALARM	228	R	The ALARM state of AXIS-Z has been activated
	EMERGENCY	229	R	The emergency state of AXIS-Z has been activated
	Reserved	230	R	Not defined
	HOME	231	R	Z phase and HOME position of AXIS-Z have been reached
	Stop Command	232	R	Motion has been stopped by STOP_SLOWLY STOP_SUDDENLY VSTOP_SLOWLY VSTOP_SUDDENLY

		233		undefined
		234		
		235		
		236		
		237		
		238		
		239		
U	SOFT LIMIT+	240	R	Forward **software** limit of AXIS-U has been encountered
	SOFT LIMIT-	241	R	Reverse **software** limit of AXIS-U has been encountered
	LIMIT+	242	R	Forward **hardware** limit of AXIS-U has been encountered
	LIMIT-	243	R	Reverse **hardware** limit of AXIS-U has been encountered
	ALARM	244	R	The ALARM state of AXIS-U has been activated
	EMERGENCY	245	R	The emergency state of AXIS-U has been activated
	Reserved	246	R	Not defined
	HOME	247	R	Z phase and HOME position of AXIS-U have been reached
	Stop Command	248	R	Motion has been stopped by STOP_SLOWLY STOP_SUDDENLY VSTOP_SLOWLY VSTOP_SUDDENLY
		249		undefined
		250		
		251		
		252		
		253		
		254		
		255		

12.5.4 DI Signal of Control Board

Appendix Table 27: DI or Status of Control Board (PLC addresses: 10129 to 10192)

Read DI or Status of Control Board				
Axis	DI	Address (zero based)	Type	Comment
X	Near Home	256	R	The NHOME/IN0 signal of AXIS-X: 0: on, 1: off
	Home	257	R	The HOME/IN1 signal of AXIS-X: 0: on, 1: off
	Index	258	R	The Index/IN2 signal of AXIS-X: 0: on, 1: off
	IN3	259	R	The IN3 state of AXIS-X: 0: on, 1: off
	MPG (+)	260	R	EXP+ (External input) signal of AXIS-X: 0: off, 1: on (used for driving by external pulses in positive direction
	MPG (-)	261	R	EXP- (External input) signal of AXIS-X: 0: off, 1: on (used for driving by external pulses in negative direction)
	InPos	262	R	INP signal of AXIS-X: 0: off, 1: on
	Alarm	263	R	The ALARM signal of AXIS-X: 0: off, 1: on
Y	Near Home	264	R	The NHOME/IN0 signal of AXIS-Y: 0: on, 1: off
	Home	265	R	The HOME/IN1 signal of AXIS-Y: 0: on, 1: off
	Index	266	R	The Index/IN2 signal of AXIS-Y: 0: on, 1: off
	IN3	267	R	The IN3 state of AXIS-Y: 0: on, 1: off
	MPG (+)	268	R	EXP+ (External input) signal of AXIS-Y: 0: off, 1: on (used for driving by external pulses in positive direction
	MPG (-)	269	R	EXP- (External input) signal of AXIS-Y: 0: off, 1: on (used for driving by external pulses in negative direction)
	InPos	270	R	INP signal of AXIS-Y: 0: off, 1: on
	Alarm	271	R	The ALARM signal of AXIS-Y: 0: off, 1: on
Z	Near Home	272	R	The NHOME/IN0 signal of AXIS-Z: 0: on, 1: off

	Home	273	R	The HOME/IN1 signal of AXIS-Z: 0: on, 1: off
	Index	274	R	The Index/IN2 signal of AXIS-Z: 0: on, 1: off
	IN3	275	R	The IN3 state of AXIS-Z: 0: on, 1: off
	MPG (+)	276	R	EXP+ (External input) signal of AXIS-Z: 0: off, 1: on (used for driving by external pulses in positive direction
	MPG (-)	277	R	EXP- (External input) signal of AXIS-Z: 0: off, 1: on (used for driving by external pulses in negative direction)
	InPos	278	R	INP signal of AXIS-Z: 0: off, 1: on
	Alarm	279	R	The ALARM signal of AXIS-Z: 0: off, 1: on
U	Near Home	380	R	The NHOME/IN0 signal of AXIS-U: 0: on, 1: off
	Home	381	R	The HOME/IN1 signal of AXIS-U: 0: on, 1: off
	Index	382	R	The Index/IN2 signal of AXIS-U: 0: on, 1: off
	IN3	383	R	The IN3 state of AXIS-U: 0: on, 1: off
	MPG (+)	384	R	EXP+ (External input) signal of AXIS-U: 0: off, 1: on (used for driving by external pulses in positive direction
	MPG (-)	385	R	EXP- (External input) signal of AXIS-U: 0: off, 1: on (used for driving by external pulses in negative direction)
	InPos	386	R	INP signal of AXIS-U: 0: off, 1: on
	Alarm	387	R	The ALARM signal of AXIS-U: 0: off, 1: on

13 RS-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 RS-M8194H via RS-232.
	Off	No data is sent by the RS-M8194H via RS-232.
Rx	Flashing	The device is receiving data via RS-232.
	Off	No data is being received.
NET		
	Flashing	Data is being transmitted.
	Off	No data is being transmitted.
MOD	On	Module i-8094H is plugged into RS-M8194H device.
	Flashing	A module different than i-8094H is plugged into RS-M8194H device.
	Off	No module is plugged into the RS-M8194H device.

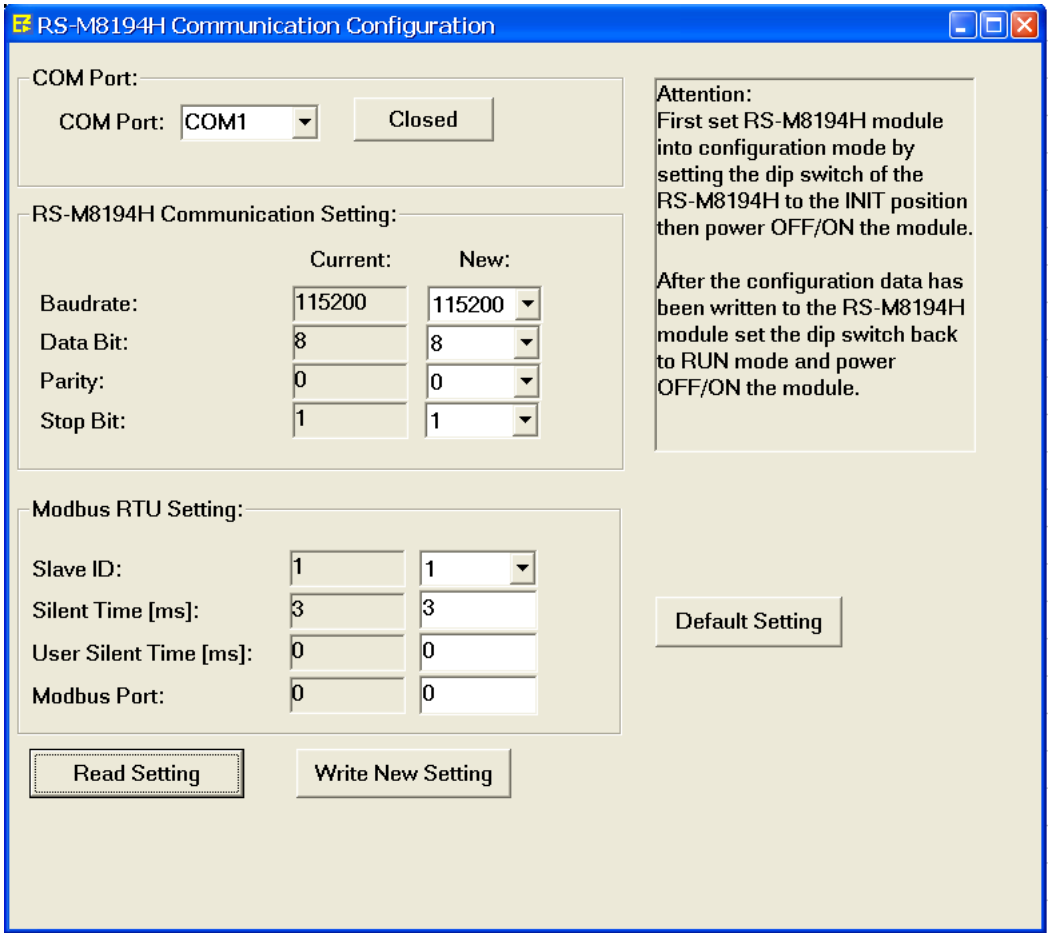
LED description of the i-8094H module:

- P is the power indicator,
- A is the FRnet indicator, and
- D is the pulse output indicator.

14 EzMove Utility

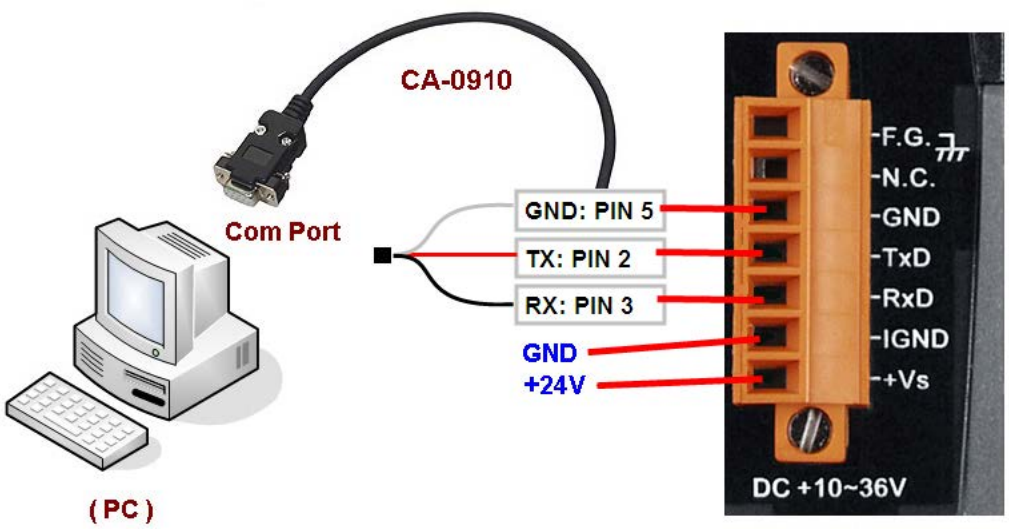
14.1 Serial Communication Parameter Settings

Execute the EzMove Utility, and click Menu [Setting] –> [RS-M8194H Setting] –> [By COM Port] –> [Communication Configuration], to open the following window:



1. Power off the RS-M8194H.

2. Connect the RS-232 serial port of the PC to the RS-M8194H device by using the RS-232 cable (CA-0910). The Tx, Rx and GND pins of the CA-0910 cable have to be connected to the corresponding Rx, Tx and GND ports of COM1 of the RS-M8194H. The 9-pin, D-sub connector has to be plugged into the RS232 COM port of the desktop/laptop.



3. Set the DIP-switch to “Init”, then power up the RS-M8194H.

(Dip Switch -- Init)

4. Select on the configuration window the COM Port to which the RS-M8194H is connected to and then click the Open button.

5. Click the Read Setting button to read the current serial configuration of the device.

6. Enter your new serial parameter setting.

7. Click the Write New Setting button to download the parameter setting to the RS-M8194H.
To restore the factory default setting just click Default Setting and Write New Setting.

8. **Set the DIP-switch back to the “Run” position and power off/on the RS-M8194H.**

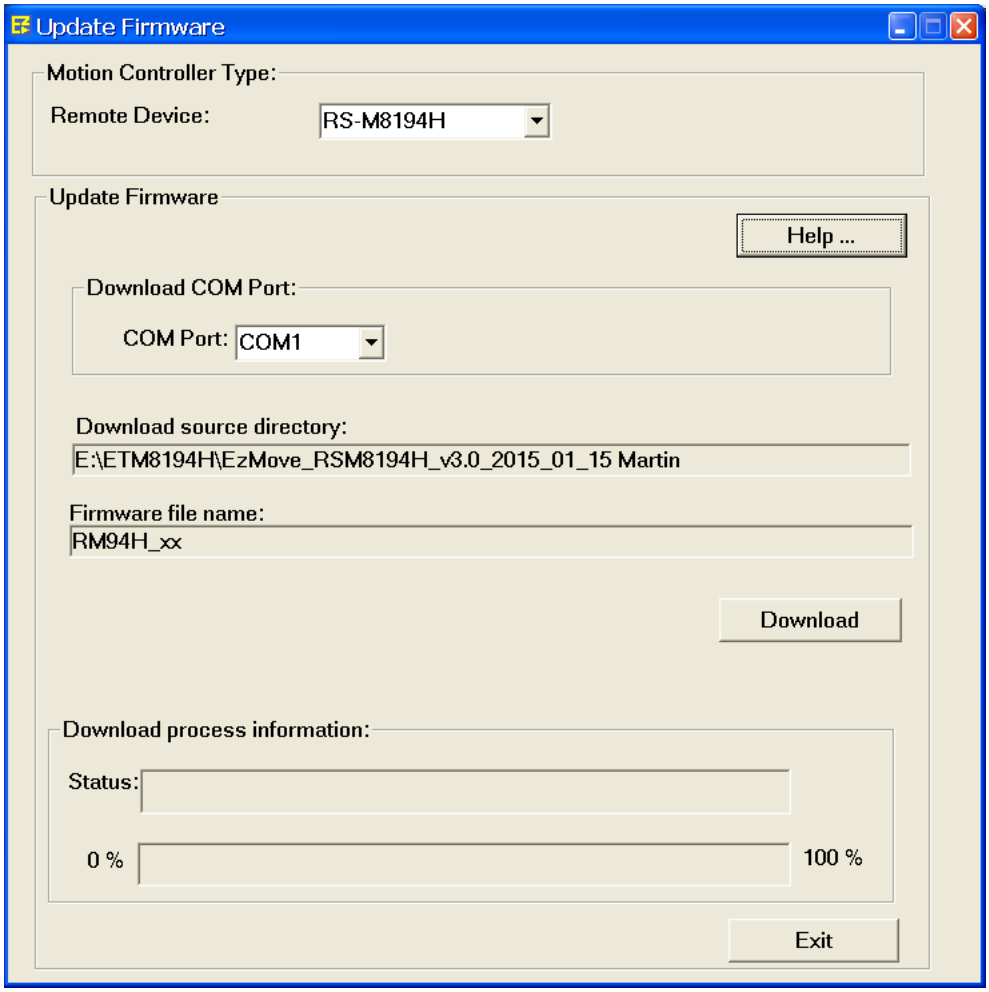
(Dip Switch -- Run)

ATTENTION!!!

Remove the RS-232 cable (CA-0910) from the RS-M8194H after configuration to prevent the device to be affected by noises.

14.2 Firmware Update Procedure

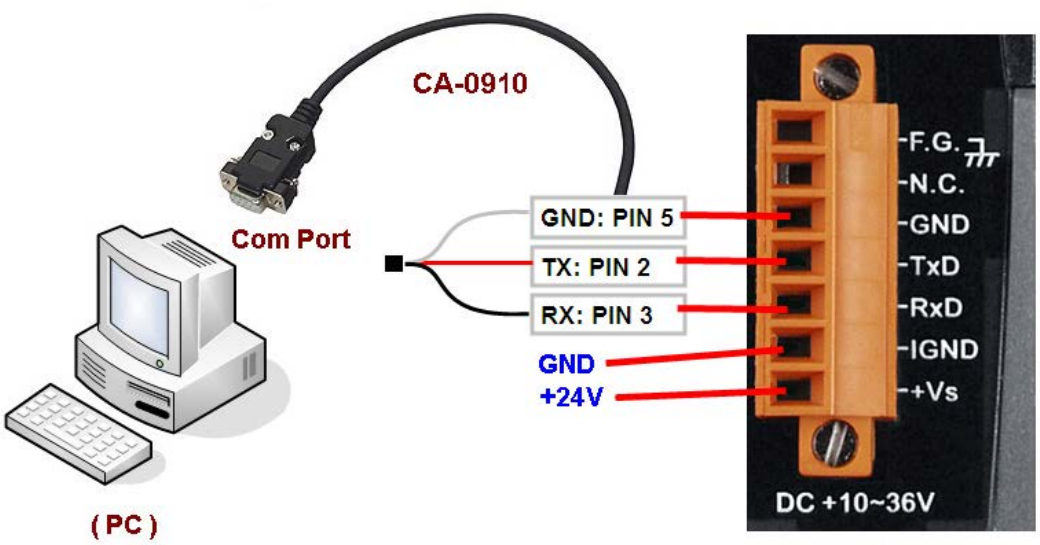
Start the *EzMove Utility* and open the "Update Firmware" dialog from the menu-bar: [Setting] – [RS-M8194H Setting] – [By COM Port] – [Update Firmware].



Please follow the following procedures to update firmware:

1. **Shut down the RS-M8194H.**

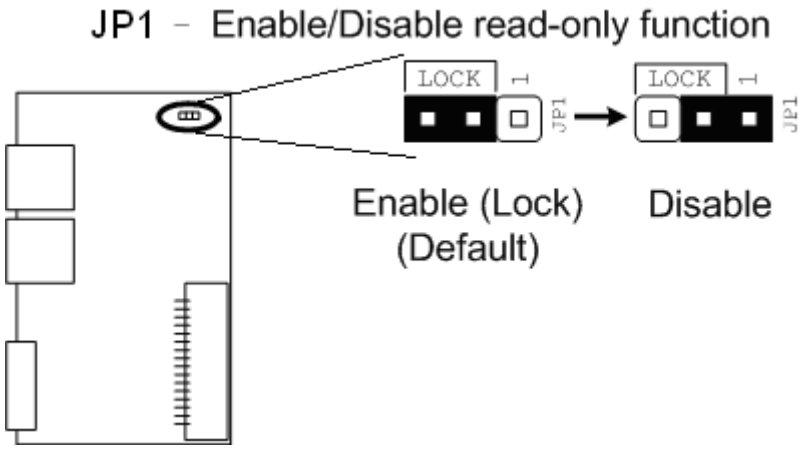
2. **Connect to RS-M8194H with the associated RS-232 cable (CA-0910). The Tx, Rx and GND pins of CA-0910 are connected to the Rx, Tx and GND ports of RS-M8194H. Please connect the other end (9-pin, D-sub connector) to the normal COM port of desktop/laptop.**



3. Set the DIP-switch to “Init”.

(Dip Switch -- Init)

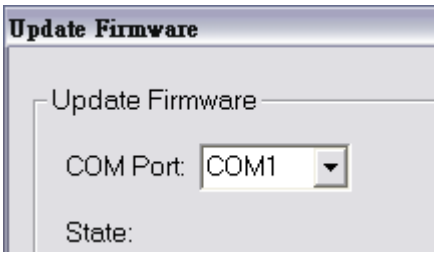
4. Change the setting of JP1 to *disable* ead-only function.



5. Power up the RS-M8194H.

6. **Delete the autoexec.bat and RM94H_XX.EXE in the installed folder of EzMove (\ICPDAS\RS_M8194H\EzMove_Utility). Copy the new autoexec.bat, RS_M8194H_API.dll and the RM94H_XX.EXE into the same folder.**

7. **Choose the COM Port that is connected to RS-M8194H (through CA-0903 cable).**



8. **Click Dowload button to start firmware-updating procedure.**

9. **After finishing firmware-updating procedure, shut down the RS-M8194H.**

10. **Restore the setting of JP1 to *enable* ead-only function.**

11. **Set the DIP-switch to "Run".**

(Dip Switch -- Run)

12. **Power up the RS-M8194H.**