MITSUBISHI ELECTRIC

Mitsubishi Electric Industrial Robot

CR800 series controller
CR750/CR751 series controller

Ethernet Function Instruction Manual

MELFA
BFP-A3379-A

■ Revision History

Print Date	Instruction Manual No.	Revision Content
2015-03-18	BFP-A3379	First print
2017-04-28	BFP-A3379-A	Descriptions about the CR800 controller have been added.

⚠ Safety Precautions

Always read the following precautions and the separate "Safety Manual" before starting use of the robot to learn the required measures to be taken.

⚠CAUTION All teaching work must be carried out by an operator who has received special training.
(This also applies to maintenance work with the power source turned ON.)
→Enforcement of safety training

⚠CAUTION For teaching work, prepare a work plan related to the methods and procedures of operating the robot, and to the measures to be taken when an error occurs or when restarting. Carry out work following this plan.
(This also applies to maintenance work with the power source turned ON.)
→Preparation of work plan

⚠WARNING Prepare a device that allows operation to be stopped immediately during teaching work.
(This also applies to maintenance work with the power source turned ON.)
→Setting of emergency stop switch

⚠CAUTION During teaching work, place a sign indicating that teaching work is in progress on the start switch, etc.
(This also applies to maintenance work with the power source turned ON.)
→Indication of teaching work in progress

⚠DANGER Provide a fence or enclosure during operation to prevent contact of the operator and robot.
→Installation of safety fence

⚠CAUTION Establish a set signaling method to the related operators for starting work, and follow this method.
→Signaling of operation start

⚠CAUTION As a principle turn the power OFF during maintenance work. Place a sign indicating that maintenance work is in progress on the start switch, etc.
→Indication of maintenance work in progress

⚠CAUTION Before starting work, inspect the robot, emergency stop switch and other related devices, etc., and confirm that there are no errors.
→Inspection before starting work

The points of the precautions given in the separate "Safety Manual" are given below.
Refer to the actual "Safety Manual" for details.

DANGER When automatic operation of the robot is performed using multiple control devices (GOT, programmable controller, push-button switch), the interlocking of operation rights of the devices, etc. must be designed by the customer.

CAUTION Use the robot within the environment given in the specifications. Failure to do so could lead to faults or a drop of reliability.
(Temperature, humidity, atmosphere, noise environment, etc.)

CAUTION Transport the robot with the designated transportation posture. Transporting the robot in a non-designated posture could lead to personal injuries or faults from dropping.

CAUTION Always use the robot installed on a secure table. Use in an instable posture could lead to positional deviation and vibration.

CAUTION Wire the cable as far away from noise sources as possible. If placed near a noise source, positional deviation or malfunction could occur.

CAUTION Do not apply excessive force on the connector or excessively bend the cable. Failure to observe this could lead to contact defects or wire breakage.

CAUTION Make sure that the workpiece weight, including the hand, does not exceed the rated load or tolerable torque. Exceeding these values could lead to alarms or faults.

WARNING Securely install the hand and tool, and securely grasp the workpiece. Failure to observe this could lead to personal injuries or damage if the object comes off or flies off during operation.

WARNING Securely ground the robot and controller. Failure to observe this could lead to malfunctioning by noise or to electric shock accidents.

CAUTION Indicate the operation state during robot operation. Failure to indicate the state could lead to operators approaching the robot or to incorrect operation.

WARNING When carrying out teaching work in the robot's movement range, always secure the priority right for the robot control. Failure to observe this could lead to personal injuries or damage if the robot is started with external commands.

CAUTION Keep the jog speed as low as possible, and always watch the robot. Failure to do so could lead to interference with the workpiece or peripheral devices.

CAUTION After editing the program, always confirm the operation with step operation before starting automatic operation. Failure to do so could lead to interference with peripheral devices because of programming mistakes, etc.

CAUTION Make sure that if the safety fence entrance door is opened during automatic operation, the door is locked or that the robot will automatically stop. Failure to do so could lead to personal injuries.

CAUTION Never carry out modifications based on personal judgments, non-designated maintenance parts. Failure to observe this could lead to faults or failures.

WARNING When the robot arm has to be moved by hand from an external area, do not place hands or fingers in the openings. Failure to observe this could lead to hands or fingers catching depending on the posture.

CAUTION Do not stop the robot or apply emergency stop by turning the robot controller's main power OFF. If the robot controller main power is turned OFF during automatic operation, the robot accuracy could be adversely affected. Also a dropped or coasted robot arm could collide with peripheral devices.

CAUTION Do not turn OFF the robot controller's main power while rewriting the robot controller's internal information, such as a program and parameter. Turning OFF the robot controller's main power during automatic operation or program/parameter writing could break the internal information of the robot controller.

DANGER Do not connect the Handy GOT when using the GOT direct connection function of this product. Failure to observe this may result in property damage or bodily injury because the Handy GOT can automatically operate the robot regardless of whether the operation rights are enabled or not.

DANGER Do not connect the Handy GOT to a programmable controller when using an iQ Platform compatible product with the CR750-Q/CR751-Q/CR800-R controller. Failure to observe this may result in property damage or bodily injury because the Handy GOT can automatically operate the robot regardless of whether the operation rights are enabled or not.

DANGER Do not remove the SSCNET III cable while power is supplied to the multiple CPU system or the servo amplifier when using an iQ Platform compatible product with the CR750-Q/CR751-Q/CR800-R controller.. Do not look directly at light emitted from the tip of SSCNET III connectors or SSCNET III cables of the Motion CPU or the servo amplifier. Eye discomfort may be felt if exposed to the light.
(Reference: SSCNET III employs a Class 1 or equivalent light source as specified in JIS C 6802 and IEC60825-1 (domestic standards in Japan).)

DANGER Do not remove the SSCNET III cable while power is supplied to the controller.
Do not look directly at light emitted from the tip of SSCNET III connectors or SSCNET III cables. Eye discomfort may be felt if exposed to the light.
(Reference: SSCNET III employs a Class 1 or equivalent light source as specified in JIS C 6802 and IEC60825-1 (domestic standards in Japan).)

DANGER Attach the cap to the SSCNET III connector after disconnecting the SSCNET III cable. If the cap is not attached, dirt or dust may adhere to the connector pins, resulting in deterioration connector properties, and leading to malfunction.

CAUTION Make sure there are no mistakes in the wiring. Connecting differently to the way specified in the manual can result in errors, such as the emergency stop not being released. In order to prevent errors occurring, please be sure to check that all functions (such as the teaching box emergency stop, customer emergency stop, and door switch) are working properly after the wiring setup is completed.

CAUTION Use the network equipments (personal computer, USB hub, LAN hub, etc.) confirmed by manufacturer. The thing unsuitable for the FA environment (related with conformity, temperature or noise) exists in the equipments connected to USB. When using network equipment, measures against the noise, such as measures against EMI and the addition of the ferrite core, may be necessary. Please fully confirm the operation by customer. Guarantee and maintenance of the equipment on the market (usual office automation equipment) cannot be performed.

Contents

1. Before use

This chapter describes the confirmation items and cautionary items which must be read before practical use of the Ethernet interface.

1.1. How to use the instruction manual

1.1.1. Content of instruction manual

Through the following configuration, this document introduces the Ethernet interface function. As for the functions available in the standard robot controller and the operation method, please refer to the "Instruction Manual" provided with the robot controller.

Table 1.1 Content of the instruction manual

Chapter	Title	Content
1	Before use	In addition to the using method of the instruction manual, the confirmation items and cautionary items are introduced to use the Ethernet interface.
2	Preparation before use	The preparatory work is introduced to use the Ethernet interface. Referring to the chapter, install the interface card, apply the cabling and wiring and confirm the other setting items.
3	Trial for operation	Using the system configured in "2. Preparation before use" in this manual, it introduces a series of the operating methods from the start-up to the stop. Referring to each introduction, understand the basic operating method.
4	Explanation of functions	The method to operate the Ethernet interface is introduced to each operation function. The details of each operation method are introduced in this chapter.
5	Real-time monitor functional	The details of the real-time function is explained in this chapter.
6	SLMP Connection	The communication with external devices by SLMP communication server function is explained in this chapter.
7	Appendix	Since the added errors when indexing the terms or using the Ethernet interface are herein described, refer to this chapter as necessary.

1.2. Terms used in the instruction manual

The following terms are used in this document.

(1) Ethernet interface

The Ethernet interface is network functions to the robot controller.

(2) Network personal computer

The personal computer is a commercially available one which provides the network function, integrating the Ethernet interface card. Windows XP / Windows 7 / Windows 8 are applicable as the operating system.

(3) MELFA-BASIC V/VI command

This is a type of robot language.

1.3. Confirmation of product

The standard configuration of the product supplied by the customer is as follows. Confirm the configuration.
In addition to the standard robot system configuration, the following is necessary. These devices are separately procured by the customer.

No.	Part name	Type	Qty.
1)	Network personal computer (Network interface is necessary.)	Personal computer operated by Windows XP / Windows 7 / Windows 8. Computer with TCP/IP network functions, such as Linux OS (Operation is not verified.)	1 or more
2)	Ethernet cable (Select the straight cable or cross cable depending on the connection system.)	Cable	1 or more

Prepare the following as necessary.

3)	Hub (Necessary if it is used in the LAN environment.)	(Goods on the market)	1
4)	Robot controller programming aiding tool corresponding to Windows for Robot controller of our company	(An optional) Personal computer Support Software	1
5)	Application for network communication program production corresponding to Windows	(Goods on the market) Microsoft. Visual Studio etc.	1

1.4. Ethernet interface

1.4.1. Function of Ethernet interface

The Ethernet interface has the following functions.

(1) The connections with 100BASE-TX (for CR750/CR751) and with 1000BASE-T (for CR800) are supported.

(2) TCP/IP protocol is used to allow the communication with the personal computer on the Ethernet.

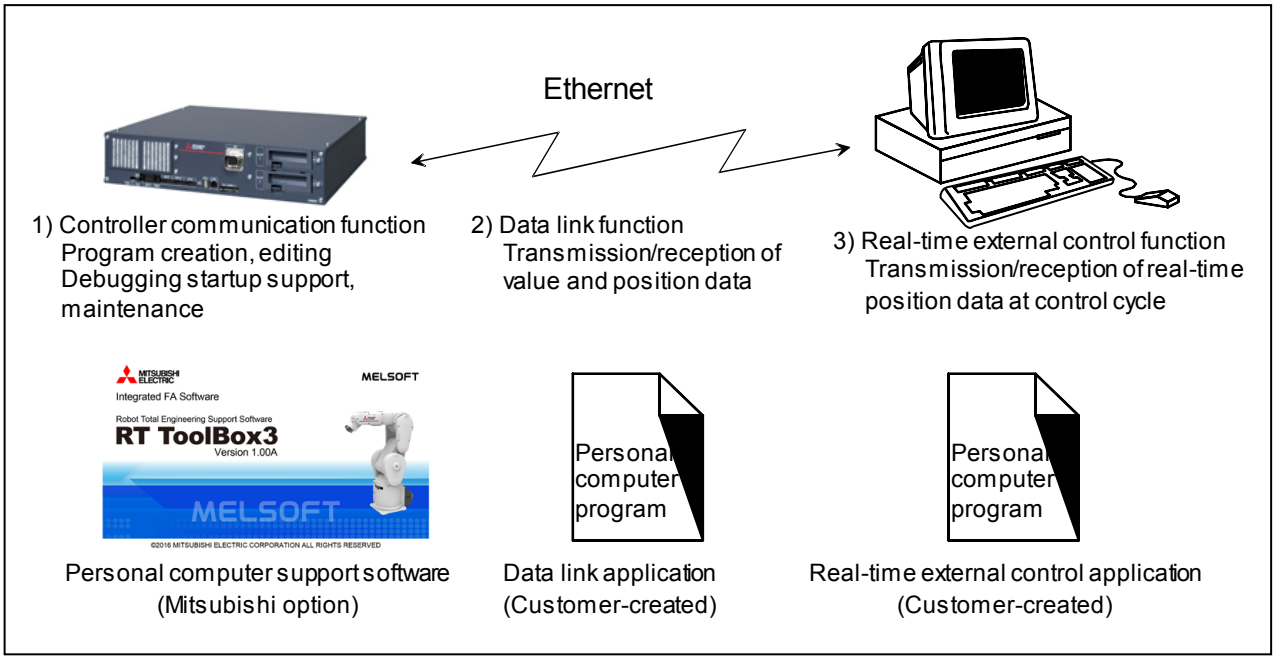
(3) The sampling program (corresponding to Microsoft Visual Basic Version 5.0) of the personal computer is equipped.
The following is provided as the samples. (Refer to Chapter 6 Appendix.)
• The data link function is used to transmit and receive the variables of personal computer and robot (characters and numerical values). (OPEN/INPUT#/PRINT#)
Here, approve that the result of the operation of the application which the customer produces on the basis of the sample is out of the responsibility with our company.

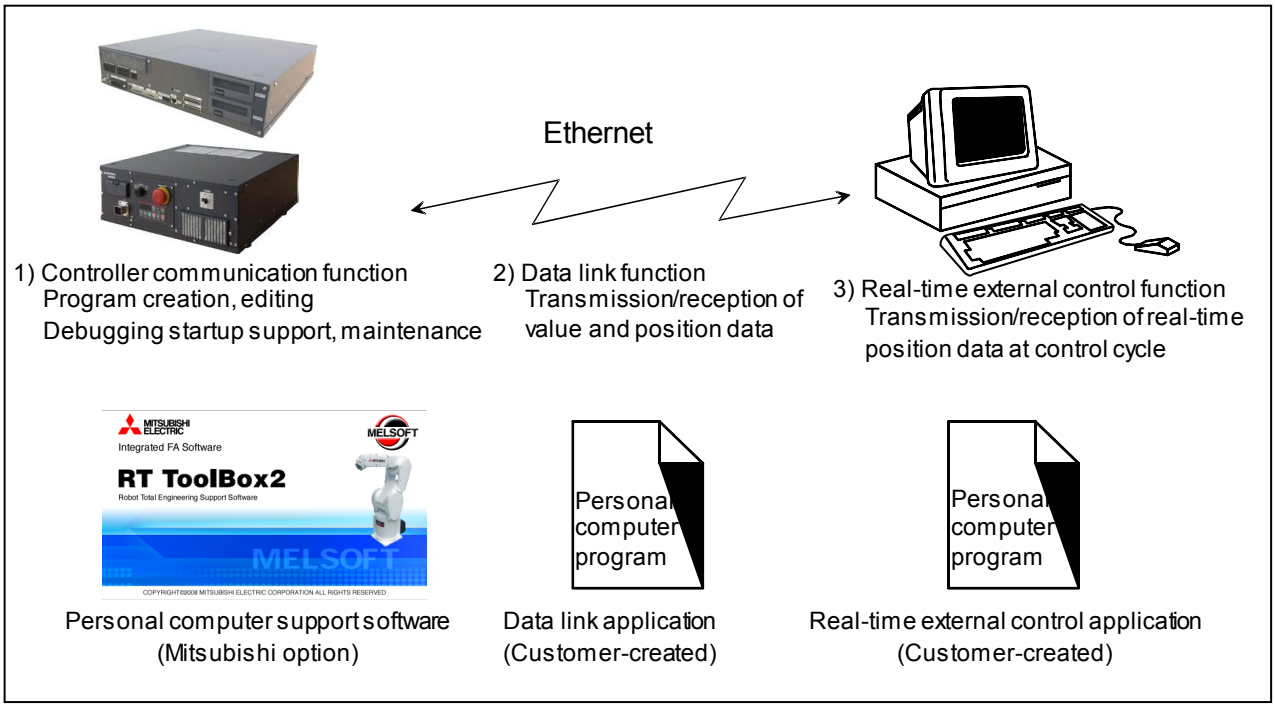
(4) The three Ethernet functions are described below.

Refer to the section "4. Explanation of each function" for details on each function.

No.	Outline of function	Remarks	Reference page
1)	Controller communication function Data can be communicated with the robot controller via Ethernet. (Program upload/download, status monitor, etc.) Personal computer support software (optional) is available as an application.	* Communication with up to 16 clients is possible.	Chapter 1 General Chapter 2 General Chapter 3.1 Chapter 6.1
2)	Data link function The value and position data can be linked between the robot program and personal computer using MELFA-BASIC V/VI language (OPEN/PRINT/INPUT command).	* By changing the communication open destination COM No., communication with applications in up to 8 clients is possible.	Chapter 1 General Chapter 2 General Chapter 3.2 Chapter 4.1 Chapter 6.1 Chapter 6.2.1
3)	Real-time external control function The position command data can be retrieved and operated at the robot motion control cycle unit. Joint, XYZ or motor pulse can be designated for the position data. It is also possible to monitor the input/output signals or output the signals simultaneously. Control is started with the MXT command (MELFA-BASIC V/VI language).	* The user must create an application program on the personal computer side to control the robot. Communication is carried out one-on-one.	Chapter 1 General Chapter 2 General Chapter 3.3 Chapter 4.2 Chapter 6.1 Chapter 6.2.2

* The personal computer used to communicate with the robot controller must be located on the same network. Communication cannot be carried out over firewalls (from internet) or over gateways (from different adjacent network, etc.). Consider operation with a method that communicates via a server (i.e., HTTP server, etc.) connected to the same network as the robot controller. Pay special attention to safety and response in this case.





2. Preparation before use

What is done before use is described.

Connection of Ethernet cable … Refer to 2.1.

↓

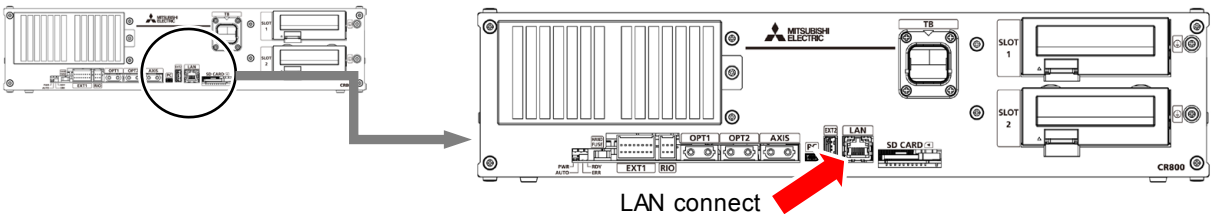
Parameter setting … Refer to 2.2.

2.1. Connection of Ethernet cable

As shown below, connect the Ethernet cable to the connector.
When the hub is used, use the straight cable. Or when the personal computer and controller are connected to each other one to one, use the cross cable.

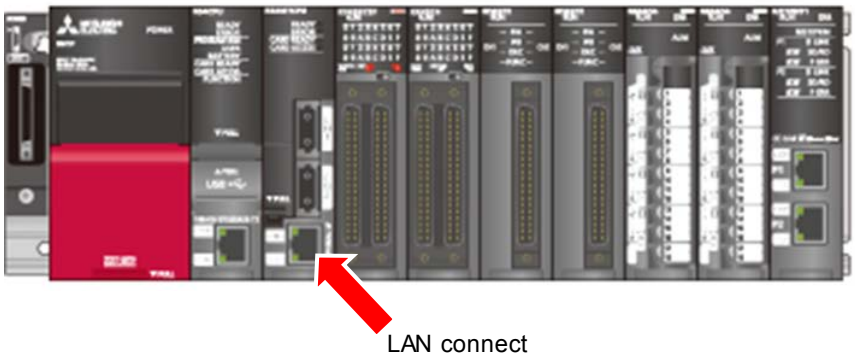
<CR800-D controller>

CR800-D controller front



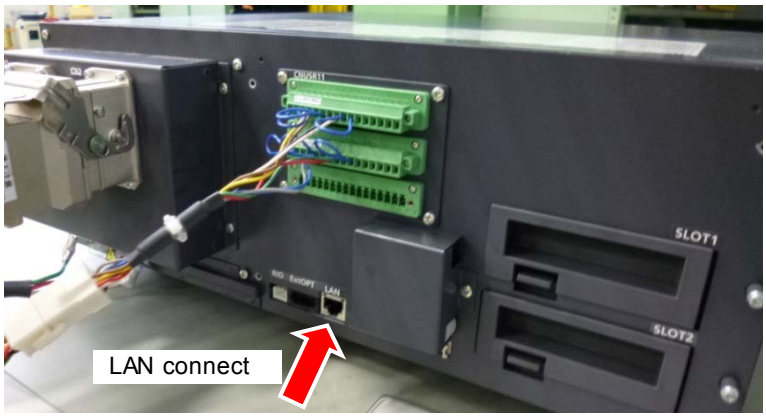
<CR800-R controller>

Robot CPU unit front



<CR750 controller>

CR750 controller back

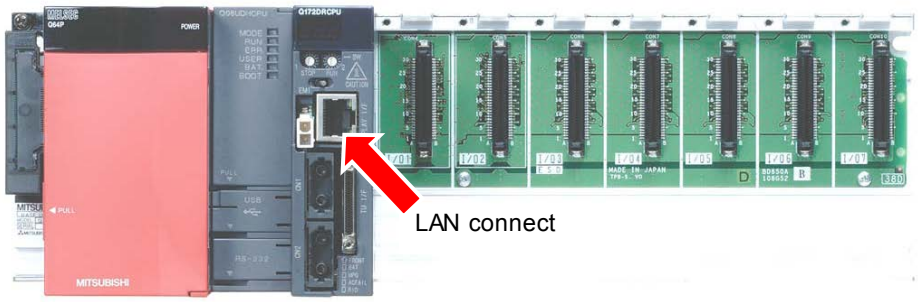


<CR751 controller>

CR751 controller front

<CR750-Q/CR751-Q controller>

Robot CPU unit front



2.2. Parameter setting

Before use, it is necessary to set the following parameters. The parameters which are set on the robot controller are shown in the following list. For the method to set the parameter, refer to the instruction manual of the controller.

⚠CAUTION After changing the parameters, turn the power supply of the controller from OFF to ON. Unless this is done, the changed parameters will not become valid.

2.2.1. Parameter list

The parameters are listed below. For details of the parameters, refer to "2.2.2. Details of parameters".

O ... Setting is necessary
- ... Setting is unnecessary

Parameter list

Parameter name	Details	Number of elements	Default value	Controller communication function	Data link function	Real-time control function
NETIP	IP address of robot controller	Character string 1	"192.168.0.20"	O	O	O
NETMSK	Sub-net-mask	Character string 1	"255.255.255.0"	O	O	O
NETPORT	Port No. Range 0 to 32767 For function expansion (reserved), ---------- Correspond to OPT 11-19 of COMDEV (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Numerical value 10	10000, 10001, 10002, 10003, 10004, 10005, 10006, 10007, 10008, 10009	O	O	O
CPRCE11 CPRCE12 CPRCE13 CPRCE14 CPRCE15 CPRCE16 CPRCE17 CPRCE18 CPRCE19	Protocol 0: No-procedure 1: Procedure, 2: Data link (1: Procedure has currently no function.) Correspond to OPT 11-19 of COMDEV (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Numerical value 9	0 0 0 0 0 0 0 0 0	-	O	-

Parameter name	Details	Number of elements	Default value	Controller communication function	Data link function	Real-time control function
COMDEV	Definition of device corresponding to COM1: to 8 Definition of device corresponding to COM1:, Definition of device corresponding to COM2:, Definition of device corresponding to COM3:, Definition of device corresponding to COM4:, Definition of device corresponding to COM5:, Definition of device corresponding to COM6:, Definition of device corresponding to COM7:, Definition of device corresponding to COM8: . When the data link is applied, setting is necessary. OPT11 to OPT19 are allocated.	Character string 8	, , , , , , ,	-	O	-
NETMODE	Server designation (1: Server, 0: Client) (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Numerical value 9	1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1	-	O	-
NETHSTIP	The IP address of the data communication destination server. * It is valid if specified as the client by NETMODE only. (OPT11) (OPT12) (OPT13) (OPT14) (OPT15) (OPT16) (OPT17) (OPT18) (OPT19)	Character string 9 .	192.168.0.2 , 192.168.0.3 , 192.168.0.4 , 192.168.0.5 , 192.168.0.6 , 192.168.0.7 , 192.168.0.8 , 192.168.0.9 , 192.168.0.10	-	O	-
MXTTOUT	Timeout time for executing real-time external control command (Multiple of 7.1msec, Set -1 to disable timeout)	Value 1 (0-32767)	-1	-	-	O
NETGW	Gateway address	Character string 1	192.168.0.254	O	O	O

2.2.2. Details of parameters

The parameters are herein described in details.

(1) NETIP (IP address of robot controller)

The IP address of the robot controller is set. IP address is like the address of the mail.

The format of IP address is composed of 4 numbers of 0 to 255 and the dot (.) between the numbers.

For example, it is set as 192.168.0.1 or 10.97.11.31.

If the controller and network personal computer are directly connected to each other one-to-one, it is allowed to set default value (a random value) but if it is connected to the local area network (LAN), IP address must be set as instructed by the manager of customer's LAN system.

If any IP addresses are overlapped, the function will not properly operate. Therefore, take care to prevent it from being overlapped with another during setting.

The personal computer used for communication with the robot controller.

(2) NETMSK (sub-net-mask)

Set the sub-net-mask of the robot controller. Among the IP addresses, the sub-net-mask is set to define the sub-net-work.
The format of the sub-net-mask is composed of 4 numbers of 0 to 255 and the dot (.) between the numbers.
For example, it is set as 255.255.255.0 or 255.255.0.0.
As usual, it is allowed to set default value. If it is connected to the local area network (LAN), the sub-net-mask must be set as instructed by the manager of customer's LAN system.

(3) NETPORT (port No.)

The port No. of the robot controller is set. The port No. is like the name of the mail.
For the nine elements, the port numbers are each expressed with a value.
The first element (element No. 1) is used for real-time control.
The second to ninth elements (elements No. 2 to 9) are used for the support software or data link.
Normally, the default value does not need to be changed. Make sure that the port numbers are not duplicated.

(4) CRRCE11 to 19 (protocol)

When using the data link function, the setup is necessary.
Sets the protocol (procedure) for communication. The protocol has three kinds of no-procedure, procedure and data link.
0... No-procedure: The protocol is applied to use the personal computer Support Software .
1... Procedure: Reserved. (Since it is not any function, don't set it by mistake.)
2... Data link: The protocol is used to use OPEN/INPUT/PRINT commands for communication.

(5) COMDEV (Definition of devices corresponding to COM1: to 8)

When using the data link function, the setup is necessary.
Definition of device corresponding to COM1: to 8 is set. COM1: to 8 is used for OPEN command of the robot program.
Be sure to set it only when the data link is specified on setting of the protocol (CPRCE11 to 19).
The setting values of the Ethernet interface option correspond to the port Nos. which are set at the parameter NETPORT.
* In the following parameters NETOPORT (n) and COMDEV(n), n indicates the element No. of that parameter.

n	The device name set up by COMDEV(n)	Port number
1	OPT11	The port number specified by NETPORT(2)
2	OPT12	The port number specified by NETPORT(3)
3	OPT13	The port number specified by NETPORT(4)
4	OPT14	The port number specified by NETPORT(5)
5	OPT15	The port number specified by NETPORT(6)
6	OPT16	The port number specified by NETPORT(7)
7	OPT17	The port number specified by NETPORT(8)
8	OPT18	The port number specified by NETPORT(9)
9	OPT19	The port number specified by NETPORT(10)

For example, if the port No. specified at NETPORT(3) is allocated to the data link of COM:3, the following will be applied.
COMDEV(3) = OPT13 * OPT13 is set at 3rd element of COMDEV.
CPRCE13 = 2 * Set up as a data link.

(6) NETMODE (server specification)
Set up, when using the data link function.
Set the TCP/IP communication in the data link function of the robot controller as the server or the client.
It is necessary to change with the application of the equipment connected to the robot controller.

(7) NETHSTIP (The IP address of the server of the data communication point)
Set up, when using the robot controller as a client by the data link function.
Specify the IP address of the partner server which the robot controller connects by the data link function.
Set up, when only set the robot controller to the client by server specification of NETMODE.

(8) MXTTOUT (Timeout setting for executing real-time external control command)
This is changed when using real-time external control command and setting the timeout time for communication with the robot controller.
Set a multiple of the control cycle (refer to the following).

Controller	Control cycle
CR750/CR751 series	Approx. 7.11 msec
CR800 series	Approx. 3.5 msec (*If user mechanical is set, approx. 7.11 msec)

When the real-time external control command is executed, the timeout time during which no communication data is received by the robot controller from the personal computer is counted up. If the count reaches the value set in MXTTOUT, the operation will stop with the error (#7820). For example, to generate an error when there is no communication for approx. 7 seconds, set 1000.
This setting is set to -1 (timeout disabled) as the default.

2.2.3. Parameter setting example 1 (When the Support Software is used)

The setting example to use the Support Software is shown below.

Set the parameters for the robot controller, and the network for the personal computer OS being used.

Conditions for example 1

IP address of robot controller	192.168.0.20
IP address of personal computer	192.168.0.10
Port No. of robot controller	10001

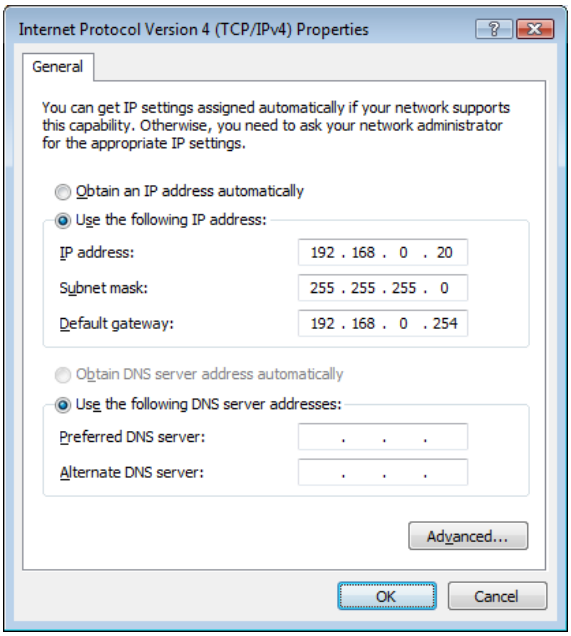
Set the robot controller parameters as shown below.

If the default settings are to be used, the parameters do not need to be changed.

Parameter setting for example 1

Parameter	Before/after change	Parameter value
NETIP	Before	192.168.0.20
	After	192.168.0.20 (unchanged)
NETPORT	Before	10001
	After	10001 (unchanged)

Next, set the personal computer IP address to 192.168.0.10. Set this value on the Network Properties screen.



The personal computer IP address is set with the Windows TCP/IP Property Network setting (property in network computer). Because the set-up screen differs with versions of Windows, refer to the manuals enclosed with Windows, etc., for details on setting this address.

Refer to the instruction manuals enclosed with the personal computer support software for details on setting and using the personal computer support software.

2.2.4. Parameter setting example 2-1

(When the data link function is used: When the controller is the server)

Shows the example of the setting, when the controller is server by the data link function.

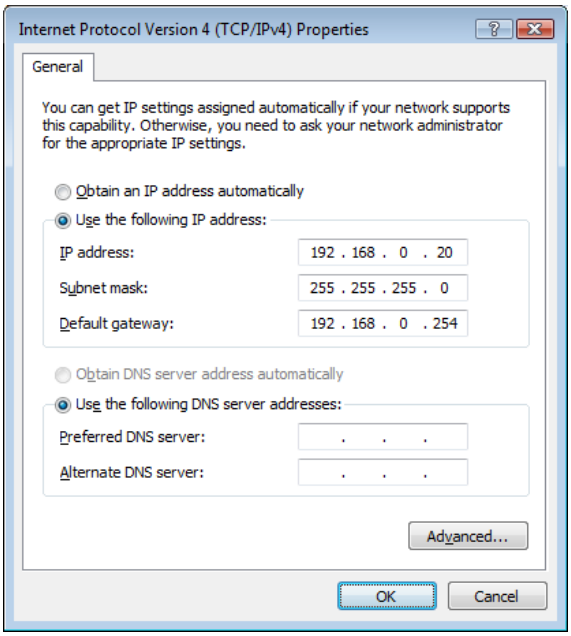
Conditions for example 2-1

Robot controller IP address	192.168.0.20
Personal computer IP address	192.168.0.10
Robot controller port No.	10003
Communication line No. <For MELFA-BASIC V/VI> OPEN command COM No.	 COM3:

Parameter setting for example 2-1

Parameter	Before/after change	Parameter value
NETIP	Before	192.168.0.20
	after	192.168.0.20 (unchanged)
NETPORT	Before	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009
	after	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009 (unchanged)
CPRCE13	Before	0
	after	2
COMDEV	Before	, , , , , , ,
	after	, , OPT13, , , , ,

Next, set the personal computer IP address to 192.168.0.10. Set this value on the Network Properties screen.



The personal computer IP address is set with the Windows TCP/IP Property Network setting (property in network computer). Because the set-up screen differs with versions of Windows, refer to the manuals enclosed with Windows, etc., for details on setting this address.

Refer to the instruction manuals enclosed with the personal computer support software for details on setting and using the personal computer support software.

2.2.5. Parameter setting example 2-2

(When the data link function is used: When the controller is the client)

Shows the example of the setting, when the controller is client by the data link function.

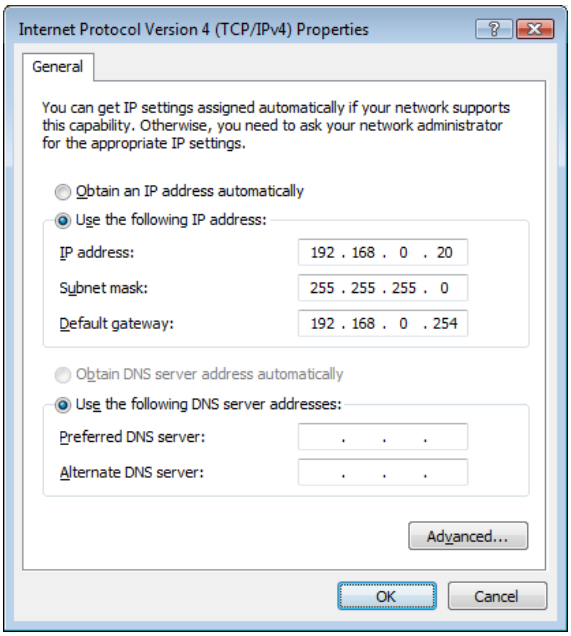
Conditions for example 2-2

Robot controller IP address	192.168.0.20
Personal computer IP address	192.168.0.10
Robot controller port No.	10003
Communication line No. <For MELFA-BASIC V/VI> OPEN command COM No.	 COM3:

Parameter setting for example 2-2

Parameter	Before/after change	Parameter value
NETIP	Before	192.168.0.20
	After	192.168.0.20 (unchanged)
NETPORT	Before	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009
	After	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009 (unchanged)
CPRCE13	Before	0
	After	2
COMDEV	Before	, , , , , , ,
	After	, , OPT13, , , , ,
NETMODE	Before	1,1,1,1,1,1,1,1,1,1
	After	1,1,0,1,1,1,1,1,1,1
NETHSTIP	Before	192.168.0.2, 192.168.0.3, 192.168.0.4, 192.168.0.5, 192.168.0.6, 192.168.0.7, 192.168.0.8, 192.168.0.9, 192.168.0.10
	After	192.168.0.2, 192.168.0.3, 192.168.0.2, 192.168.0.5, 192.168.0.6, 192.168.0.7, 192.168.0.8, 192.168.0.9, 192.168.0.10

Next, set the personal computer IP address to 192.168.0.10. Set this value on the Network Properties screen.



The personal computer IP address is set with the Windows TCP/IP Property Network setting (property in network computer). Because the set-up screen differs with versions of Windows, refer to the manuals enclosed with Windows, etc., for details on setting this address.

Refer to the instruction manuals enclosed with the personal computer support software for details on setting and using the personal computer support software.

2.2.6. Parameter setting example 3 (for using the real-time external control function)

An example of the settings for using the real-time external control function is shown below.

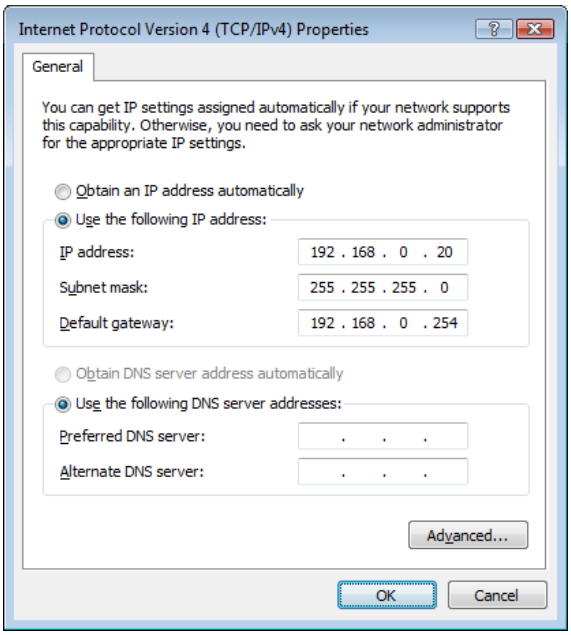
Conditions for example 3

Robot controller IP address	192.168.0.20
Personal computer IP address	192.168.0.10
Robot controller port No.	10000

Parameter setting for example 3

Parameter	Before/after change	Parameter value
NETIP	Before	192.168.0.20
	after	192.168.0.20 (unchanged)
NETPORT	Before	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009
	after	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009 (unchanged)
MXTTOUT	Before	-1
	after	-1 (unchanged)

Next, set the personal computer IP address to 192.168.0.10. Set this value on the Network Properties screen.



The personal computer IP address is set with the Windows TCP/IP Property Network setting (property in network computer). Refer to the manuals enclosed with Windows, etc., for details on setting this address.

Refer to the instruction manuals enclosed with the personal computer support software for details on setting and using the personal computer support software.

2.3. Connection confirmation

Before use, confirm the following items again.

Connection confirmation

No.	Confirmation item	Check
1	Is the teaching pendant securely fixed?	
2	Is the Ethernet cable properly connected between the controller and personal computer? (Refer to 2.1 in this manual.)	
3	Is any proper Ethernet cable used? (This cross cable is used to connect the personal computer and controller one-on-one. When using a hub with LAN, use a straight cable.)	
4	Is the parameter of the controller properly set? (Refer to 2.2 in this manual.)	
5	Is the power supply of the controller turned off once after the parameter set?	

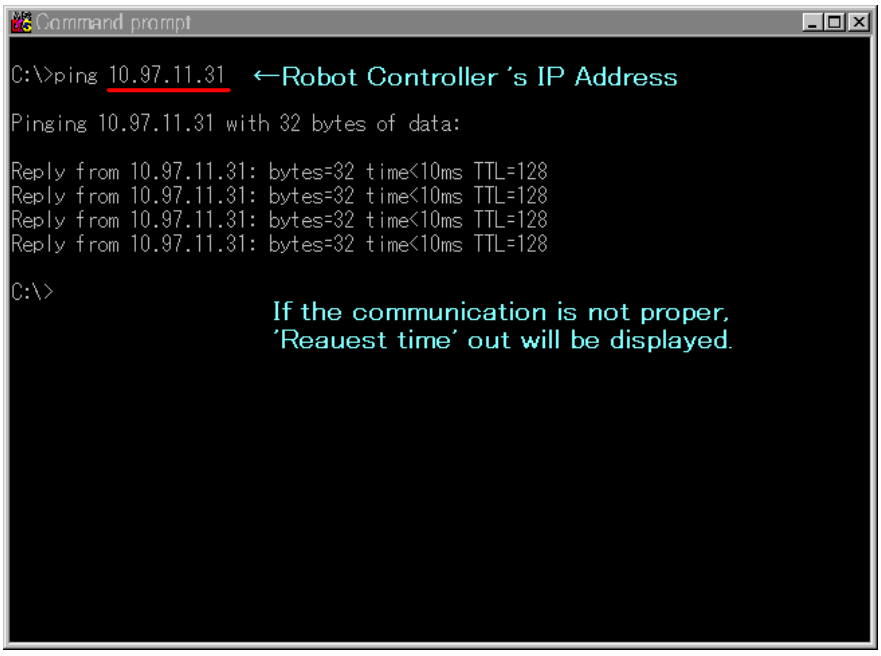
2.3.1. Checking the connection with the Windows ping command

The method for checking the connection with the Windows ping command is shown below.

Start up the " MS-DOS Prompt " from the Windows " Start " - " Programs " menu, and designate the robot controller IP address as shown below.

If the communication is normal, " Reply from ... " will appear as shown below.

If the communication is abnormal, " Request time out " will appear.

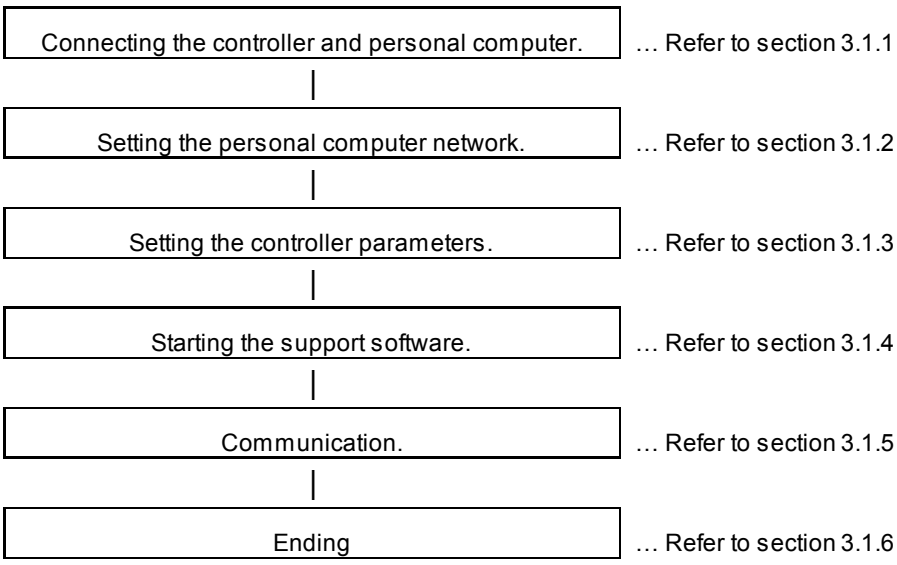


3. Operation

This chapter explains the methods for using the three Ethernet option functions with a system in which the controller and network personal computer are connected with a one-on-one cross cable.

3.1. Controller communication function

The operations for communicating with the personal computer support software are explained in this section.



3.1.1. Connecting the controller and personal computer

Connect the controller and the personal computer with the following Ethernet cable.

Controller	Ethernet cable
CR750/CR751 series	100BASE-TX compatible cable
CR800-R series	100BASE-TX compatible cable
CR800-D series	1000BASE-TX compatible cable

Refer to the connection described in section "2.1 Ethernet cable".

3.1.2. Setting the personal computer network

Refer to section "2.2.3 Example of setting the parameters 1 (for using the support software)" and set the network.

3.1.3. Setting the controller parameters

Turn ON the robot controller power, and set the parameters as shown below.

If the default settings are to be used, the parameters do not need to be changed.

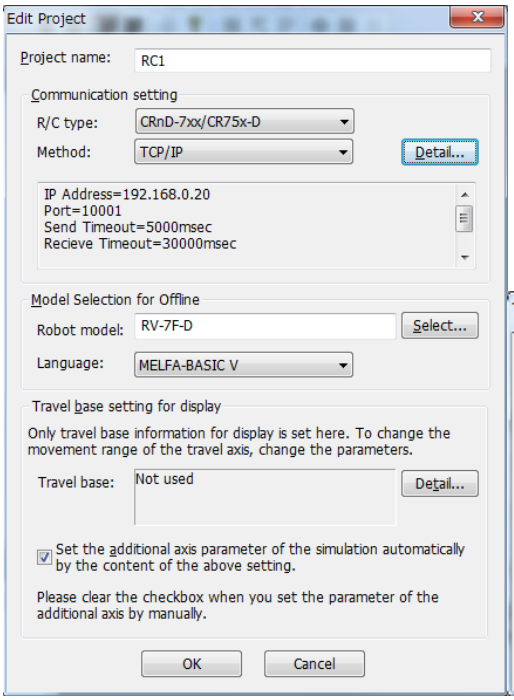
Name of parameter to change	Before/after changes	Parameter value
NETIP	Before	192.168.0.20
	After	192.168.0.20 (Default value)
NETPORT	Before	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009
	After	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009 (Default value)

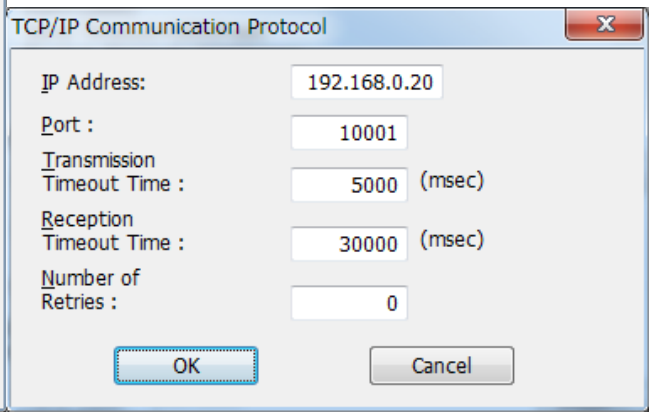
After setting the parameters, turn the robot controller power OFF and ON.

Refer to the instruction manual enclosed with the robot controller for details on setting the parameters.

3.1.4. Setting the personal computer support software communication

Start the personal computer support software and make the communication settings. Set the communication method to TCP/IP, and the IP Address to 192.168.0.20.

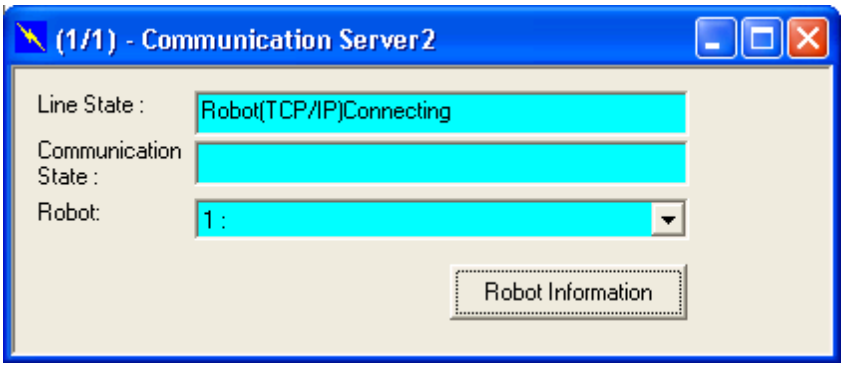




Refer to the instruction manual enclosed with the personal computer support software for details on setting the personal computer support software.

3.1.5. Communication

Communicate with the personal computer support software.



Communication can be carried out with the Ethernet interface TCP/IP.

Refer to the instruction manual enclosed with the personal computer support software for details on using the personal computer support software.

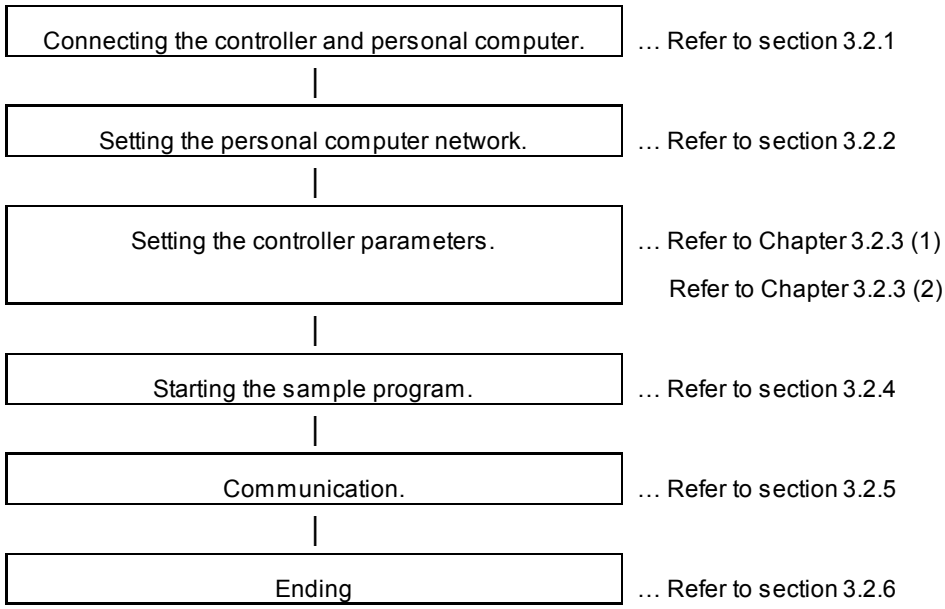
If communication is not possible, refer to section "2.3 Checking the connection" and check the state.

⚠CAUTION

When the robot controller power is turned OFF and ON, the connection will be disconnected and communication will be disabled.
In this case, end the application software on the personal computer once, and then restart.

3.2. Data link function

This section explains the operations for starting the sample program given in "6.2.1 Sample program for data link function" and communicating with a system in which the controller and network personal computer are connected with a one-on-one cross cable.



3.2.1. Connect the controller and personal computer.

Connect the controller and personal computer with a cross cable.
Refer to the connection described in section "2.1 Ethernet cable".

3.2.2. Setting the personal computer network.

Set one of the following clauses as reference corresponding to the customer's system configuration. (The controller is the server or the client)

- 2.2.4 Parameter setting example 2-1 (When the data link function is used: When the controller is the server.)
- 2.2.5 Parameter setting example 2-2 (When the data link function is used: When the controller is the client.)

3.2.3. Setting the controller parameters.

The contents of the setting of parameter differ, when the robot controller is specified as server and client of TCP/IP connection.

Turn ON the robot controller power, and set the parameters as shown below.

The NETIP/NETPORT parameters do not need to be changed when using the default values.

After setting the parameters, turn the robot controller power OFF and ON.

Refer to the instruction manual enclosed with the robot controller for details on setting the parameters.

(1) When the controller is specified as the server

Parameter	Before/after change	Parameter value
NETIP	Before	192.168.0.20
	After	192.168.0.20 (unchanged)
NETPORT	Before	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009
	After	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009 (unchanged)
CPRCE13	Before	0
	After	2
COMDEV	Before	, , , , , , ,
	After	, , OPT13, , , , ,

(2) When the controller is specified as the client

Parameter	Before/after change	Parameter value
NETIP	Before	192.168.0.20
	After	192.168.0.20 (unchanged)
NETPORT	Before	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009
	After	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009 (unchanged)
CPRCE13	Before	0
	After	2
COMDEV	Before	, , , , , , ,
	After	, , OPT13, , , , ,
NETMODE	Before	1,1,1,1,1,1,1,1,1,1
	After	1,1,0,1,1,1,1,1,1,1
NETHSTIP	Before	192.168.0.2, 192.168.0.3, 192.168.0.4, 192.168.0.5, 192.168.0.6, 192.168.0.7, 192.168.0.8, 192.168.0.9, 192.168.0.10
	After	192.168.0.2, 192.168.0.3, 192.168.0.2, 192.168.0.5, 192.168.0.6, 192.168.0.7, 192.168.0.8, 192.168.0.9, 192.168.0.10

3.2.4. Starting the sample program

The test program is an example for establishing a data link between the robot and personal computer. COM3 is used.

(1) Using the teaching pendant or personal computer support software, register the following robot program with an appropriate program name.

<Robot program>

1) Example for MELFA-BASIC V

```
1 OPEN "COM3:" AS #1                ' Open as communication line COM3
2 PRINT #1,"START"                  ' Send START character string
3 *LOOP:INPUT #1,DATA               ' Wait for reception of value in DATA variable
4 IF DATA<0 THEN GOTO *LEND         ' If DATA is negative, jump to line 7 and end
5 PRINT #1,"DATA=";DATA             ' Reply DATA = value
6 GOTO *LOOP                        ' Jump to line 3 and repeat
7 *LEND:PRINT#1,"END"               ' Send END character string
8 END                               ' End
```

(2) Start the personal computer data link program

Refer to section "6.2.1 Sample program for data link function" and create the execution file. (The created execution file will be sample.exe.)

Start Windows Explorer, and double-click on sample.exe.

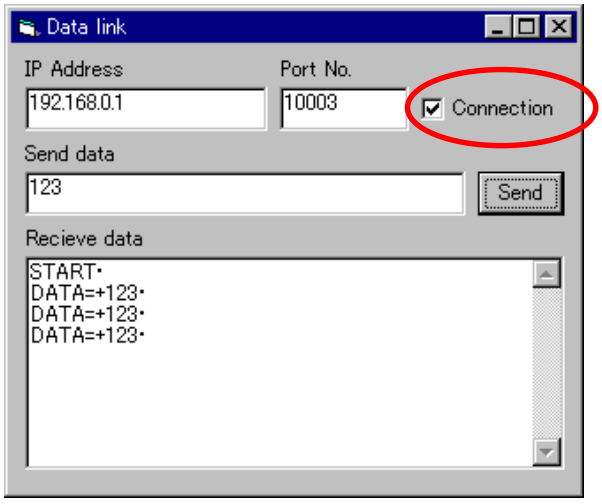
Set the IP address and port No., click on the connection check box, and open the communication line with the controller.

If the Send button is not validated, check that the IP address matches NETIP set with the controller.

If the button is still not validated, refer to section "2.3 Checking the connection", and check the connection cable or restart the controller and sample.exe.

(3) Start the robot program.

Press the START button on the robot controller's operating panel, and start the robot program.



3.2.5. Communication

(1) When the robot controller program is started, first the following data will be sent to the personal computer.

"START"(CR) (CR) indicates the CR code.

(2) When the personal computer receives the data, the characters will appear in the received data area.

START•

(3) Send value data from the personal computer.

For example, input the value data 123 in the transmission data area, and click on the Send button with the mouse.

(4) When the robot controller receives the value data in the DATA variable, it will reply data to the personal computer.

DATA=123 will appear in the personal computer's received data area.

If communication cannot be carried out correctly, refer to section "2.3 Checking the connection" in this manual.

⚠CAUTION

When the robot controller power is turned OFF and ON, the connection will be disconnected and communication will be disabled.
In this case, end the application software on the personal computer once, and then restart.

3.2.6. Ending

(1) Press the END button on the robot controller operating panel, and enter cycle operation.

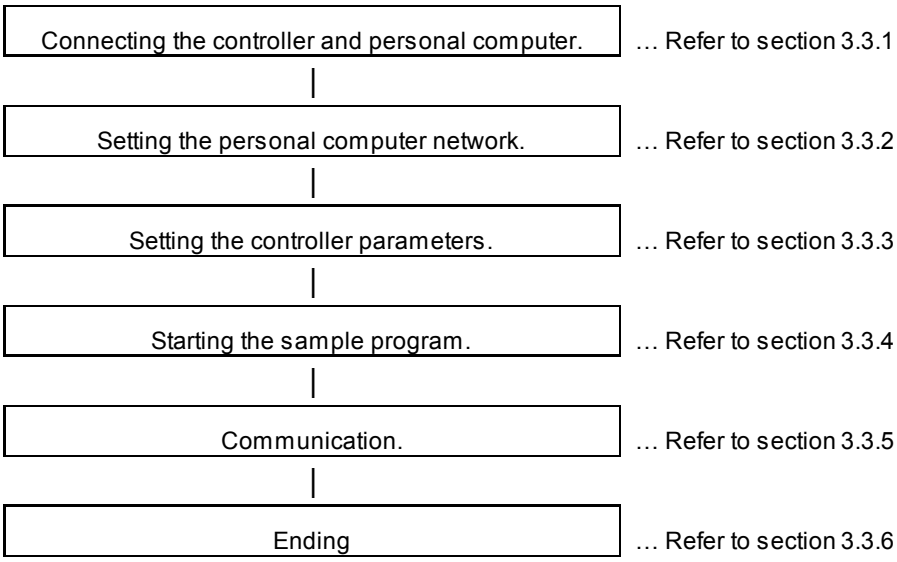
(2) Input the value -1 from the personal computer, and end the program.

(3) End the personal computer's sample program.

(4) Turn OFF the robot controller's power.

3.3. Real-time external control function

This section explains the operations for starting the sample program given in "6.2.2 Sample program for real-time external control function" and communicating with a system in which the controller and network personal computer are connected with a one-on-one cross cable.



3.3.1. Connecting the controller and personal computer

Connect the controller and personal computer with a cross cable.
Refer to the connection described in section "2.1 Ethernet cable".

3.3.2. Setting the personal computer network

Refer to section "2.2.5 Example of setting the parameters 3 (for using the real-time external control function)" and set the network.

3.3.3. Setting the controller parameters

Turn ON the robot controller power, and set the parameters as shown below.
If the default settings are to be used, the parameters do not need to be changed.
After setting the parameters, turn the robot controller power OFF and ON.
Refer to the instruction manual enclosed with the robot controller for details on setting the parameters.

Name of parameter to change	Before/after changes	Parameter value
NETIP	Before	192.168.0.20
	After	192.168.0.20 (Default value)
NETPORT	Before	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009
	After	10000,10001,10002,10003,10004,10005,10006,10007,10008,10009 (Default value)
MXTTOUT	Before	-1
	After	-1 (Default value)

3.3.4. Starting the sample program

The test program is an example of communicating in real-time between the robot and personal computer. The XYZ position data X axis or joint position data J1 axis is commanded from the personal computer to the robot and controlled.

(1) Using the teaching pendant or personal computer support software, register the following robot program with an appropriate program name.

<Robot program>

1) Example for MELFA-BASIC V

```
1 OPEN "ENET:192.168.0. 20" AS #1    ' Designate personal computer side IP address as Ethernet in file No. 1
2 MOV P1                             ' Move to default position P1 (teach random position as P1)
3 MXT1,0                             ' Move according to command value issued from file No. 1
                                       Current XYZ position is replied from controller to personal computer
4 MOV P1                             ' After external control mode ends, move to default position P1 with joint
                                       interpolation
5 HLT                                ' Halt
6 END                                ' End
```

(2) Start the robot program.

Press the START button on the robot controller's operating panel, and start the robot program.

The robot will move to the default position P1, and real-time external control will be executed with the MXT command.

(3) Start the personal computer's real-time external control sample program.

Refer to section "6.2.2 Sample program for real-time external control function" and create the execution file. (The created execution file will be sample.exe.)

Start Windows Explorer, and double-click on sample.exe.

3.3.5. Moving the robot

Specify and input the following values for the numerical value displayed on the screen according to the message of the sample program.

*The IP address (192.168.0.20) of the robot controller of the connection point

*The port number (10001)

*The data type of command

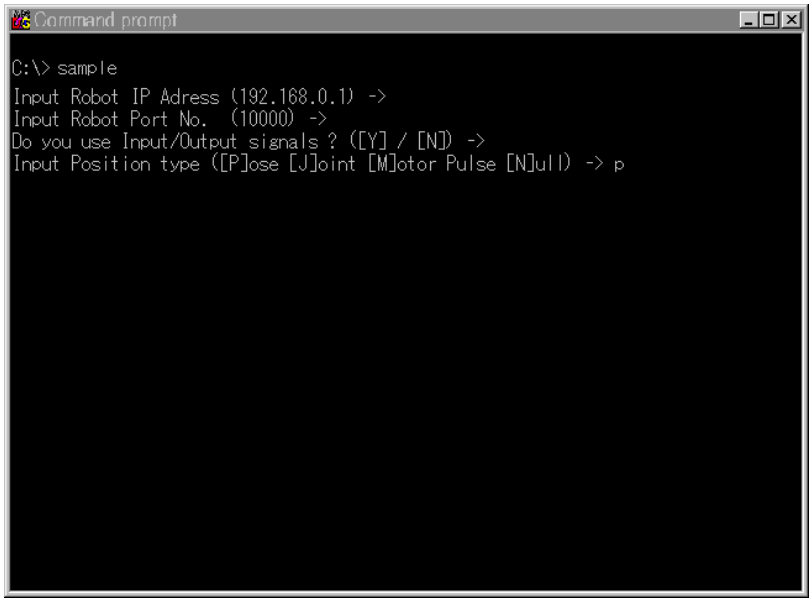
*The data type of monitoring, etc.

Fit the data type of command to the argument of the MXT command of the robot program

Key operation is as follows. For details, refer to the sample program.

Key	Contents
Z or X .	The robot moves.
C	The instruction value is set to 0 and the robot stops.
D	Each time the MOVE key is pressed, change the display/ un-displaying of the monitor data
ENTER	End the MXT command.

If the amount of instructions becomes too large or the movement range of the robot is exceeded, an error is generated and the robot controller stops. In this case, reset the robot controller.



If communication cannot be carried out correctly, refer to section "2.3 Checking the connection", and check the connection cable or restart the controller and sample.exe.

⚠CAUTION

When the robot controller power is turned OFF and ON, the connection will be disconnected and communication will be disabled.
In this case, end the application software on the personal computer once, and then restart.

3.3.6. Ending

(1) Press the END button on the robot controller operating panel, and enter cycle operation.
(2) End the personal computer's sample program.
When the [ENTER] key is pressed, the MXT command will end, the robot will return to the default position, and the robot program will stop.
The sample program will also end.
(3) Turn OFF the robot controller's power.

4. Explanation of functions

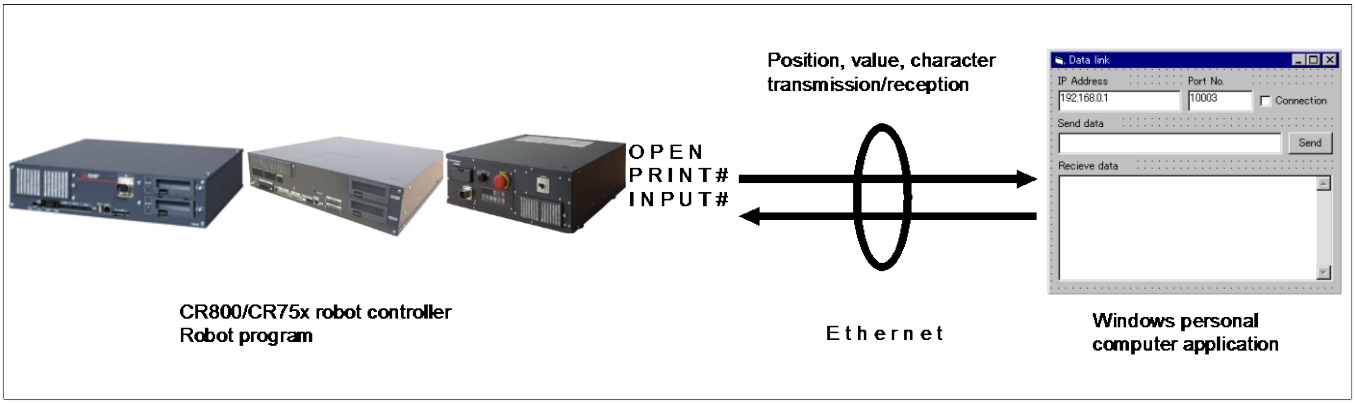
This chapter describes the detailed functions of the Ethernet interface.

4.1. Data link function

OPEN/PRINT/INPUT of the robot language can be used in the Ethernet.
For each robot language, refer to the instruction manual appended to the robot controller.

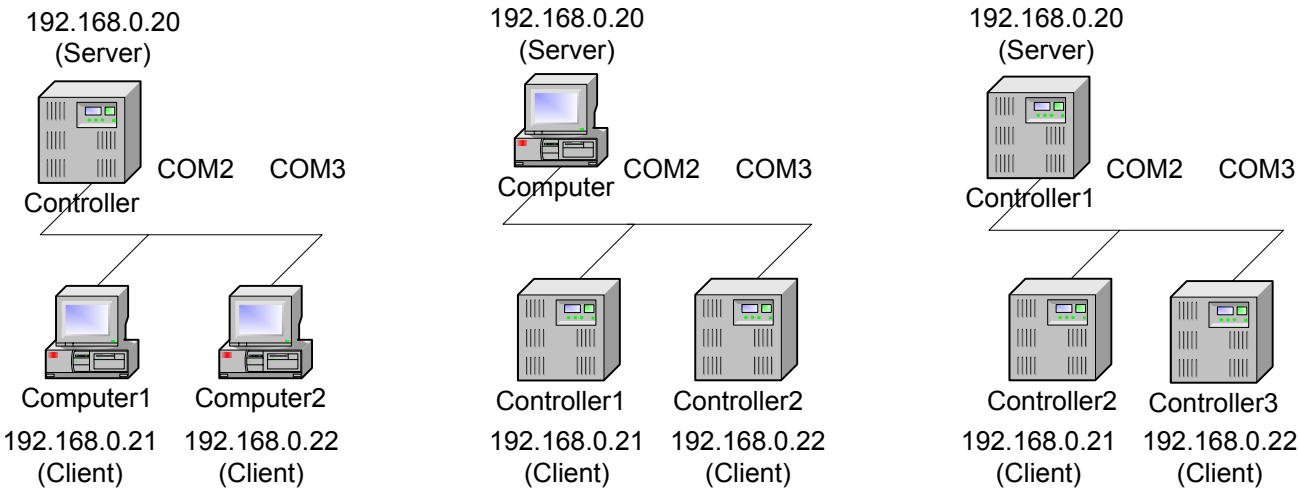
[Statement example] To set port No. 10003 as communication destination and open as #1
Set parameter COMDEV (element No. 3) to OPT13, NETPORT to 10003.

```
1 OPEN "COM3:" AS #1        'Set port No.
2 INPUT #1, C1$             'Read
3 PRINT #1, "Reply", C1$    'Writing
4 CLOSE #1                  'Line closing
5 HLT                       'Stop
```



The data link function of the Ethernet interface has the two kinds shown below.

* Uses the robot controller as the server.
* Uses the robot controller as the client.



Two or more clients are not connectable with the one line number COMn.
Change the line number, when using the robot controller as the server and connecting two or more clients.

4.1.1. MELFA-BASIC V/VI Commands

This section describes the robot language (MELFA-BASIC V/VI).
For more information about OPEN, CLOSE, INPUT# and PRINT# used for data linking, refer to the INSTRUCTION MANUAL Detailed explanations of functions and operations.

M_OPEN

[Function]
Indicates whether or not the file has been opened.

[Format]

<Numeric variable> = M_OPEN [(<file number>)]

[Terminology]

<Numeric variable>	Specify a numeric variable to be assigned.
<File number>	Specify a file number constant between 1 and 8 for the communication line that was opened by the OPEN instruction. If omitted, 1 is set. If 9 or higher is specified, an error occurs when executed.

[Reference Program]

```
1 ' Client Program ----------
2 M1=0
3 M_TIMER(1)=0                                  'Resets the timer to 0.
4 *LOPEN:OPEN "COM2:" AS #1                     'Opens the line.
5 IF M_TIMER(1)>10000.0 THEN *LERROR            'Jumps when 10 seconds elapses.
6 IF M_OPEN(1)<>1 THEN GOTO *LOPEN              'Loops if no connection is made.
7 DEF ACT 1,M_OPEN(1)=0 GOSUB *LHLT2            'Monitors the down state of the server using an interrupt.
8 ACT 1=1                                       'Starts monitoring.
9 *LOOP:M1=M1+1
10 IF M1<10 THEN C1$="MELFA" ELSE C1$="END"     'Sends END after sending the "MELFA" string nine times.
11 PRINT #1,C1$                                 'Sends a character string.
12 INPUT #1,C2$                                 'Receives a character string.
13 IF C1$="END" THEN *LHLT                      'Jumps to CLOSE after sending "END."
14 GOTO *LOOP                                   'Loops.
15 *LHLT:CLOSE #1                               'Closes the line.
16 HLT                                          'Halts the program.
17 END                                          'Ends.
18 *LERROR:ERROR 9100                           'Generates error 9100 if no connection can be made to the
                                                 server.
19 CLOSE #1
20 HLT
21 END
22 ERROR 9101                                   'Generates error 9101 if the server is down during
                                                 processing.
23 *LHLT2:CLOSE #1
24 HLT
25 END
```

[Explanation]

(1) This command is used in a combination with the OPEN instruction. The following lists the meanings and values for the types of the files specified by the OPEN instruction.

Type of file to be opened	Meaning		Value
File	Indicates whether or not the file has been opened. 1 is always returned after executing the OPEN instruction.		1: Already opened. -1: The file number is undefined (not opened).
Communication line Ethernet	Indicates whether or not connection is made with the counterpart.	For server setting	1: Client is already connected. 0: Client is not connected. -1: The file number is undefined (not opened).
		For client setting	1: Already connected to the server. (Connection has been made.) 0: Not connected to the server. (Connection has not been made. Equivalent to when the server is down after being opened.) -1: The file number is undefined. (When the file has not been opened, or has been opened while the server is down.)

[Related Instruction]

OPEN

[Related Parameters]

COMDEV, CPRE**, NETMODE

C_COM

[Function]

Sets the parameters for the line to be opened by the OPEN instruction. This is used when the communication destination is changed frequently.

* Character string type
* Only for a client with the Ethernet option.

[Format]

C_COM (<communication line number>) = “ETH: <server side IP address> [, <port number>]”

[Terminology]

ETH:	An identifier to indicate that the target is an Ethernet
<Communication line number>	The number of the COM to be specified by the OPEN instruction (The line type is assigned by the COMDEV parameter.) Specify 1 through 8.
<Server side IP address>	Server side IP address (May be omitted.)
<Port number>	Port number on the server side (If omitted, the set value of the NETPORT parameter is used.)

[Reference Program]

Example when OPT12 is set in the second element of the COMDEV parameter

```
1 C_COM(2)="ETH:192.168.0.10,10010"   ' Set the IP address of the communication destination server
                                        corresponding to communication line COM2
2 *LOPEN1:OPEN "COM2:" AS #1          ' As 192.168.0.10 and the port number as 10010, and then open the line.
3 IF M_OPEN(1)<>1 THEN *LOPEN1        ‘ Loops if unable to connect to the server.
4 PRINT #1, "HELLO"                   ‘ Sends a character string.
5 INPUT #1, C1$                       ‘ Receives a character string.
6 CLOSE #1                            ‘ Closes the line.
7 C_COM(2)="ETH:192.168.0.11,10011"   ‘ Set the IP address of the communication destination server
                                        corresponding to communication line COM2
8 *LOPEN2:OPEN "COM2:" AS #1          ‘ As 192.168.0.11 and the port number as 10011, and then open the line.
9 IF M_OPEN(1)<>1 THEN *LOPEN2        ‘ Loops if unable to connect to the server.
10 PRINT #1, C1$                      ‘ Sends a character string.
11 INPUT #1, C2$                      ‘ Receives a character string.
12 CLOSE #1                           ‘ Closes the line.
13 HLT                                ‘ Halts the program.
14 END                                ‘ Ends.
```

[Description]

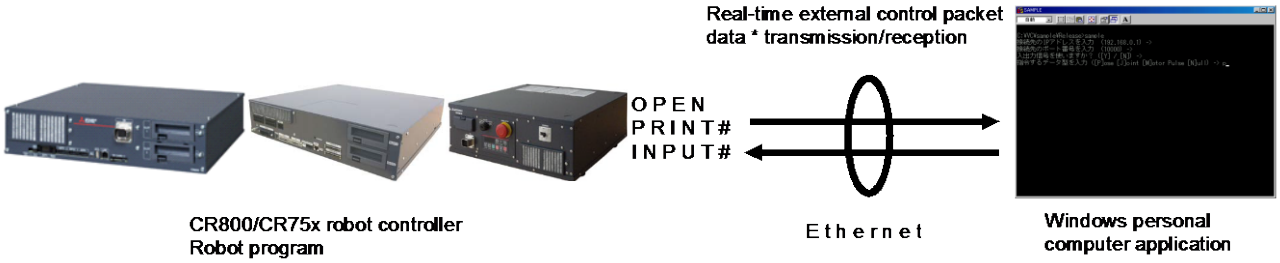
(1) It is not necessary to use this command when the communication counterpart of the robot controller is specified with the NETHSTIP and NETPORT parameters and the specified communication counterpart will not be changed at all.
(2) Currently, this function is valid only for a client of a data link with the Ethernet.
(3) Because the communication parameters of the OPEN instruction are set, it is necessary to execute this command before the OPEN instruction.
(4) When the power is turned on, the set values specified by the NETHSTIP and NETPORT parameters are used. When this command is executed, the values specified by the parameters of this command are changed temporarily. They are valid until the power is turned off. When the power is turned on again, the values revert to the original values set by the parameters.
(5) If this command is executed after the OPEN instruction, the current open status will not change. In such a case, it is necessary to close the line with the CLOSE instruction once, and then execute the OPEN instruction again.
(6) If an incorrect syntax is used, an error occurs when the program is executed, not when the program is edited.

[Related Parameters]

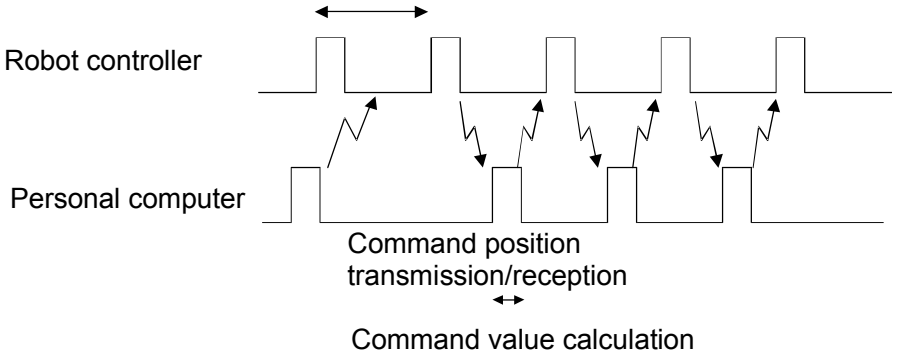
NETHSTIP, NETPORT

4.2. Real-time external control function

The robot motion movement control can retrieve the position command at real-time in cycle units, and move to the commanded position. It is also possible to monitor the input/output signals or output the signals simultaneously.
Using the robot language MXT command, real-time communication (command/monitor) is carried out with communication.



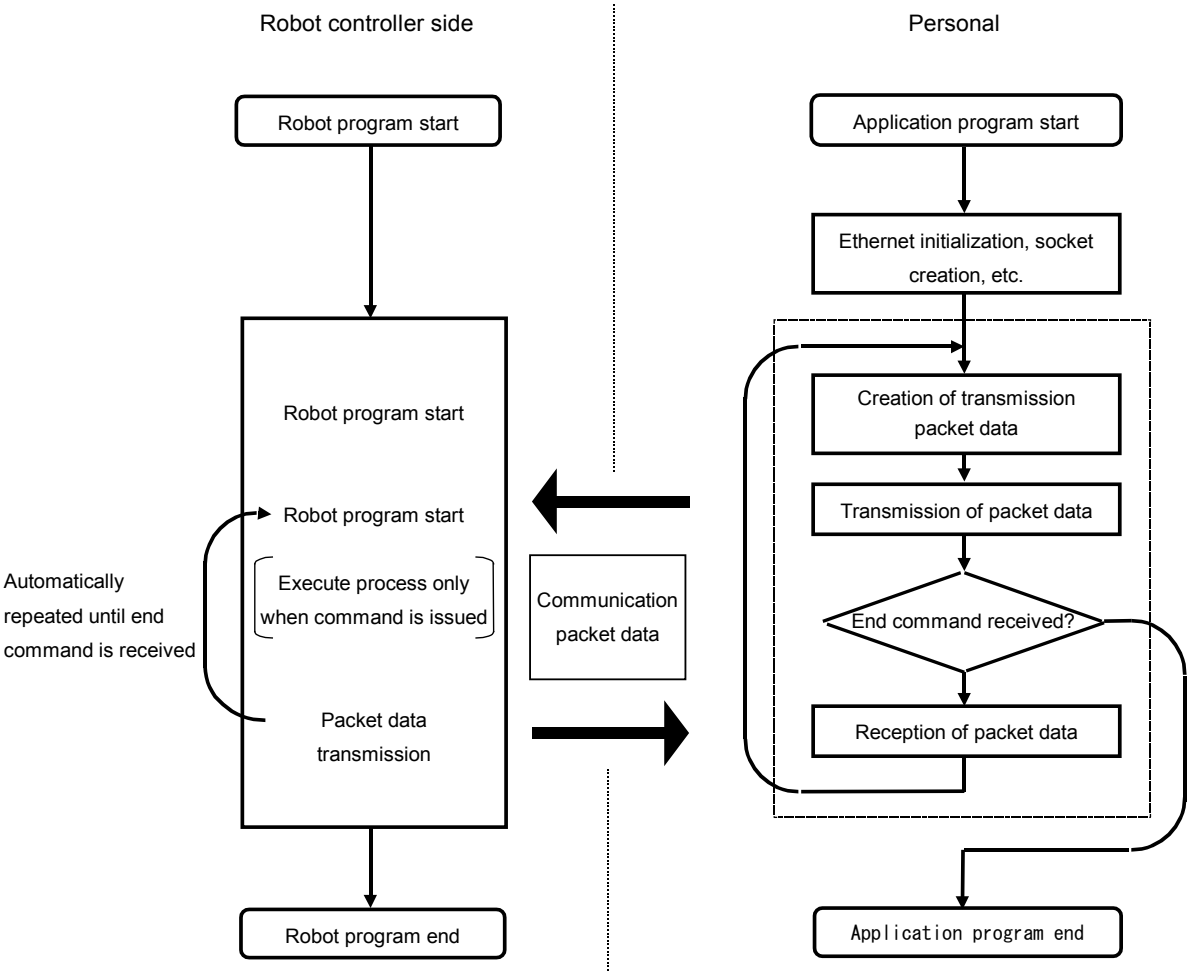
Motion movement control cycle (CR750/CR751 series: approx. 7.1 ms,
CR800 series: approx. 3.5 ms (If user mechanical is set, approx. 7.1 ms))



The following table lists the position command data for giving the target move position from the personal computer to the robot for each hour of the motion operation control cycle, and the monitor data types from the robot.
For more information about communication data, see Section 4.2.1, “Command Explanation” and Section 4.2.2, “Communication Data Packet Explanation” in this document.

Position command data type	Monitor data type
[1] Rectangular coordinate data [2] Joint coordinate data [3] Motor pulse coordinate data	[1] Rectangular coordinate data [2] Joint coordinate data [3] Motor pulse coordinate data [4] Rectangular coordinate data (command value after filter processing) [5] Joint coordinate data (command value after filter processing) [6] Motor pulse coordinate data (after filter processing) [7] Rectangular coordinate data (encoder feedback value) [8] Joint coordinate data (encoder feedback value) [9] Motor pulse coordinate data (encoder feedback value) [10] Current command (%) [11] Current feedback (%)

* Flow of real-time external control



4.2.1. Explanation of command

Either the MELFA-BASIC V command languages can be used with the real-time external control function.
Note that the meanings of the arguments differ for the MELFA-BASIC V commands. (Refer to following format and terminology.)
Refer to section "4.2.2 Explanation of communication data packet" for details on the structure of the communication data packet used with this function.

MXT (Move External)

[Function]

The absolute position data is retrieved from an external source at each controller control time (currently approx. 7.1msec), and the robot is directly moved.

[Format]

MXT <File No.>, <Reply position data type> [, <Filter time constant>]

[Terminology]

<File No.>	Describe a number between 1 and 8 assigned with the OPEN command. If the communication destination is not designated with the OPEN command, an error will occur, and communication will not be possible. In addition, data received from a source other than the communication destination will be ignored.
<Replay position data type>	Designate the type of the position data to be received from the personal computer. A XYZ/joint/motor pulse can be designated. 0: XYZ coordinate data 1: Joint coordinate data 2: Motor pulse coordinate data
<Filter time constant>	Designate the filter time constant (msec). If 0 is designated, the filter will not be applied. (0 will be set when omitted.) A filter is applied on the reception position data, an obtuse command value is created and output to the servo.

[Reference Program]

```
1 OPEN "ENET:192.168.0.2" AS #1     'Ethernet communication destination IP address
2 MOV P1                            'Move to P1
3 MXT1,1,50                         'Move with real-time external control with filter time constant set to
                                     50msec
4 MOV P1                            'Move to P1
5 HLT                               'Halt program
```

[Explanation]

* When the MXT command is executed, the position command for movement control can be retrieved from the personal computer connected on the network. (One-on-one communication)
* One position command can be retrieved and operated at the operation control time (currently 7.1msec).
* Operation of MXT command
  1) When this command is executed with the controller, the controller enters the command value reception enabled state.
  2) When the controller receives the command value from the personal computer, it will output the received command value to the servo within the next control process cycle.
  3) After the command value is sent to the servo, the controller information, such as the current position is sent from the controller to the personal computer.
  4) A reply is made from the controller to the personal computer only when the command value from the personal computer is sent to the controller.
  5) If the data is not received, the current position is maintained.
  6) When the real-time external command end command is received from the personal computer, the MXT command is ended.
  7) When the operation is stopped from the operating panel or external input, the MXT command will be halted, and the transmission/reception will also be halted until restart.
* The timeout is designated with the parameter MXTTOUT.
* One randomly designated (head bit, bit width) input/output signal can be transmitted and received simultaneously with the position data.
* A personal computer with sufficient processing speed must be used to command movement in the movement control time.

4.2.2. Explanation of communication data packet

The structure of the communication data packet used with the real-time external control function is explained in this section. The same communication data packet for real-time external control is used for commanding the position and for monitoring. The contents differ when transmitting (commanding) from the personal computer to the controller and when receiving (monitoring) from the controller to the personal computer.

(1) Communication data packet.

Name	Data type	Explanation
Command	unsigned short (2-byte)	Designate the validity of the real-time external command, and the end. 0 // Real-time external command invalid 1 // Real-time external command valid 255 // Real-time external command end
Transmission data type designation SendType	unsigned short (2-byte)	1) When transmitting (commanding) from the personal computer to the controller, designate the type of position data transmitted from the personal computer. There is no data at the first transmission. 0 // No data 1 // XYZ data 2 // Joint data 3 // Motor pulse data 2) When receiving (monitoring) from the controller to the personal computer, indicate the type of position data replied from the controller. 0 // No data 1 // XYZ data 2 // Joint data 3 // Motor pulse data 4 // XYZ data (Position after filter process) 5 // Joint data (Position after filter process) 6 // Motor pulse data (Position after filter process) **7 // XYZ data (Encoder feedback value)** **8 // Joint data (Encoder feedback value)** **9 // Motor pulse data (Encoder feedback value)** **10 // Current command [%]** **11 // Current feedback [%]** * It is the same as RecvType. You may use whichever.

Name	Data type	Explanation
Reply data type designation RecvType	unsigned short (2-byte)	1) When transmitting (commanding) from the personal computer to the controller, designate the type of data replied from the controller. 0 // No data 1 // XYZ data 2 // Joint data 3 // pulse data 4 // XYZ data (Position after filter process) 5 // Joint data (Position after filter process) 6 // Motor pulse data (Position after filter process) **7 // XYZ data (Encoder feedback value)** **8 // Joint data (Encoder feedback value)** **9 // Motor pulse data (Encoder feedback value)** **10 // Current command [%]** **11 // Current feedback [%]** 2) When receiving (monitoring) from the controller to the personal computer, indicate the type of position data replied from the controller. 0 // No data 1 // XYZ data 2 // Joint data 3 // Motor pulse data 4 // XYZ data (Position after filter process) 5 // Joint data (Position after filter process) 6 // Motor pulse data (Position after filter process) **7 // XYZ data (Encoder feedback value)** **8 // Joint data (Encoder feedback value)** **9 // Motor pulse data (Encoder feedback value)** **10 // Current command [%]** **11 // Current feedback [%]** * It is the same as RecvType. You may use whichever.
Reservation reserve	unsigned short (2byte)	Not used.
Position data Pos / jnt / pls	POSE, JOINT or PULSE (40-byte) * Refer to strdef.h in the sample program for details on each data structure.	1) When transmitting (commanding) from the personal computer to the controller, designate the command position data transmitted from the personal computer. Set this to the same data type as that designated for the transmission data type designation. 2) When receiving (monitoring) from the controller to the personal computer, this indicates the position data replied from the controller. The data type is shown in SendType (= RecvType) . The contents of data are common to command/monitor. POSE // XYZ type [mm/rad] JOINT // Joint type [rad] PULSE // Motor pulse type [the pulse] or Current type [%].

Name	Data type	Explanation
Transmission input/output signal data designation SendIOType	unsigned short (2-byte)	1) When transmitting (commanding) from the personal computer to the controller, designate the data type of the input/output signal transmitted from the personal computer. Designate "No data" when not using this function. 2) When receiving (monitoring) from the controller to the personal computer, this indicates the data type of the input/output signal replied from the controller. The contents of the data are common. 0 // No data 1 // Output signal 2 // Input signal
Reply input/output signal data designation RecvIOType	unsigned short (2-byte)	1) When transmitting (commanding) from the personal computer to the controller, designate the data type of the input/output signal replied from the controller. Designate "No data" when not using this function. 0 // No data 1 // Output signal 2 // Input signal 2) When receiving (monitoring) from the controller to the personal computer, Not used.

Input/output signal data
BitTop
BitMask
IoData
	unsigned short
	unsigned short
	unsigned short
	(2-byte x 3)
		1) When transmitting (commanding) from the personal computer to the controller, designate the output signal data transmitted from the personal computer.
		
		2) When receiving (monitoring) from the controller to the personal computer, this indicates the input/output signal data replied from the controller.
		
		The contents of the data are common.
		BitTop; // Head bit No. of input or output signal
		BitMask; // Bit mask pattern designation (valid only for commanding)
		IoData; // Input or output signal data value (for monitoring)
		Output signal data value (for commanding)
		* Data is 16-bit data

Timeout time counter value Tcount	unsigned short (2-byte)	1) When transmitting (commanding) from the personal computer to the controller, Not used. 2) When receiving (monitoring) from controller to personal computer, if the timeout time parameter MXTTOUT is a value other than -1, this indicates the No. of times communication with the controller was not possible. When the No. of times is counted and reaches the maximum value, the value will return to the minimum value 0, and the count will be repeated. This is set to 0 when the MXT command is started.
Counter value for communication data Ccount	unsigned long (4-byte)	1) When transmitting (commanding) from the personal computer to the controller. 2) When receiving (monitoring) from controller to personal computer, this indicates the No. of communication times.

Name	Data type	Explanation
Reply data-type specification addition 1 RecvType1	**unsigned short (2-byte)**	**It is the same as reply data-type specification (RecvType). Do not use it for instructions.**
Reservation 1 reserve1	**unsigned short (2-byte)**	**Not used.**
Data addition 1 pos / jnt / pls	**Any of POSE/JOINT/PULSE. (40-byte)**	**It is the same as data of pos/jnt/pls. Do not use it for instructions.**
Reply data-type specification addition 2 RecvType2	**unsigned short (2-byte)**	**It is the same as reply data-type specification (RecvType). Do not use it for instructions.**
Reservation 2 Reserve2	**unsigned short (2-byte)**	**Not used.**
Data addition 2 pos / jnt / pls	**Any of POSE/JOINT/PULSE. (40-byte)**	**It is the same as data of pos/jnt/pls. Do not use it for instructions.**
Reply data-type specification addition 3 RecvType3	**unsigned short (2-byte)**	**It is the same as reply data-type specification (RecvType). Do not use it for instructions.**
Reservation 3 Reserve3	**unsigned short (2-byte)**	**Not used.**
Data addition 3 pos / jnt / pls	**Any of POSE/JOINT/PULSE. (40-byte)**	**It is the same as data of pos/jnt/pls. Do not use it for instructions.**

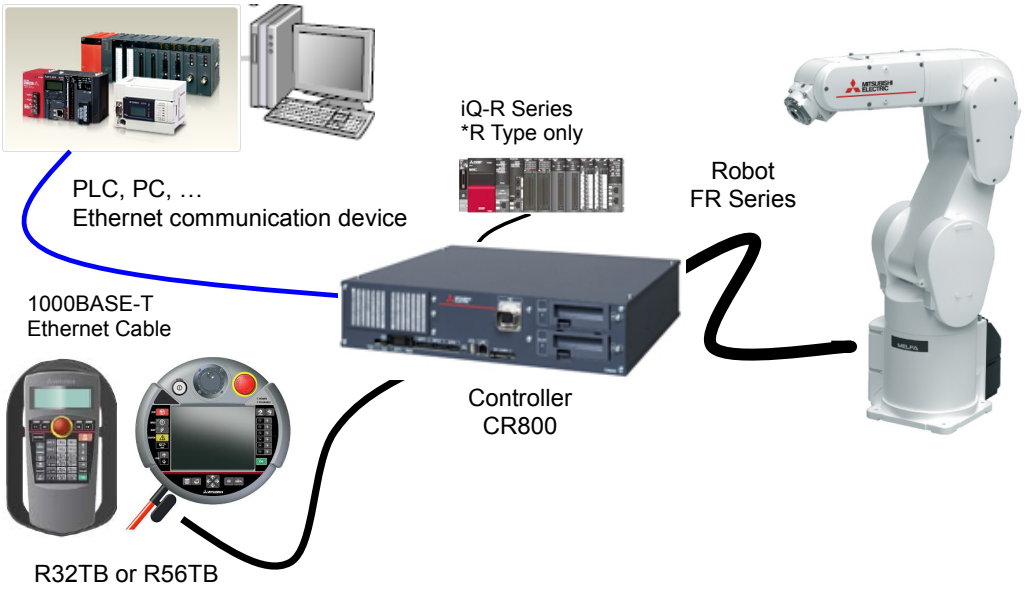
5. Real-time monitor functional

5.1. Overview

This function is obtained from the Ethernet communication device, such as the tool point speed and current position of the robot.

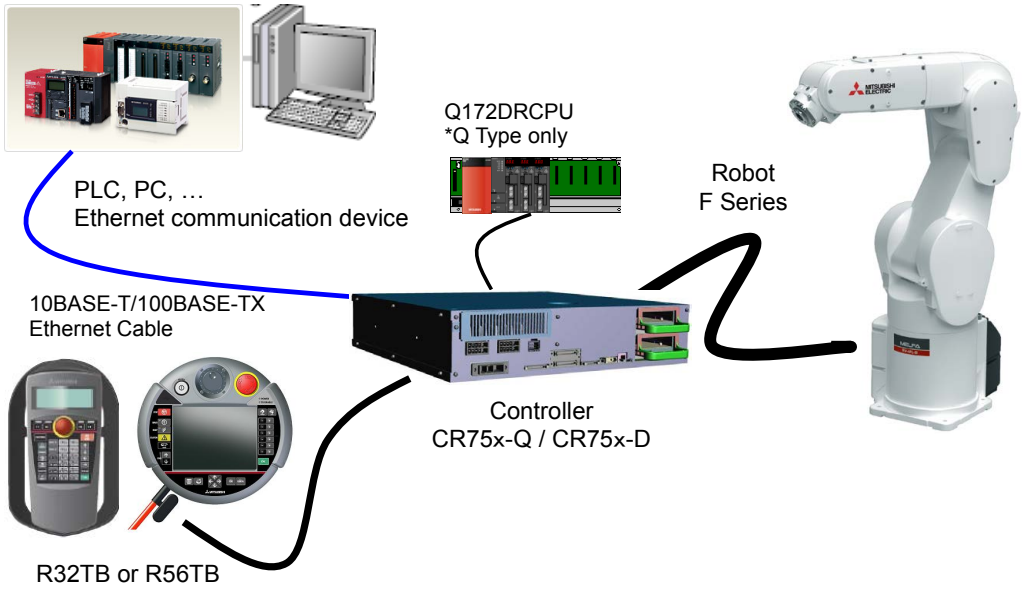
The Ethernet UDP communication is the ability to monitor in real-time, such as joint position data and orthogonal position of the robot controller.

5.1.1. CR800 series



System configuration (Example)

5.1.2. CR75n series



System configuration (Example)

5.2. Supported version

Controller type	Version	Remarks
CR75x-Q	Ver.R3n	RT2 Oscillograph function corresponding Ver.R4b or later
CR75x-D	Ver.S3n	RT2 Oscillograph function corresponding Ver.R4b or later

The CR800 controller is supported by all the software versions.

5.3. Setup

It is a set-up procedure of example conditions.

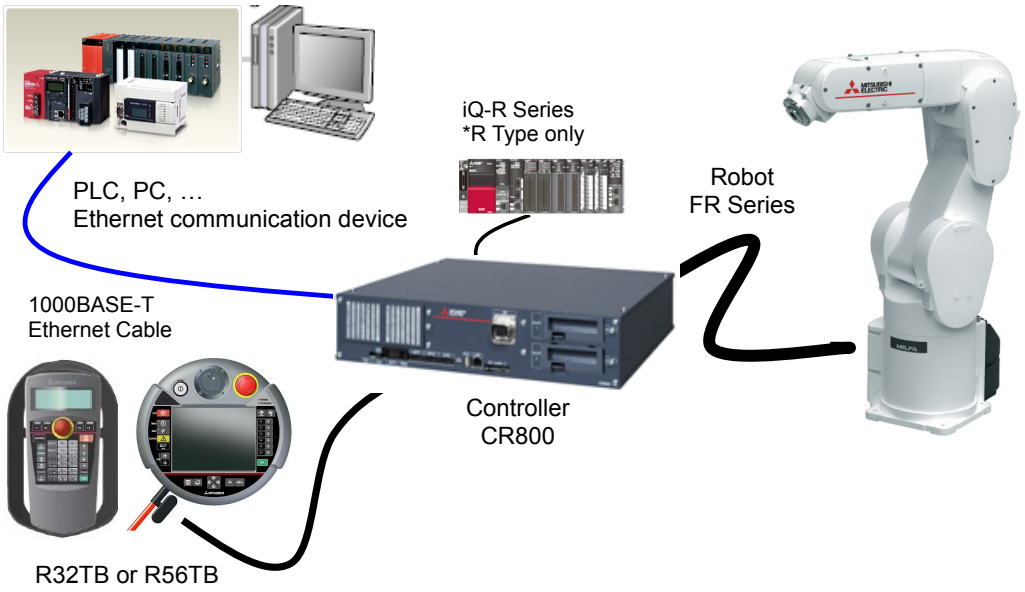
5.3.1. CR800 series

List conditional example

IP address of Robot controller	192.168.0.20
IP address of PC	192.168.0.2
Port number for Real-time monitor	12000, 12001 Receive port = 12000 , Send port = 12001

(1) Connecting the controller and personal computer

Connect the Ethernet cable to the connector of the controller. When the hub is used, use the straight cable. Or when the personal computer and controller are connected to each other one to one, use the cross cable.



(2) Setting the controller parameters

Set the parameters of the robot controller as shown in Table. For more information about parameters, see 5.7.

Parameter setting example

Parameter	Before/after change	Parameter value
NETIP	before	192.168.0.20 (D type Robot controller) 192.168.0.10 (R type Robot controller)
	after	Same as above (unchanged)
MONMODE	before	0
	after	1
MONPORT	before	12000, 0
	after	12000, 12001* Only when a change is required

(3) Setting the personal computer

To suit your network, please perform the communication settings. Please specify the UDP protocol of Ethernet communication.

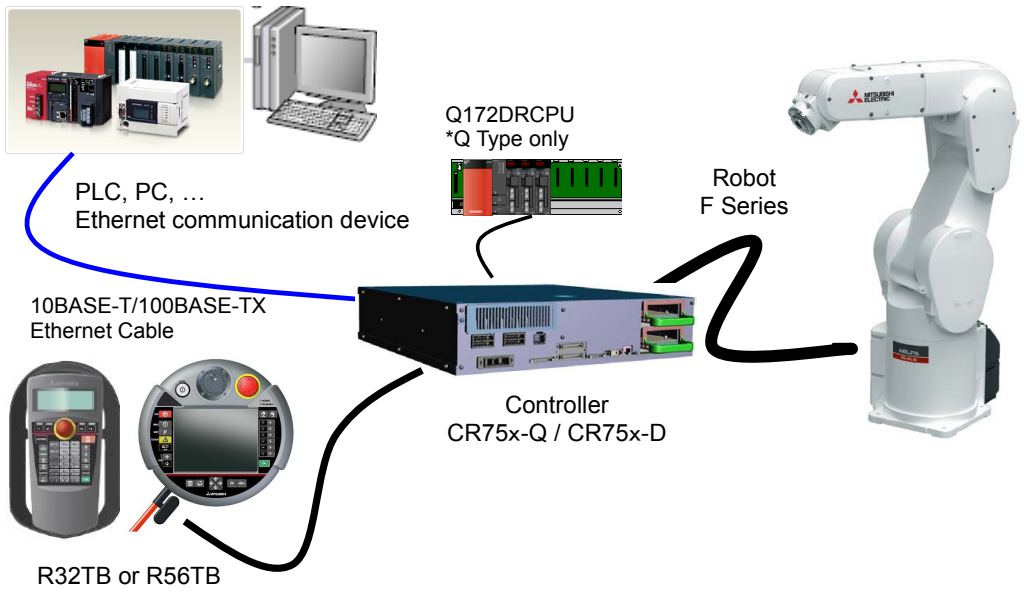
5.3.2. CR75n series

Example of conditions

IP address of Robot controller	192.168.0.20 (D type Robot controller)
IP address of PC	192.168.0.2
Port number for Real-time monitor	12000, 12001 Receive port = 12000 , Send port = 12001

(1) Connecting the controller and personal computer

Connect the Ethernet cable to the connector of the controller. When the hub is used, use the straight cable. Or when the personal computer and controller are connected to each other one to one, use the cross cable.



(2) Setting the controller parameters

Set the parameters of the robot controller as shown in Table. For more information about parameters, see 5.7.

Parameter setting example

Parameter	Before/after change	Parameter value
NETIP	before	192.168.0.20 (D type Robot controller) 192.168.100.1 (Q type Robot controller)
	after	Same as above (Default value)
MONMODE	before	0
	after	1
MONPORT	before	12000, 0
	after	12000, 12001* Only when a change is required

(3) Setting the personal computer

To suit your network, please perform the communication settings. Please specify the UDP protocol of Ethernet communication.

5.4. Start of monitor / End of monitor

Explain start of monitor and end of the monitor.

(1) Start of monitor

Set the data type ID as a starting packet data, set data output start (1) on the command, you want to monitor the return data type 1-4 In addition, it sends to the robot controller.

If the start packet data is accepted normally, the robot controller continuously sends reply packet data (output data) to the Ethernet communication device by every control cycle of the robot controller (refer to the following).

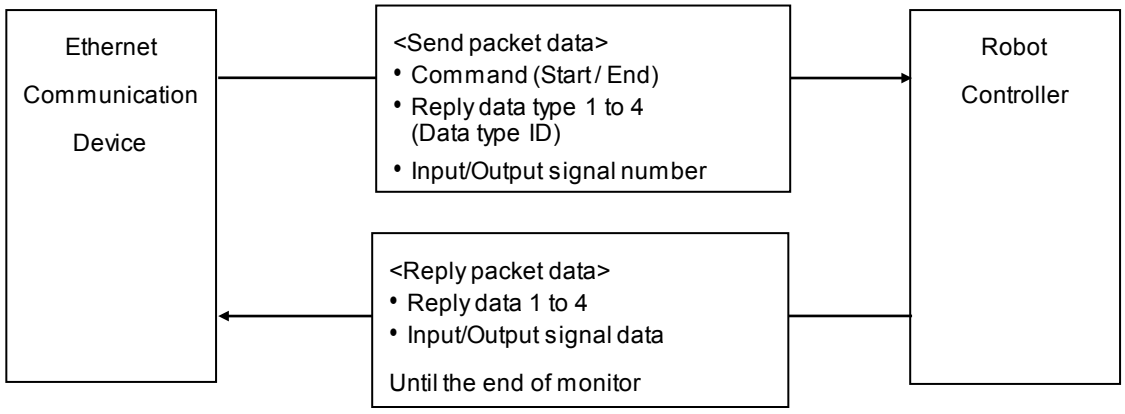
Controller	Control cycle (*1)
CR750/CR751 series	Approx. 7.11 msec
CR800 series	Approx. 3.5 msec (*If user mechanical is set, approx. 7.11 msec)

*1 Because it depends on the performance of the communication path and the communication device, the period is not guaranteed.

(2) End of monitor

It will be sent to the robot controller by setting the data output end (255) to the command end packet data. If accepted, the robot controller to exit the sending of the reply packet data.

If you want to change the type of output data on the monitor the way, it sends a start packet data.



About communication device

• Communication device is the only one. It is not possible to communicate with the other device of two or more.

• It is disconnected from the communication device in communication first, and then communicates with a corresponding later

⚠Caution

The data output from the robot, for that is sent (UDP) communication via Ethernet without the retransmission process, because there is the case that such noise environments, such as the transmission of data or a wrong data dropout occurs, the guarantee of data is not possible.

5.5. Explanation of communication data packet

It describes the structure of the communication packet data to be used in real-time monitoring function. To the robot controller, I will use the same packet structure on both send and receive from Ethernet communication device. Storage method of data is little-endian. Real data in 32-bit real number is IEEE754 standard method. Data packet size is 196 bytes fixed.

Table 5-1 Data packet

Name	Data type	Explanation	Address
Command	unsigned short 2 byes	Specifies the start or end of the real-time monitoring function. 1 // Start of the real-time monitor 255 // End of the real-time monitor	0-1
Not used(reserve)	2 byes	Not used	2-3
Reply data type 1	unsigned short 2 byes	1) Communication device → Robot controller Specifies the <Data type ID> of the data that you want to monitor. 2) Robot controller → Communication device Echo back of send 1) *Data type ID see [5.6 Data type ID]	4-5
Not used(reserve)	2 byes	Not used	6-7
Reply data 1	Data structure POSE, JOINT, PULSE, ROBMON, FORCE or FLOAT8 40 bytes * Each data structure is described in Table Table 5-2, Table 5-3 Table 5-4, Table 5-5 Table 5-6, Table 5-7	1) E Communication device → Robot controller Not used. Set to zero. 2) Robot controller → Communication device The output data sent back from the controller. Data type is seeing in the return data type. *Data structure POSE // XYZ type [mm/rad] JOINT // Joint type [rad] PULSE // Motor pulse type [pulse] or Current type[%] FORCE // Force sensor type ROBMON // Robot movement information FLOAT8 // General purpose, float×8	8-47
Input signal number of the top * Ver.R4b/S4b or later	unsigned short 2 byes	1) Communication device → Robot controller Input signal number of the top (0 to 32767) 2) Robot controller → Communication device Echo back of send 1)	48-49
Output signal number of the top * Ver.R4b/S4b or later	unsigned short 2 byes	1) Communication device → Robot controller Output signal number of the top (0 to 32767) 2) Robot controller → Communication device Echo back of send 1)	50-51
Input signal data * Ver.R4b/S4b or later	unsigned long 4 byes	1) Communication device → Robot controller Not used. Set to zero. 2) Robot controller → Communication device Input signal data(0x00000000-0xffffffff)	52-53
Output signal data * Ver.R4b/S4b or later	unsigned long 4 byes	1) Communication device → Robot controller Not used. Set to zero. 2) Robot controller → Communication device Output signal data(0x00000000-0xffffffff)	56-57
Communication data counter	unsigned long 4 byes	1) Communication device → Robot controller Not used. Set to zero. 2) Robot controller → Communication device The number of communications. To return to the minimum value 0 and the maximum value by integrating.	60-63
Reply data type 2	unsigned short 2 byes	Same Reply data type 1	64-65
Not used(reserve)	2 byes	Not used	66-67
Reply data 2	POSE, JOINT, PULSE, ROBMON, FORCE or FLOAT8 40 bytes	Same Reply data 1	68-107
Reply data type 3	unsigned short 2 byes	Same Reply data type 1	108-109
Not used (reserve)	2 byes	Not used	110-111
Reply data 3	POSE, JOINT, PULSE, ROBMON, FORCE or FLOAT8 40 bytes	Same Reply data 1	112-151
Reply data type 4	unsigned short 2 byes	Same Reply data type 1	152-153
Not used (reserve)	2 byes	Not used	154-155
Reply data 4	POSE, JOINT, PULSE, ROBMON, FORCE or FLOAT8 40 bytes	Same Reply data 1	156-195

Table 5-2 POSE (XYZ) data structure

X element	4 bytes float	XYZ data [mm / rad], 40 bytes * Data type 1 and 7 is unit of radians. Data type 1001 and 1007 is unit of degrees.
Y element	4 bytes float	
Z element	4 bytes float	
A element	4 bytes float	
B element	4 bytes float	
C element	4 bytes float	
L1 element	4 bytes float	
L2 element	4 bytes float	
FL1(Structure flag 1)	4 bytes long	
FL2(Structure flag 2)	4 bytes long	

Table 5-3 JOINT data structure

J1 element	4 bytes float	Joint data [rad], 32 bytes * Data type 2 and 8 is unit of radians. Data type 1002 and 1008 is unit of degrees.
J2 element	4 bytes float	
J3 element	4 bytes float	
J4 element	4 bytes float	
J5 element	4 bytes float	
J6 element	4 bytes float	
J7 element	4 bytes float	
J8 element	4 bytes float	
Not used	8 bytes	Not used. Value is zero.

Table 5-4 PULSE (Pulse/%) data structure

M1 element	4 bytes long	Motor pulse data or current data [0.1% rate] , 32 bytes
M2 element	4 bytes long	
M3 element	4 bytes long	
M4 element	4 bytes long	
M5 element	4 bytes long	
M6 element	4 bytes long	
M7 element	4 bytes long	
M8 element	4 bytes long	
Not used	8 bytes	Not used. Value is zero.

Table 5-5 FORCE (N/Nm) data structure

F1 element	4 bytes float	Force sensor data[N, Nm] , 32 bytes
F2 element	4 bytes float	
F3 element	4 bytes float	
F4 element	4 bytes float	
F5 element	4 bytes float	
F6 element	4 bytes float	
Not used	16 bytes	Not used. Value is zero.

Table 5-6 ROBMON (Robot information) data structure

Tool point speed (feedback)	4 bytes float	Speed of a tool center point (feedback) [mm/s]
Remaining distance (feedback)	4 bytes float	The remaining distance to the target position (in mm) while the robot is moving (feedback).
Tool point speed (command)	4 bytes float	Speed of a tool center point (command) Same as status variable values “M_RSpd”
Remaining distance (command)	4 bytes float	The remaining distance to the target position (in mm) while the robot is moving (command). Same as status variable values “M_RDst”.
Gap of command and feedback	4 bytes float	The gap of a command position and a feedback position. Same as status variable values “M_Fbd”.
Transport factor (command)	2 bytes integer	Speed of a tool center point (feedback)
Acceleration state (command)	2 bytes integer	The current acceleration/deceleration status. (command) [0=Stopped,1=Accelerating, 2= Constant speed, 3= Decelerating] Same as status variable values “M_AclSts”.
Step number	2 bytes integer	Step number (Only slot 1), (1-32767)
Program name	6 bytes character	Program name (Only slot 1) Max Program name is 6 characters
Controller temperature	2 bytes integer	Controller temperature [0.1°C]
Not used	2 bytes	Not used.
Monitoring counter	4 bytes long	After power-on, +1 is counted from 0 in an operation control cycle unit (64/9 (*1)). The counting repeats in the range of 0 to 4294967295.

(*1) CR750/CR751 series: approx. 7.111 ms, CR800 series: approx. 3.5 ms (If user mechanical is set, approx. 7.1 ms)

Table 5-7 FLOAT8 (short real) data structure

float 1	4 bytes float	float (short real), 32 bytes
float 2	4 bytes float	
float 3	4 bytes float	
float 4	4 bytes float	
float 5	4 bytes float	
float 6	4 bytes float	
float 7	4 bytes float	
float 8	4 bytes float	
Not used	8 bytes	Not used. Value is zero.

5.6. Data type ID

The type of data that can be monitored in real-time monitor function.

Table 5-8 Data type ID

ID	Contents	Data structure	Ver.
0	no data	–	R3n/S3n or later
1	XYZ position (Command) *Angle in radians	POSE	
2	Joint position (Command) *Angle in radians	JOINT	
3	Motor pulse position (Command)	PULSE (Long×8)	
7	XYZ position (Feedback) *Angle in radians	POSE	
8	Joint position (Feedback) *Angle in radians	JOINT	
9	Motor pulse position (Feedback)	PULSE (Long×8)	
10	Current command [0.1% rate]	PULSE (Long×8)	
11	Current feedback [0.1% rate]	PULSE (Long×8)	
12	Robot information	ROBMON	
13	Position droop	PULSE (Long×8)	R4b/S4b or later
14	Speed (Command) [rpm]	PULSE (Long×8)	
15	Speed (Feedback) [rpm]	PULSE (Long×8)	
16	Axis load level [%]	FLOAT8(Float×8)	
17	Encoder temperature [°C]	PULSE (Long×8)	
18	Encoder miss count	PULSE	
19	Motor voltage [V]	PULSE (Long×8)	
20	Regeneration level [%]	PULSE (Long×8)	
21	Tolerable command + [0.1% rate]	PULSE (Long×8)	
22	Tolerable command - [0.1% rate]	PULSE (Long×8)	
23	RMS current [0.1% rate]	PULSE (Long×8)	
101	Force sensor current position xyz[N]abc[Nm]	FORCE (Float×8)	R3n/S3n or later
102	Force sensor original data (after offset cancel) xyz[N]abc[Nm]	FORCE (Float×8)	
103	Force sensor original data (before offset cancel) xyz[N]abc[Nm]	FORCE (Float×8)	
104	Position command of the force sensor correction	POSE	
111	COL presumed torque [0.1% rate]	PULSE (Long×8)	
112	COL threshold + [0.1% rate]	PULSE (Long×8)	
113	COL threshold - [0.1% rate]	PULSE (Long×8)	
1001	XYZ position (Command) *Angle in degrees	POSE	R4b/S4b or later
1002	Joint position (Command) *Angle in degrees	JOINT	
1007	XYZ position (Feedback) *Angle in degrees	POSE	
1008	Joint position (Feedback) *Angle in degrees	JOINT	
1010	Current command [Arms]	FLOAT8 (Float×8)	
1011	Current feedback [Arms]	FLOAT8 (Float×8)	
1012	Tolerable command + [Arms]	FLOAT8 (Float×8)	
1013	Tolerable command - [Arms]	FLOAT8 (Float×8)	
1014	RMS current [Arms]	FLOAT8 (Float×8)	

5.7. Parameters

Table 5-9 Parameter

Parameter	Parameter name	No. of arrays	Details explanation	Factory setting
Ethernet real-time monitor	MONMODE	Integer 1	Switch to enable or disable real-time monitoring function 0: Disable 1: Enable	0
	MONPORT	Integer 2	Specify the receive port number and the send port number of real-time monitor function. (0 to 65535) First element: Receive port number Second element: Send port number Second element: 0 is special value, reply to the sender port number that is set to UDP header information in the packet data start the robot controller has received	12000, 0

5.8. Error

Table 5-10 Error

Error number	Error cause and measures	
L.7810	Error message	NETPORT/MONPORT parameter error
	Cause	The element of NETPORT(1) and MONPORT(1/2) overlap.
	Measures	Please set not to overlap to another port number.
	Detail	UDP port number to be used for real-time monitoring function and real-time external control is duplicated. That you cannot use the same port number, please change to a different port number.

6. SLMP Connection

6.1. Function Overview

SLMP is a common protocol for seamless communication between applications. Users do not have to be concerned with network layers or boundaries. SLMP communications are available among devices that can transfer messages by SLMP (programmable controllers, personal computers, HMIs and others). (For the details of the SLMP compatibility of external devices, refer to the Instruction Manual of external devices.)
The FR Series supports the SLMP communication server function.

6.2. Specifications

The following section describes the specifications of the SLMP-compatible device and SLMP communication.

6.2.1. SLMP Specifications

The SLMP specifications for the message sent by an external device or with the communication protocol support function are as follows.

Item	Communication data code	Description	Reference
SLMP	· ASCII code · Binary code	This is the same message format as that for MC protocol QnA-compatible 3E frame and 4E frame.	6.4.1 Request Messages

Compared to communication with ASCII code, communication with binary code involves approximately half the amount of communication data.

6.2.2. Parameters

Specify settings with the following parameters.

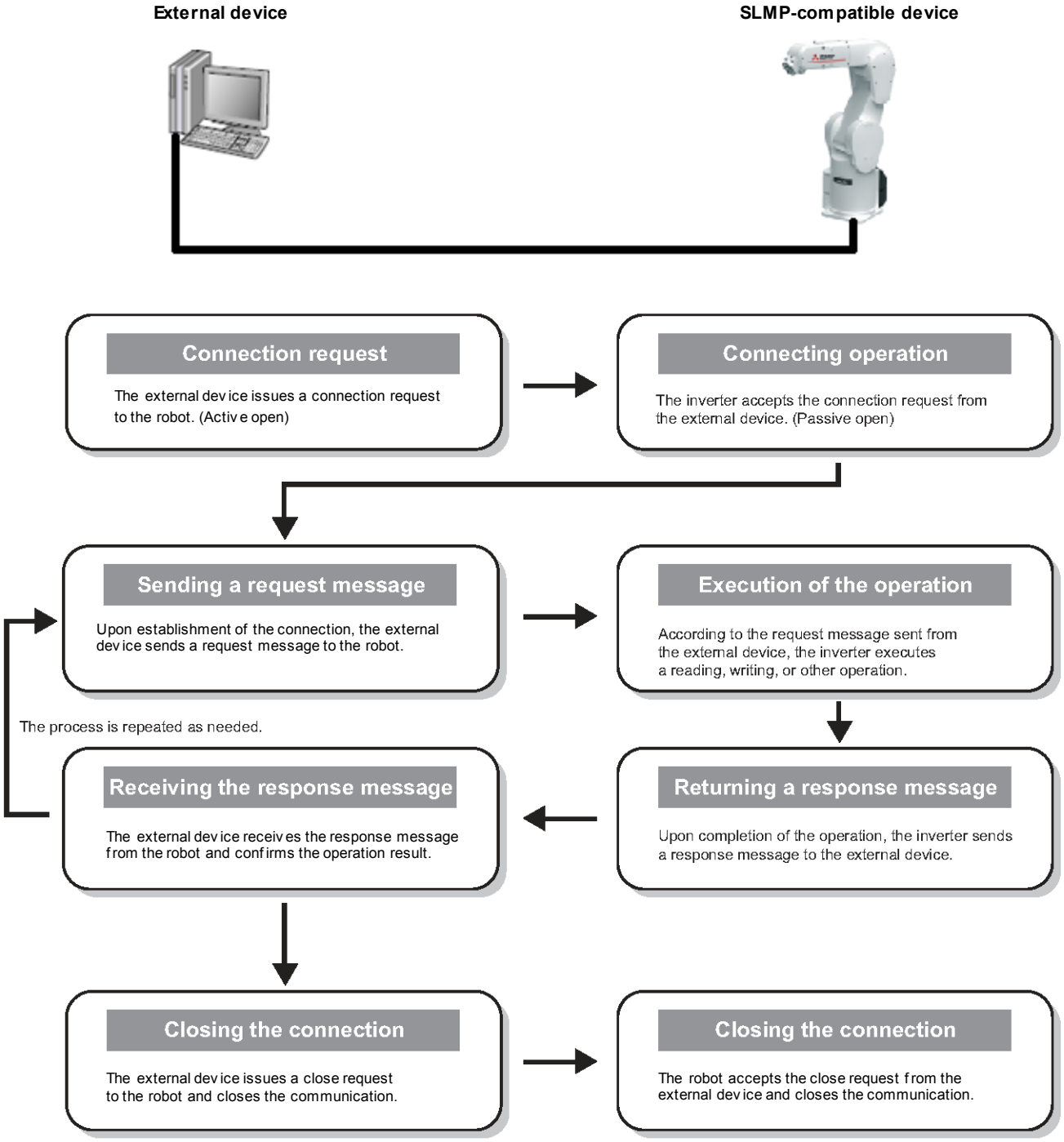
Parameter name	No. of arrays No. of characters	Description	Factory setting
SLMPPORT	Integer 1	Set the SLMP server communication port No. (1024 to 65535)	45237
SLMPCP	Integer 1	Set the SLMP server communication protocol. 0: TCP 1: UDP	1
SLMPNWNO	Integer 1	Set the SLMP network number. (1 to 239)	1
SLMPNDID	Integer 1	Set the SLMP station number. (1 to 120)	1

6.3. SLMP Communication Procedure

An external device and an SLMP-compatible device communicate as follows.

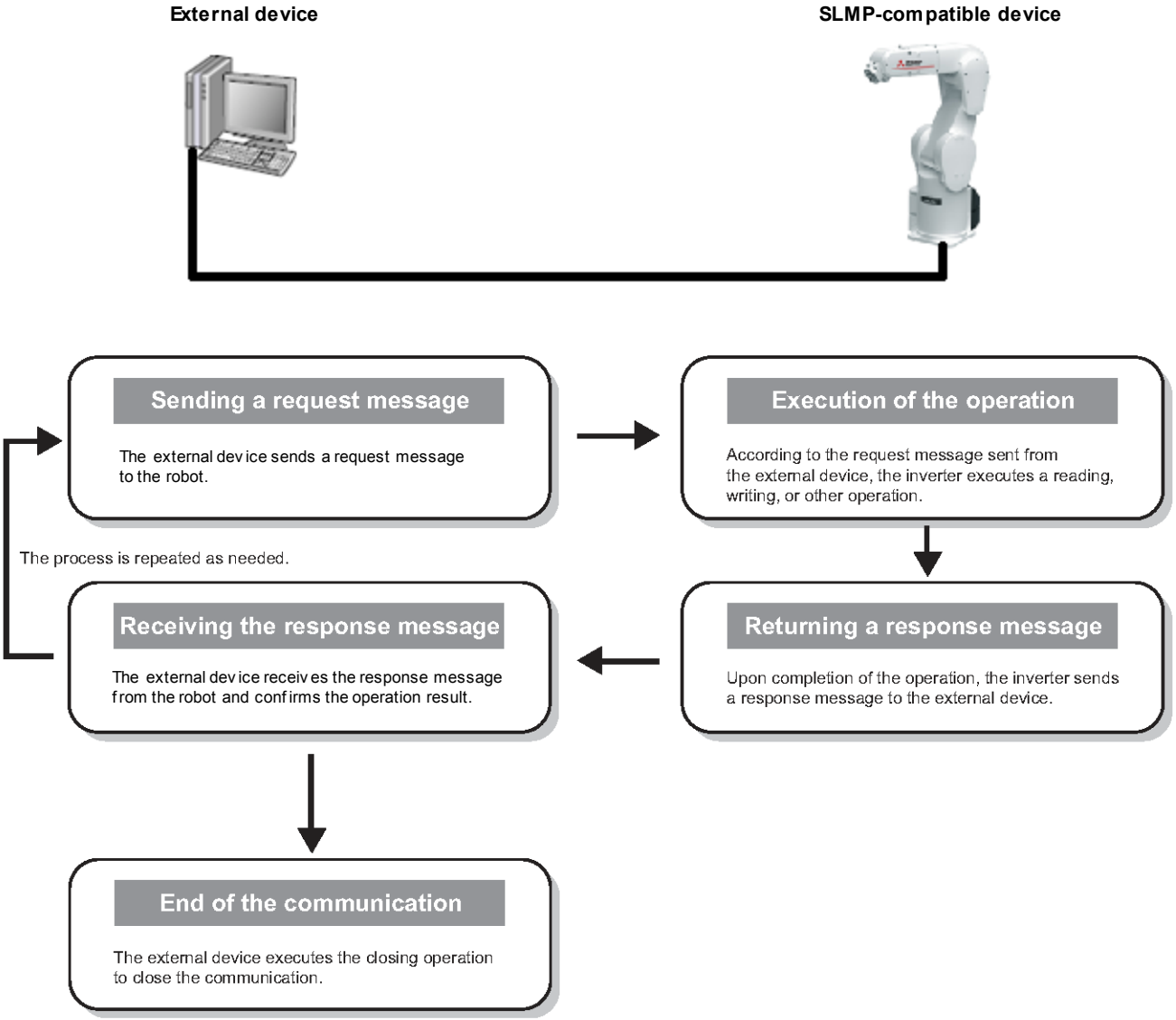
6.3.1. Using TCP/IP

The following is the communication procedure when performing SLMP communication with TCP/IP.
With TCP/IP, connections are established when updating, and whether data is received normally or not is checked to ensure reliability of data. However, the line load is high as compared to UDP/IP.



6.3.2. Using UDP/IP

The following is the communication procedure when performing SLMP communication with UDP/IP.
With UDP/IP, connections are not established when communication is executed, and whether data is received normally or not is not checked. Therefore, the line load is low. However, data is less reliable as compared to TCP/IP.



6.4. Message Format

The following section describes the SLMP message format.

6.4.1. Request Message

The following is the format of a request message sent from an external device to an SLMP-compatible device.

Header	Subheader	Destination network No.	Destination station No.	Destination unit I/O No.	Destination multidrop station No.	Request data length	Monitoring timer	Request data	Footer

■ Header

This is the header for TCP/IP or UDP/IP. The header is added by the external device before transmission. Note that the header is normally added automatically by the external device.

■ Subheader

This will differ depending on whether a serial No. is added.

The serial No. is an arbitrary number for message recognition added at the external device. When a serial No. is added and a request message sent, the same serial No. is added to the response message. Serial Nos. are used when multiple request messages are sent from an external device to the same SLMP-compatible device.

When adding a serial No. to the request message (When the serial No. is 1234H)	When not adding a serial No. to the request message
(Fixed value) (Fixed value) ASCII code: 5 4 0 0 (35H,34H,30H,30H) | 1 2 3 4 (31H,32H,33H,34H) | 0 0 0 0 (30H,30H,30H,30H) Serial No. (Fixed value) (Fixed value) Binary code: (54H,00H) | (34H,12H) | (00H,00H) Serial No.	(Fixed value) ASCII code: 5 0 0 0 (35H,30H,30H,30H) (Fixed value) Binary code: (50H,00H)

- Use and manage serial Nos. at the external device side.
- When transmitting the message in ASCII code, the serial No. is stored in the order from higher-order byte to lower-order byte.
- When transmitting the message in binary code, the serial No. is stored in the order from lower-order byte to higher-order byte.

■ Request destination network No., request destination station No.

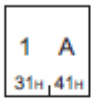
Specify the access destination network No. and station No. Specify the network No. and station No. in hexadecimal. Send the request destination network No. and request destination station No. in the order from higher-order byte to lower-order byte.

- Network No. range
  Host station: 00H
  Other station: 01H to EFH (1 to 239)
- Station No. range
  Host station: FFH (when the network No. is 00H)
  Other station: 01H to 78H (1 to 120)

Example

When 1AH(26) is specified as the request destination network No.

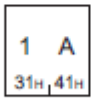
ASCII code



Binary code

When 1AH(26) is specified as the request destination station No.

ASCII code



Binary code

The host station has a network No. of 00H and a station No. of FFH. Other stations have other values. The request data addressed to the own station is received regardless of the network No. and station No. settings.

Furthermore, the request data addressed to the other stations is received when the SLMPNWNO and SLMPNDID settings are the same.

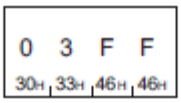
■ Destination unit I/O No.

Specify the access destination unit (fixed to 03FFH).

Example

When 03FFH is specified as the request destination unit I/O No.

ASCII code



Binary code

- When performing data communication in ASCII code
  Send data in the order higher-order byte to lower-order byte.
- When performing data communication in binary code
  Send data in the order lower-order byte to higher-order byte.

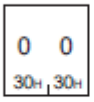
■ Request destination multidrop station No.

Specify the access destination multidrop station (fixed to 00H).

Example

When 0 is specified as the request destination multidrop station No.

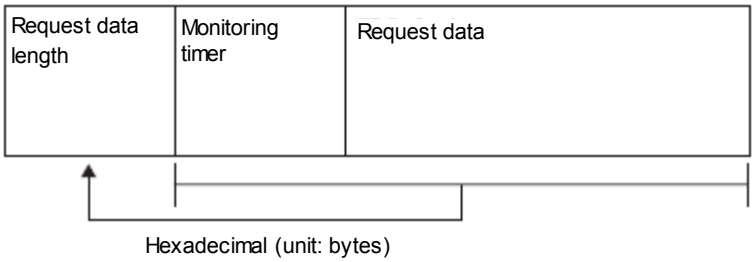
ASCII code



Binary code

■ Request data length

Specify the data length from the monitoring timer to the request data in hexadecimal. (Units: bytes)



Example

When the request data length is 24 bytes

ASCII code

0	0	1	8
30H	30H	31H	38H

Binary code

18H	00H

- When performing data communication in ASCII code
  Send data in the order from higher-order byte to lower-order byte.
- When performing data communication in binary code
  Send data in the order from lower-order byte to higher-order byte.

■ Monitoring timer

Not used (fixed to 0000H)

Example

When 10H specified for monitoring timer

ASCII code

0	0	1	0
30H	30H	31H	30H

Binary code

10H	00H

6.4.2. Response Message Format

The following is the format of a response message sent from an SLMP-compatible device to an external device.

(Normal completion)

Header	Subheader	Destination network No.	Destination station No.	Destination unit I/O No.	Destination multidrop station No.	Response data length	End code	Response data	Footer

(Failed completion)

Header	Subheader	Destination network No.	Destination station No.	Destination unit I/O No.	Destination multidrop station No.	Response data length	End code	Network No. (responding station)	Station No. (responding station)	Destination unit I/O No.	Destination multidrop station No.	Command	Subcommand	Footer

Error information

* The following items contain the same information described in section 6.4.1 of this manual.

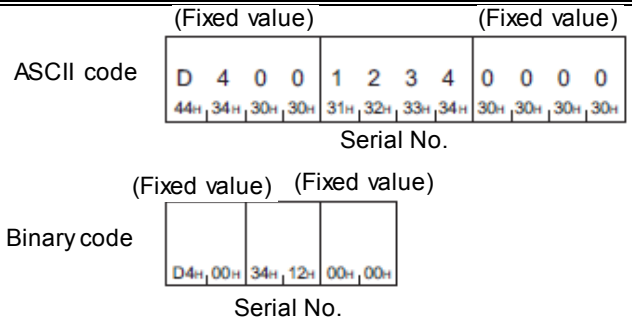
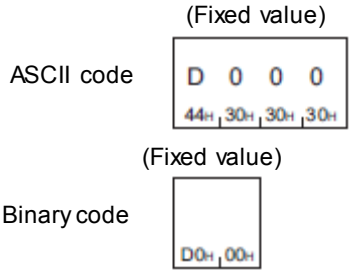
- Request destination network No.
- Request destination station No.
- Request destination unit I/O No.
- Request destination multidrop station No.

■ Header

Contains the Ethernet header.

■ Subheader

Contains the subheader for the request message.

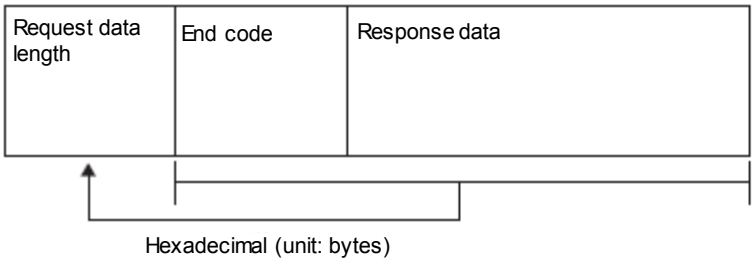
When adding a serial No. to the request message (when the serial No. is 1234H)	When not adding a serial No. to the request message
	

- When performing data communication in ASCII code
  Serial Nos. are stored in the order from higher-order byte to lower-order byte.
- When performing data communication in binary code
  Serial Nos. are stored in the order from lower-order byte to higher-order byte.

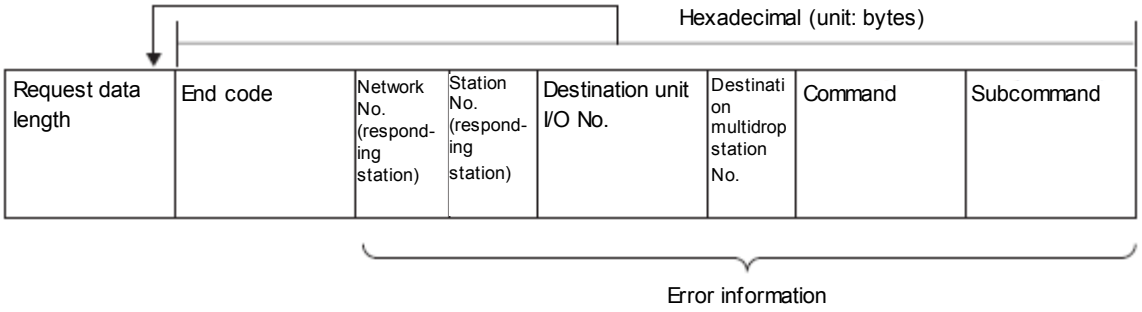
■ Response data length

The data length from the end code to the response data (successful completion) or error information (failed completion) is stored in hexadecimal. (Units: bytes)

(Normal completion)



(Failed completion)



Example

When the response data length is 22 bytes

ASCII code

0	0	1	6
30H	30H	31H	36H

Binary code

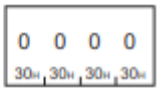
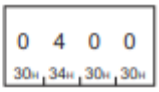
16H, 00H

- When performing data communication in ASCII code
  Send data in the order from higher-order byte to lower-order byte.
- When performing data communication in binary code
  Send data in the order from lower-order byte to higher-order byte.

■ End code

The command processing result is stored.

The value "0" is stored for normal completion. An error code is stored for abnormal completion. (See section 6.6 of this manual.)

Successful completion	Failed completion (0400H)
ASCII code 	ASCII code 
Binary code 	Binary code 

- When performing data communication in ASCII code
  The command processing result is stored in the order from higher-order byte to lower-order byte.
- When performing data communication in binary code
  The command processing result is stored in the order from lower-order byte to higher-order byte.

■ Response data

When the command is completed successfully, data such as the read data corresponding to the command is stored.

Refer to the "Response data" section in the command description for details on response data.

■ Error information

The command and the subcommand, etc. for which an error occurred are stored.

6.5. Commands

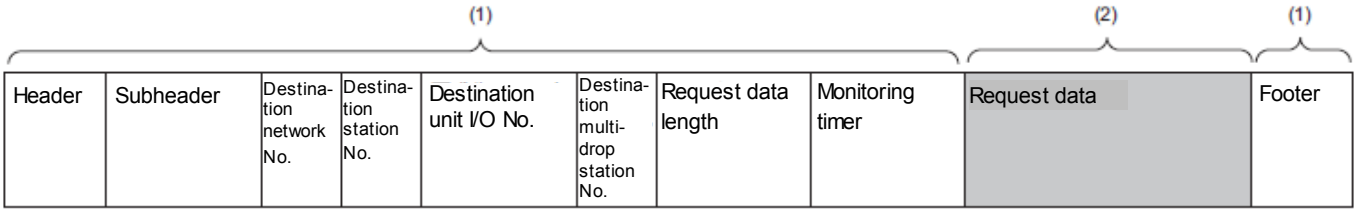
The following section describes SLMP commands.

Refer to section 6.4 of this manual for details on message formats for other than the command sections.

■ Request message format

(1) 6.4.1 Request Message

(2) Request data contains both commands and subcommands. Refer to section 6.5.2 onward in this manual for details.

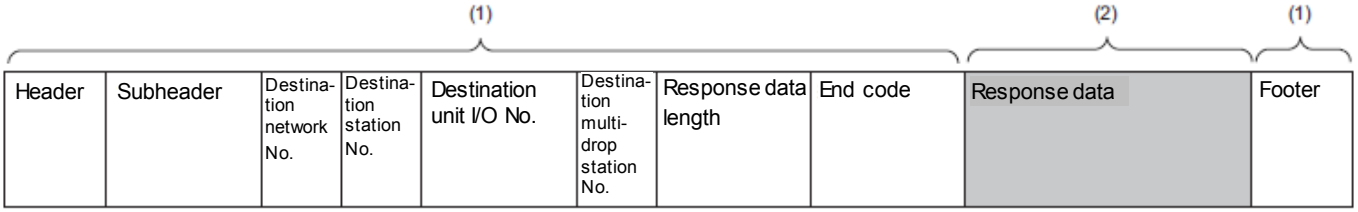


■ Response message format

- Normal completion

(1) 6.4.2 Response Message Format

(2) Refer to section 6.5.2 onward in this manual.



- Failed completion

  Refer to section 6.4.2 onward in this manual.

6.5.1. List of Commands

The following is a list of commands. The following "Subcommands" will differ depending on the device specified. Refer to section 6.5.2 onward in this manual.

Item		Command	Subcommand	Description	Reference
Category	Operation				
Device	Read	0401	00□1 00□3	The value is read in bit devices (with consecutive device numbers) in 1-bit units.	6.5.2.2 Read
			00□0 00□2	● The value is read in bit devices (with consecutive device numbers) in 16-bit units. ● The value is read in word devices (with consecutive device numbers) in 1-word units.	
	Write	1401	00□1 00□3	The value is written to bit devices (with consecutive device numbers) in 1-bit units.	6.5.2.3 Write
			00□0 00□2	● The value is written to bit devices (with consecutive device numbers) in 16-bit units. ● The value is written to word devices (with consecutive device numbers) in 1-word units.	
	Read Random	0403	00□0 00□2	The value is read in the devices with the specified numbers. The devices with non-consecutive numbers can be specified. The value is read from the word devices in 1-word or 2-word units.	6.5.2.4 Read Random
	Write Random	1402	00□1 00□3	The value is written to the bit devices with the specified device numbers (each set of 1 bits has a device number). The devices with non-consecutive numbers can be specified.	6.5.2.5 Write Random
			00□0 00□2	● The value is written to the bit devices with the specified device numbers (each set of 16 bits has a device number). The devices with non-consecutive numbers can be specified. ● The value is written to the word devices with the specified device numbers (each word or each set of two words has a device number). The devices with non-consecutive numbers can be specified.	
Self Test		0619	0000	Performs a test to determine whether communication with external devices is normal.	6.5.3 Self Test (Loopback Test)

6.5.2. Device (Device Access)

The following section describes commands used to perform device reading and writing.

6.5.2.1. Data Used in Commands

- Device code

  Access destination devices are specified in request data with the following device codes.

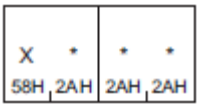
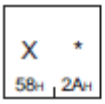
  For subcommands 0001 and 0000, specify the device code enclosed in parentheses ( ).

Device	Category	Device code		Device No. range		Remarks
		ASCII code *1	Binary code			
Special relay (SM)	Bit	SM** (SM)	0091H (91H)	SM0 to SM4095	Decimal	—
Special register (SD)	Word	SD** (SD)	00A9H (A9H)	SD0 to SD4095	Decimal	—
Input (X)	Bit	X*** (X*)	009CH (9CH)	R type: X0 to XFFF D type: X0 to X1FFF	Hexadecimal	—
Output (Y)		Y*** (Y*)	009DH (9DH)	R type: Y0 to YFFF D type: Y0 to Y1FFF	Hexadecimal	—
Internal relay (M)		M*** (M*)	0090H (90H)	M0 to M18431	Decimal	Cannot be specified with D type.
Data register (D)	Word	D*** (D*)	00A8H (A8H)	D0 to D5119	Decimal	—
CPU buffer memory access device	See section 6.5.2.6 of this manual.					Cannot be specified with D type.

*1: When performing data communication in ASCII code, specify device codes with 4 digits for subcommands 00□3 and 00□2. For device codes with 3 digits or less, add an asterisk (*) (ASCII code: 2AH) or a space (ASCII code: 20H) after the device code.

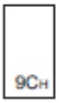
Specify device codes with 2 digits for subcommands 00□1 and 00□0. For device codes with 1 digit, add an asterisk (*) (ASCII code: 2AH) or a space (ASCII code: 20H) after the device code.

- When performing data communication in ASCII code
  Use device codes by converting them to ASCII code (2 or 4 digits), and send them in the order higher-order byte to lower-order byte. Use codes in upper case characters for letters of the alphabet.
  With subcommands 0003, 0002 and 0001, 0000, the number of digits converted to ASCII code will differ.

Subcommand	Number of digits	Example
0003 0002	Conversion to 4 ASCII code digits	If input (X) (4 digits) *1 
0001 0000	Conversion to 2 ASCII code digits	If input (X) (2 digits) *1 

*1: Send input relay device codes in order from "X". Note that asterisks (*) from the second character onward may also be specified with a space (code: 20H).

- When performing data communication in binary code
  Use numerical values (1 or 2 bytes), and send data in the order lower-order byte to higher-order byte.
  With subcommands 0003, 0002 and 0001, 0000, the data size of the numerical values will differ.

Subcommand	Number of digits	Example
0003 0002	2 bytes	If input (X) (2 bytes) 
0001 0000	1 byte	If input (X) (1 byte) 

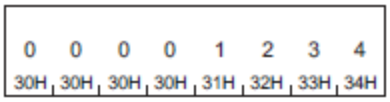
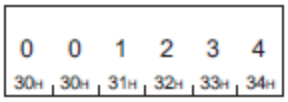
- First device No. (device No.)

  Specify the device No. for reading/writing data. When consecutive devices are specified, specify the first device No.
  Specify the first device No. in decimal or hexadecimal depending on the device type.

  - When performing data communication in ASCII code

    Use device Nos. by converting them to ASCII code (6 or 8 digits), and send them in the order higher-order byte to lower-order byte.

    With subcommands 0003, 0002 and 0001, 0000, the number of digits converted to ASCII code will differ.

Subcommand	Number of digits	Example
0003 0002	Conversion to 8 ASCII code digits	If device No. is 1234 (8 digits) *1 
0001 0000	Conversion to 6 ASCII code digits	If device No. is 1234 (6 digits) *1 

*1: Send in order from 0. The higher-order digit 0 can also be specified with a space (code: 20H).

  - When performing data communication in binary code

    Use numerical values (3 or 4 bytes), and send data in the order from lower-order byte to higher-order byte. When the device No. is a decimal device, convert it to a hexadecimal value and send.

    With subcommands 0003, 0002 and 0001, 0000, the data size of the numerical values will differ.

Subcommand	Number of digits	Example
0003 0002	4 bytes	If internal relay M1234, link relay B1234 (4 bytes) *1 
0001 0000	3 bytes	If internal relay M1234, link relay B1234 (3 bytes) *2 

*1: The device number for internal relay M1234 is in decimal, and must therefore be converted to hexadecimal. This will be 000004D2H, and should be sent in the order D2H, 04H, 00H, 00H. This will be 00001234H for link relay B1234, and should be sent in the order 34H, 12H, 00H, 00H.

*2: The device number for internal relay M1234 is in decimal, and must therefore be converted to hexadecimal. This will be 0004D2H, and should be sent in the order D2H, 04H, 00H. This will be 001234H for link relay B1234, and should be sent in the order 34H, 12H, 00H.

- Number of devices

  Specify the number of devices for reading/writing data.

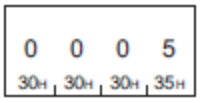
  - When performing data communication in ASCII code

    Use the number of devices by converting them to 4 ASCII code digits (hexadecimal), and send them in the order from higher-order byte to lower-order byte. Use codes in upper case characters when specifying letters of the alphabet.

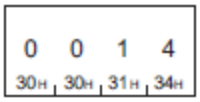
    Example

    Number of devices: 5 / 20

    5 devices

    20 devices

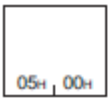
  - When performing data communication in binary code

    Use a 2-byte numerical value to indicate the number of processing devices, and send in the order from lower-order byte to higher-order byte.

    Example

    Number of devices: 5 / 20

    5 devices

    20 devices

- Read data / write data

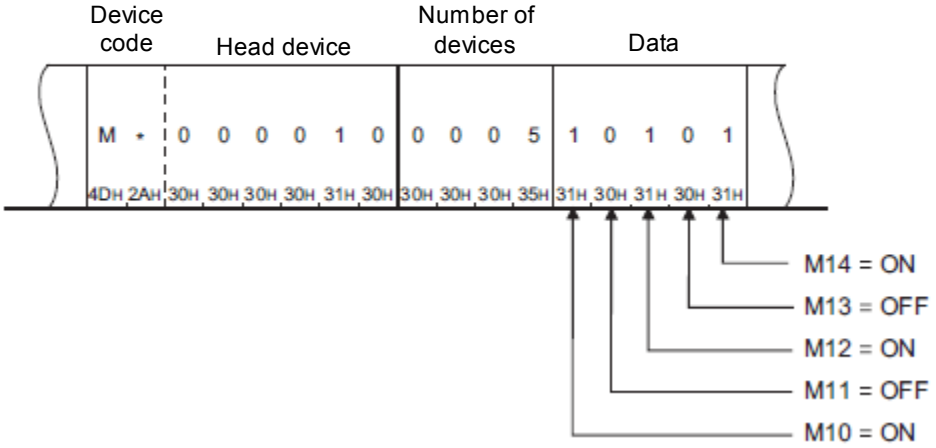
  The value read from the device is stored for reading. The value to be written to the device is stored for writing.
  The data arrangement will differ depending on the bit unit (subcommand: 00□1, 00□3) or word unit (subcommand: 00□0, 00□2).

  - In bit units (subcommand: 00□1, 00□3)

    When performing data communication in ASCII code, send data for the number of specified devices from the specified start device in the order from higher-order byte to lower-order byte. The ON state is denoted as "31H" (1), and the OFF state is denoted as "30H" (0). Use codes in upper case characters when specifying letters of the alphabet.

    Example

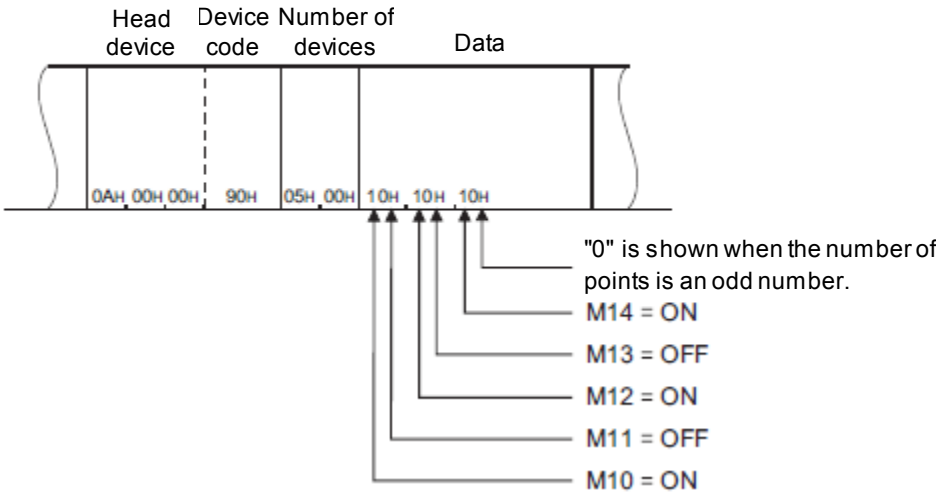
    ON/OFF state of five devices starting from M10



    When performing data communication in binary code, specify a single device with 4 bits, and send data for the number of specified devices from the specified start device in the order from higher-order byte to lower-order byte. The ON state is denoted as "1", and the OFF state is denoted as "0".

    Example

    ON/OFF state of five devices starting from M10



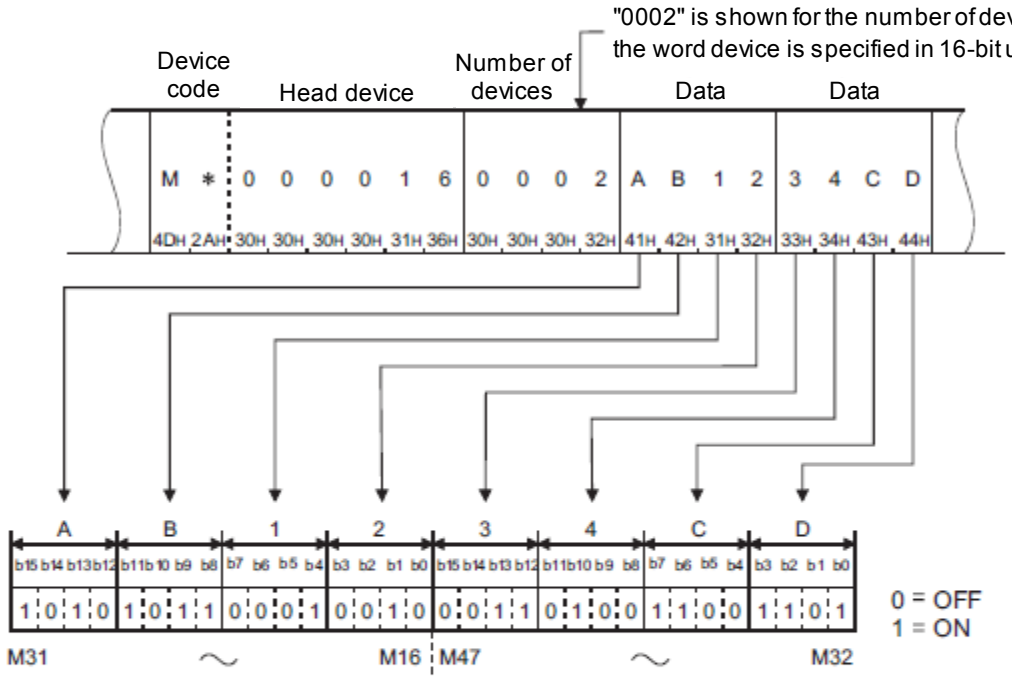
- In word units (subcommand: 00□0, 00□2)

  When performing data communication in ASCII code, send data with 1 word in 4-bit units in the order from higher-order byte to lower-order byte. Data is expressed in hexadecimal.

  Use codes in upper case characters when specifying letters of the alphabet.

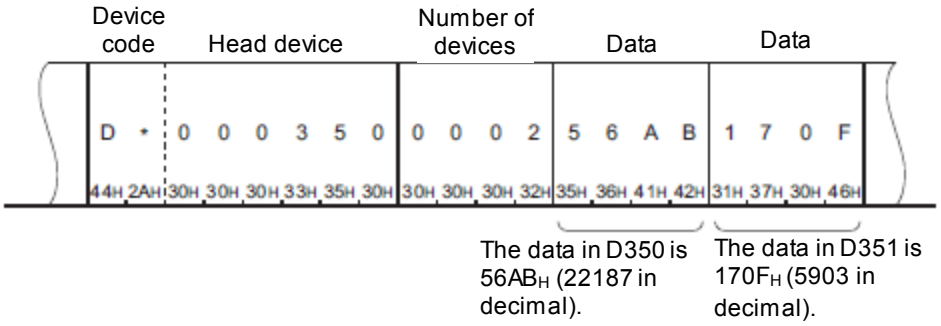
  Example

  ON/OFF state of 32 devices starting from M16



  Example

  Data stored in D350/D351



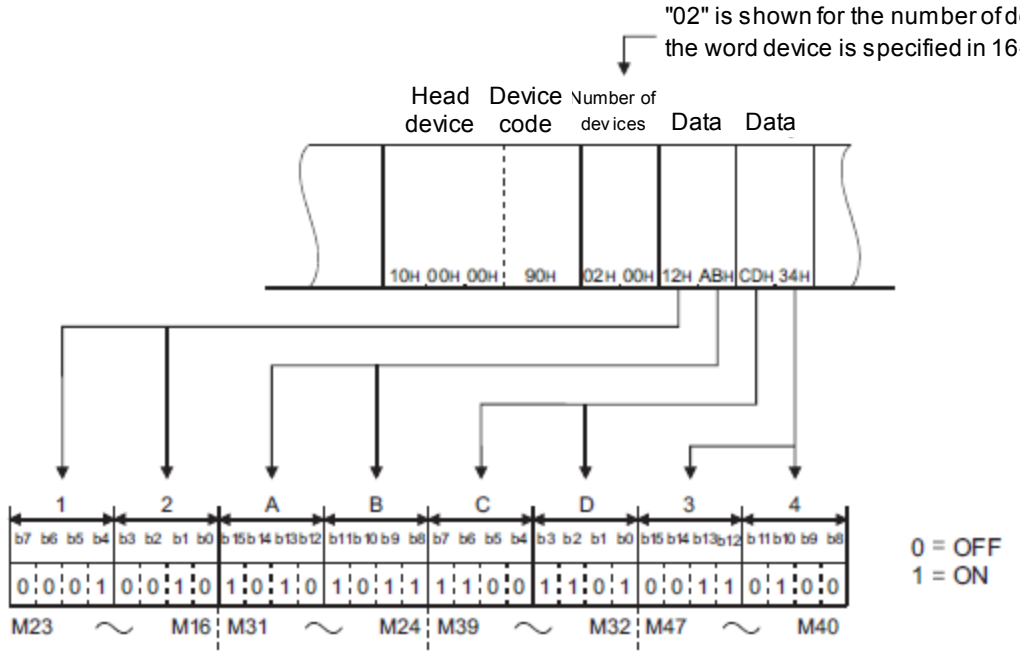
When word devices for reading data contain other than integers (real numbers, character strings), stored values are read as integer values.

- When D0 to D1 contains a real number (0.75), D0 = 0000H, and D1 = 3F40H.
- When D2 to D3 contains a character string ("12AB"), D2 = 3231H, and D3 = 4241H.

When bit devices are handled in word units when performing data communication in binary code, a single device is specified with a 1 bit as shown in the following example. Data is stored in the order from lower-order byte (bit 0 to bit 7) to higher-order byte (bit 8 to bit 15).

Example

ON/OFF state of 32 devices starting from M16



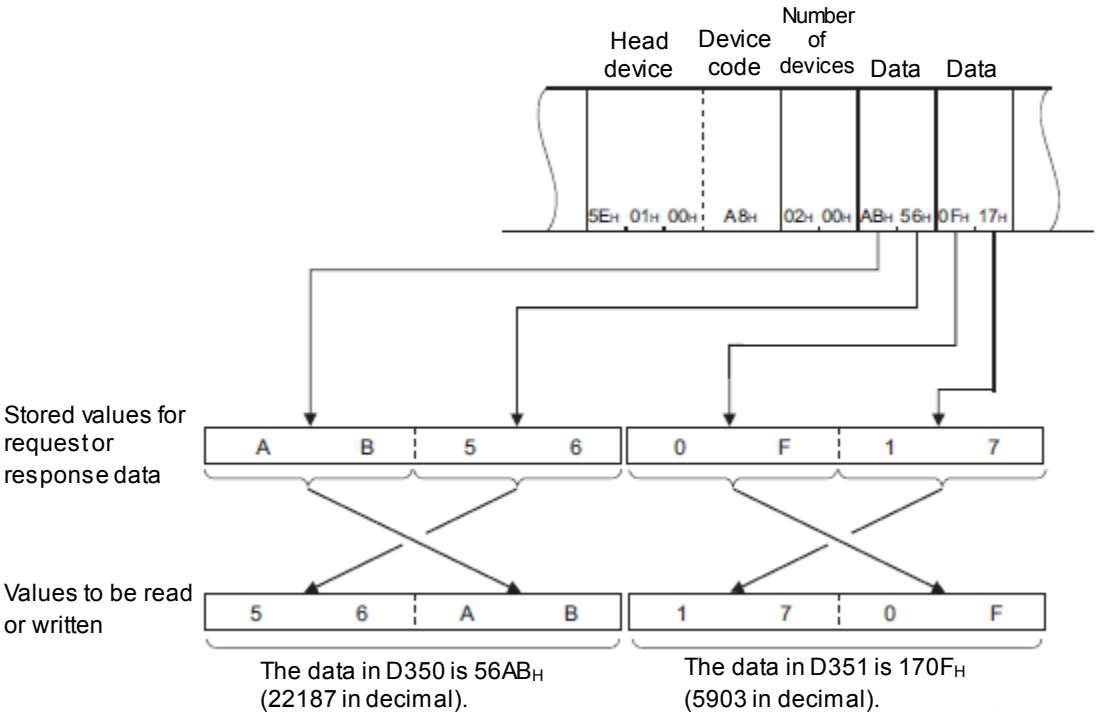
When word devices are used, 1 word is specified in 16 bits as follows. Data is stored in the order from lower-order byte (bit 0 to bit 7) to higher-order byte (bit 8 to bit 15).

When reading, do so after replacing values stored in the response data with higher/lower-order bytes at the user side.

When writing, store request data after replacing the values to be written with higher/lower-order bytes at the user side.

Example

Data stored in D350/D351



When word devices for reading data contain other than integers (real numbers, character strings), stored values are read as integer values.

- When D0 to D1 contains a real number (0.75), D0 = 0000H, and D1 = 3F40H.
- When D2 to D3 contains a character string ("12AB"), D2 = 3231H, and D3 = 4241H.

- Precautions

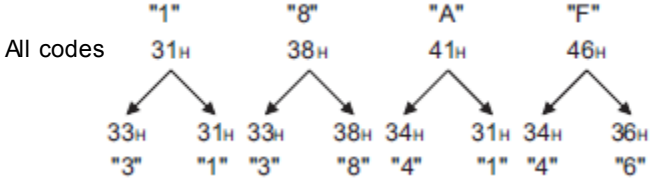
  When performing data communication with ASCII data, process as follows when passing character strings from external devices to the CPU module.

  The procedure for converting data received by SLMP-compatible devices from external devices to binary code data, and writing it to a specified device is described below.

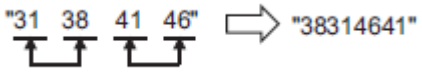
  1. Expand character strings sent from external devices to 2-byte code per one character.
  2. Rearrange the character strings expanded to 2-byte code per one character, and send them to the SLMP-compatible device.
  3. Write the data sent to the SLMP-compatible device to the specified device.

  The following is an example of a case where a character string ("18AF") received from an external device is converted to binary code data, and written to D0 to D1.

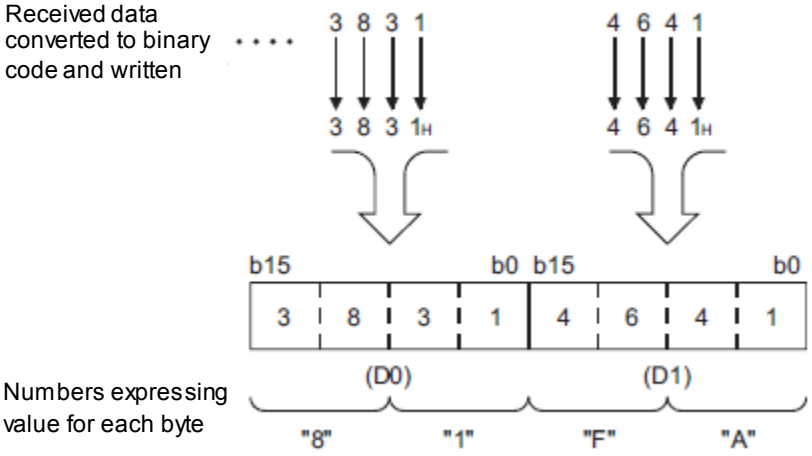
  1. Expand the character string ("18AF") sent from the external device to 2-byte code per one character.



  2. Rearrange the character strings expanded to 2-byte code per one character, and send them to the SLMP-compatible device.



  3. Write the "38314641" data sent to the SLMP-compatible device to D0 to D1.



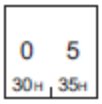
- Number of devices for bit access
  This is the data required to specify the number of devices accessed in bit units.

  - When performing data communication in ASCII code
    Convert the number of devices to 2 ASCII code digits (hexadecimal), and send them in the order from higher-order byte to lower-order byte. Use codes in upper case characters if specifying letters of the alphabet.

    Example

    Number of devices: 5 / 20

    5 devices

    20 devices

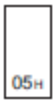
    

  - When performing data communication in binary code
    Convert the number of devices to hexadecimal and send.

    Example

    Number of devices: 5 / 20

    5 devices

    20 devices

6.5.2.2. Read (Command: 0401)

Read the data from devices.

■ Request data

ASCII

0 4 0 1 (30H 34H 30H 31H)	Subcommand	Device code	First device No.	Number of devices

Binary

	Sub-command	First device No.	Device code	Number of devices
01H 04H				

- Subcommand

Item	Subcommand *1	
	ASCII code	Binary code
When reading data in bit units	0 0 0 1 (30H 30H 30H 31H) or 0 0 8 1 (30H 30H 38H 31H)	01H 00H or 81H 00H
	0 0 0 3 (30H 30H 30H 33H) or 0 0 8 3 (30H 30H 38H 33H)	03H 00H or 83H 00H
When reading data in word units	0 0 0 0 (30H 30H 30H 30H) or 0 0 8 0 (30H 30H 38H 30H)	00H 00H or 80H 00H
	0 0 0 2 (30H 30H 30H 32H) or 0 0 8 2 (30H 30H 38H 32H)	02H 00H or 82H 00H

*1: Use subcommand 008□ if accessing a link direct device, module access device, or CPU buffer memory access device. When the subcommand is set to 008□, the message format will differ. (Reading, writing by specifying device extension)

- Device code

  Specify the type of device to be read.

- First device No.

  Specify the first number of the device to be read.

- Number of devices
  Specify the number of devices to be read.

Item	Number of devices	
	ASCII code	Binary code
When reading data in bit units	1 to 3584 devices	1 to 7168 devices
When reading data in word units	1 to 960 devices	

■ Response data

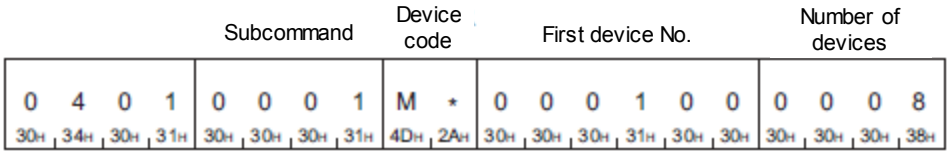
The value read from the device is stored in hexadecimal. The data order will differ depending on whether it is in ASCII code or binary code.

■ Communication example (when reading data in bit units)

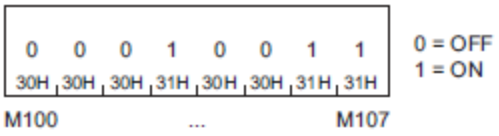
Read M100 to M107.

- When performing data communication in ASCII code

(Request data)

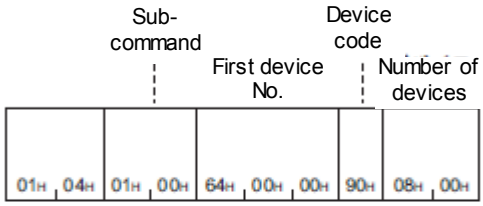


(Response data)

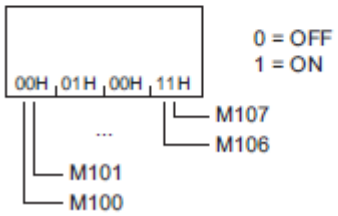


- When performing data communication in binary code

(Request data)



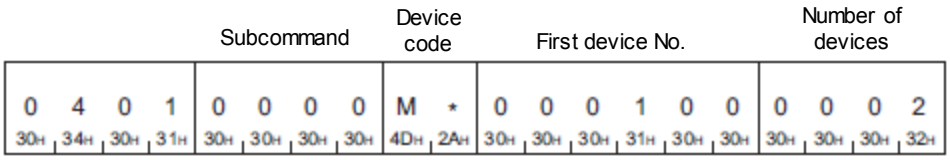
(Response data)



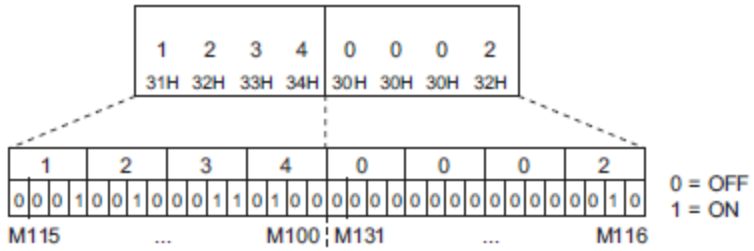
■ Communication example (when reading data in word units (bit device))

- Read M100 to M131 (data for two words).

(Request data)

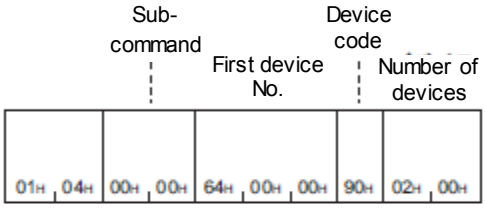


(Response data)

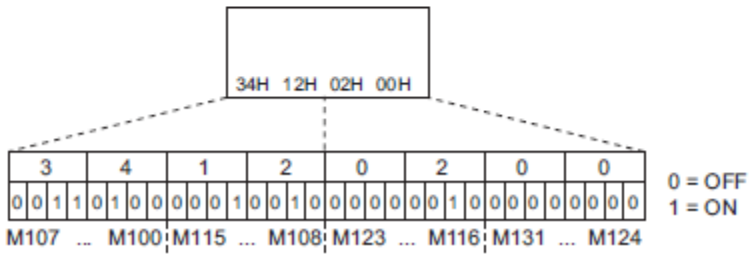


- When performing data communication in binary code

(Request data)



(Response data)



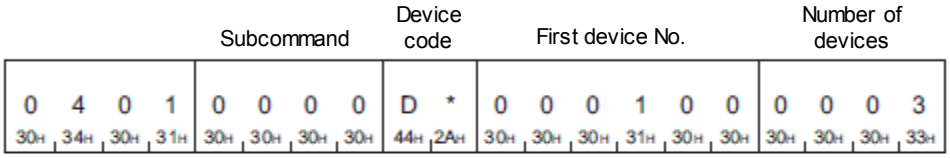
■ Communication example (when reading data in word units (word device))

Read values D100 to D102.

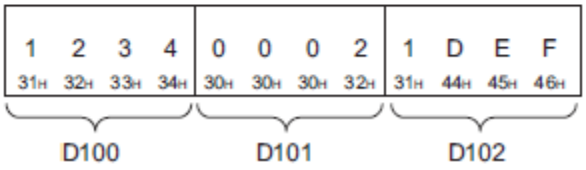
Here D100 = 4660 (1234H), D101 = 2 (2H), and D102 = 7663 (1DEFH).

- When performing data communication in ASCII code

(Request data)

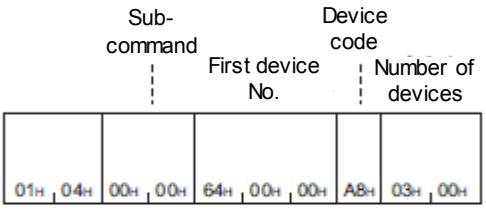


(Response data)

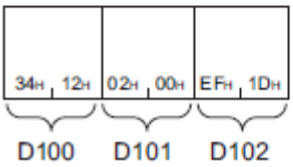


- When performing data communication in binary code

(Request data)



(Response data)



6.5.2.3. Write (Command: 1401)

Write the data to devices.

■ Request data

ASCII

1 4 0 1 (31H 34H 30H 31H)	Subcommand	Device code	First device No.	Number of devices	Write data

Binary

01H 14H	Sub-command	First device No.	Device code	Number of devices	Write data

- Subcommand

Item	Subcommand *1	
	ASCII code	Binary code
When writing data in bit units	0 0 0 1 (30H 30H 30H 31H) or 0 0 8 1 (30H 30H 38H 31H)	(01H 00H) or (81H 00H)
	0 0 0 3 (30H 30H 30H 33H) or 0 0 8 3 (30H 30H 38H 33H)	(03H 00H) or (83H 00H)
When writing values in word units	0 0 0 0 (30H 30H 30H 30H) or 0 0 8 0 (30H 30H 38H 30H)	(00H 00H) or (80H 00H)
	0 0 0 2 (30H 30H 30H 32H) or 0 0 8 2 (30H 30H 38H 32H)	(02H 00H) or (82H 00H)

*1: Use subcommand 008□ if accessing a link direct device, module access device, or CPU buffer memory access device. When the subcommand is set to 008□, the message format will differ. (Reading, writing by specifying device extension)

- Device code

  Specify the type of device to be written.

- First device No.

  Specify the first number of the device to be written.

- Number of devices

  Specify the number of devices to be written.

Item	Number of devices	
	ASCII code	Binary code
When writing data in bit units	1 to 3584 devices	1 to 7168 devices
When writing data in word units	1 to 960 devices	

- Write data

  Specify the value to be written to all the devices specified in "Number of devices" in the request data.

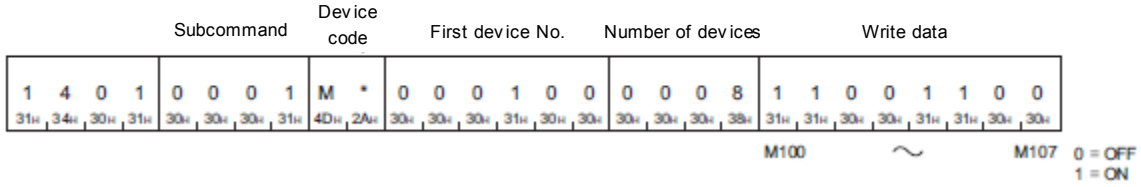
■ Request data

There is no Write command response data.

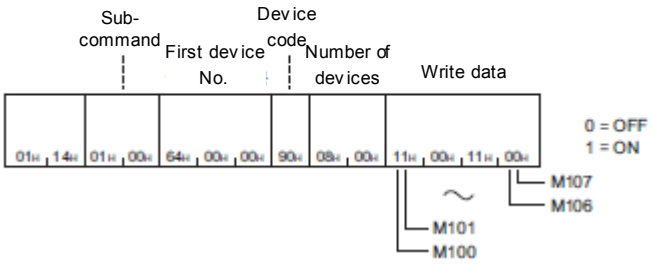
■ Communication example (when writing data in bit units)

Write values to M100 to M107.

- When performing data communication in ASCII code
  (Request data)



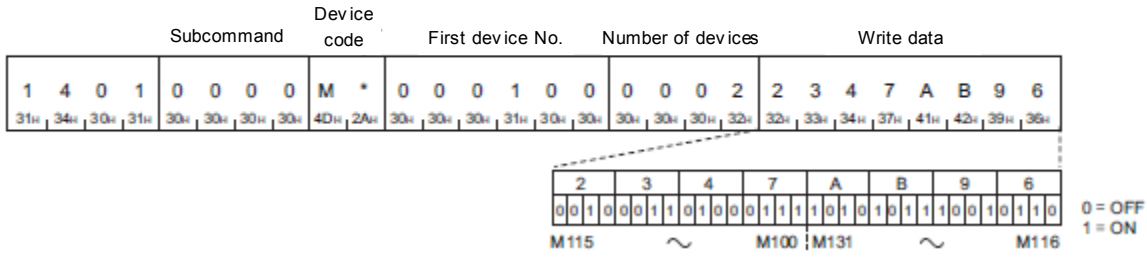
- When performing data communication in binary code
  (Request data)



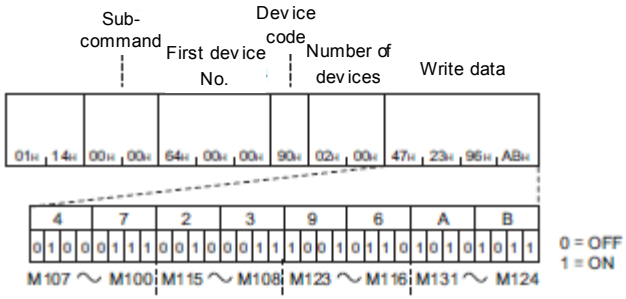
■ Communication example (when writing data in word units (bit device))

Write values to M100 to M131 (data for two words).

- When performing data communication in ASCII code
  (Request data)



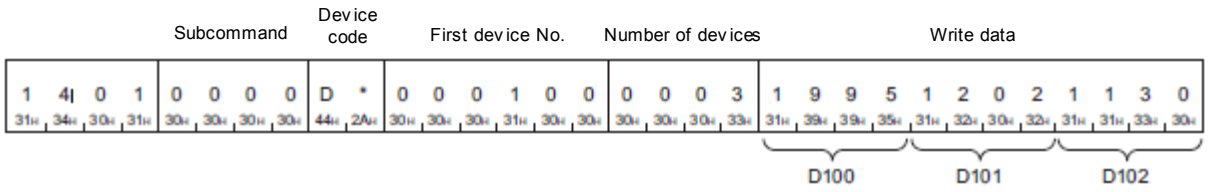
- When performing data communication in binary code
  (Request data)



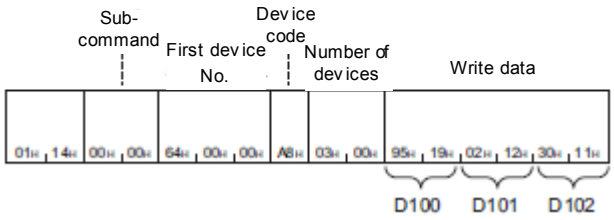
■ Communication example (when writing data in word units (word device))

Write 6549 (1995H) for D100, 4610 (1202H) for D101, and 4400 (1130H) for D102.

- When performing data communication in ASCII code
  (Request data)



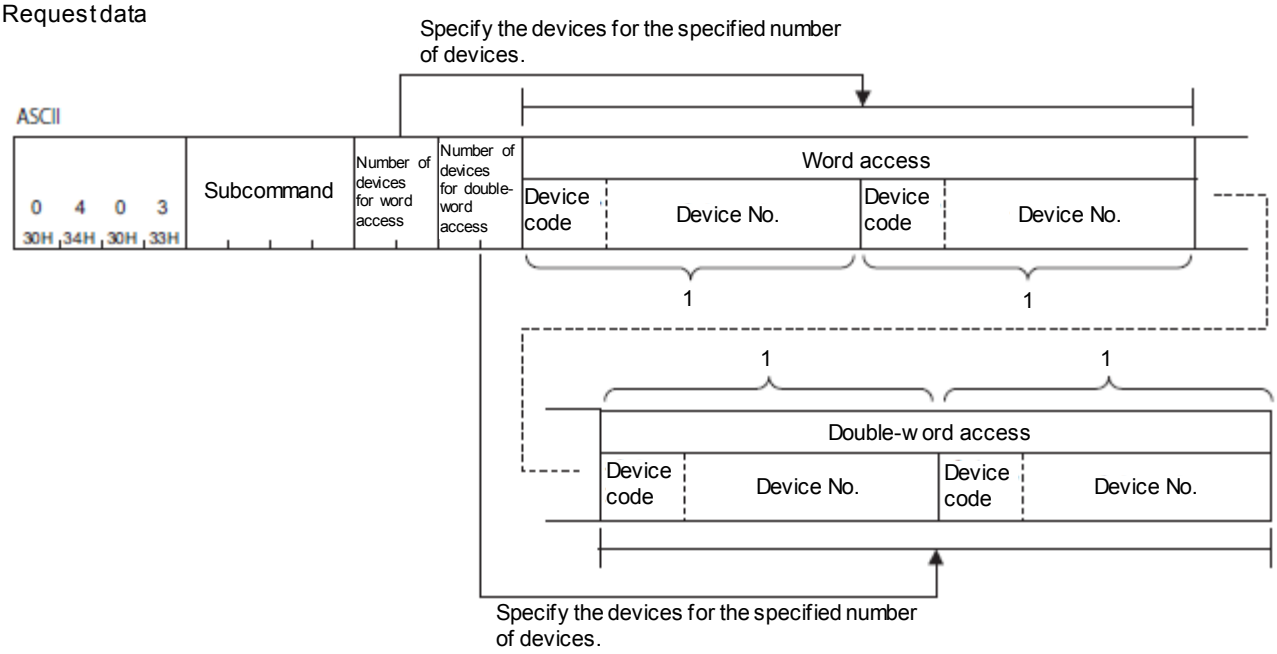
- When performing data communication in binary code
  (Request data)

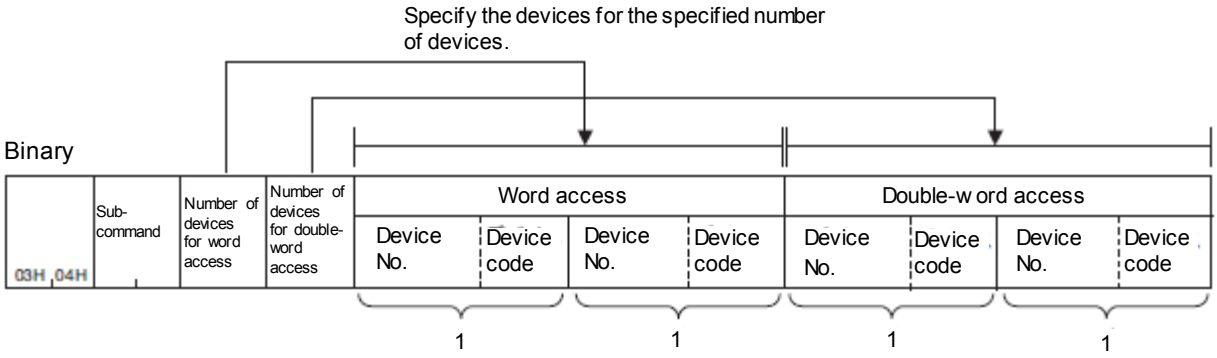


6.5.2.4. Read Random (Command: 0403)

The value is read in the devices with the specified numbers. The devices with non-consecutive numbers can be specified.

■ Request data





- Subcommand

Subcommand *1					
ASCII code			Binary code		
0 0 0 0 30H 30H 30H 30H	or	0 0 8 0 30H 30H 38H 30H	00H 00H	or	80H 00H
0 0 0 2 30H 30H 30H 32H	or	0 0 8 2 30H 30H 38H 32H	02H 00H	or	82H 00H

*1: Use subcommand 008□ if accessing a link direct device, module access device, or CPU buffer memory access device. When the subcommand is set to 008□, the message format will differ. (Reading, writing by specifying device extension)

- Number of word access devices, number of double-word access devices

  Specify the number of devices to be read with 1 byte (binary code) or 2 bytes (2 digits) (ASCII code).

Subcommand	Item	Description
0002	Number of devices for word access	Specify the number of devices for 1 word access. The applicable units are 16-bit units for bit devices, and 1-word units for word devices.
	Number of devices for double-word access	Specify the number of devices for 2 word access. The applicable units are 32-bit units for bit devices, and 2-word units for word devices.
0000	Number of devices for word access	Specify the number of devices for 1 word access. The applicable units are 16-bit units for bit devices, and 1-word units for word devices.
	Number of devices for double-word access	Specify the number of devices for 2 word access. The applicable units are 32-bit units for bit devices, and 2-word units for word devices.

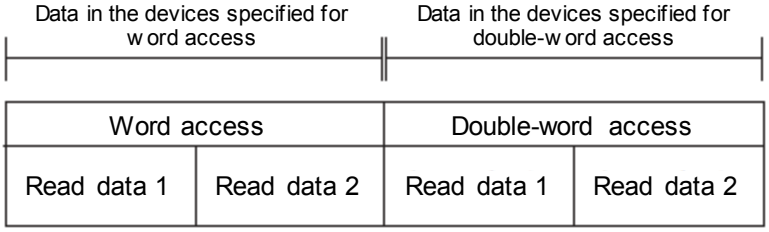
- Device code, device number

  Specify devices to be read in the order word access, double-word access.

Item	Description
Word access	Specify devices based on the number set in the request data for word access. It is not necessary to specify devices when "0" is set.
Double-word access	Specify devices based on the number set in the request data for double-word access. It is not necessary to specify devices when "0" is set.

■ Response data

Values for read devices are stored in hexadecimal. The data order will differ depending on whether it is in ASCII code or binary code.



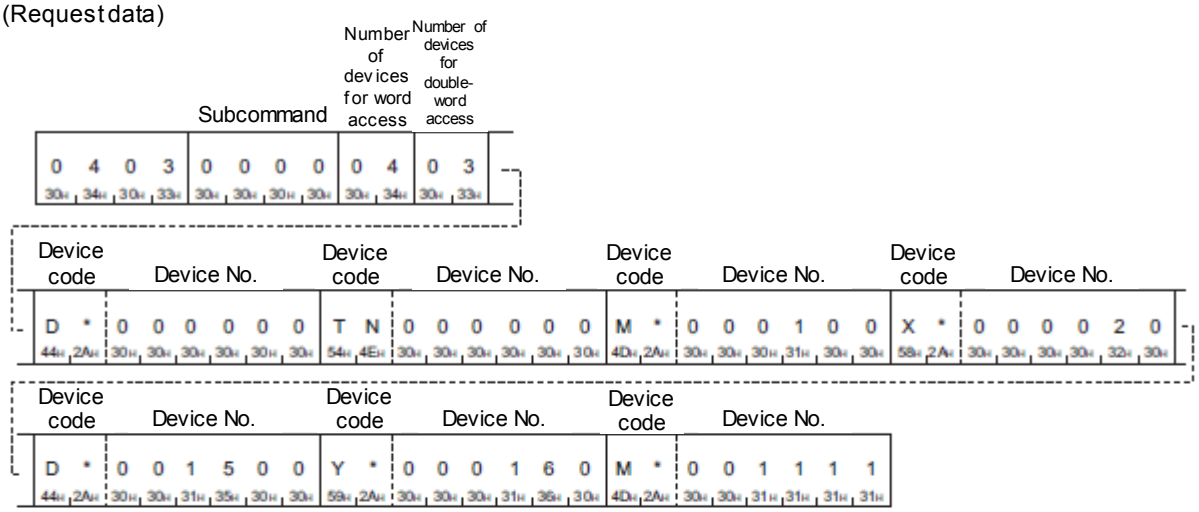
■ Communication example

With word access, read D0, D1, M100 to M115, and X20 to X2F, and with double-word access, read D1500 to D1501, Y160 to Y17F, and M1111 to M1142.

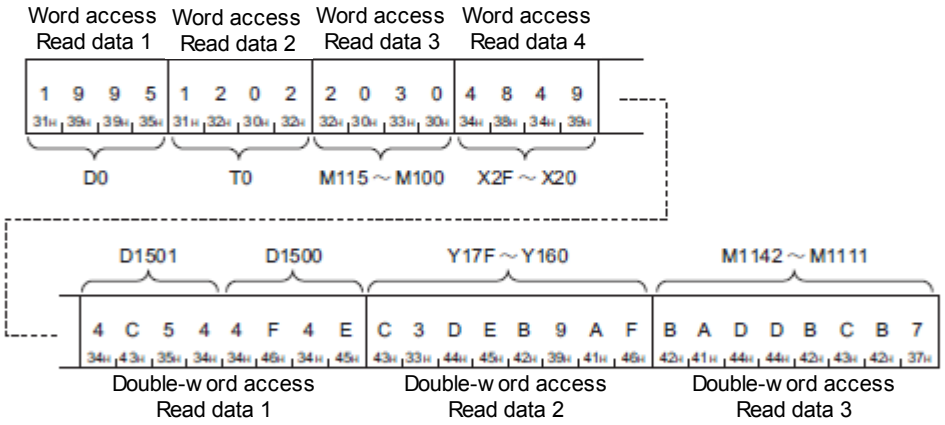
Here D0 = 6549 (1995H), D1 = 4610 (1202H), D1500 = 20302 (4F4EH), and D1501 = 19540 (4C54H).

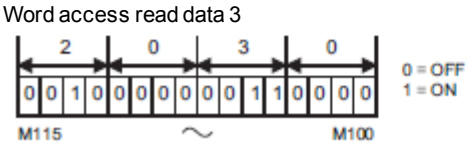
- When performing data communication in ASCII code

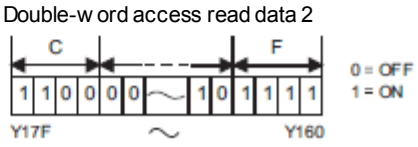
(Request data)

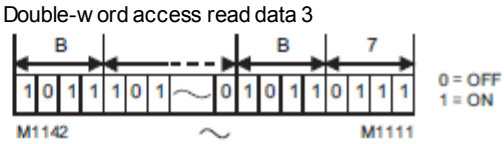


(Response data)



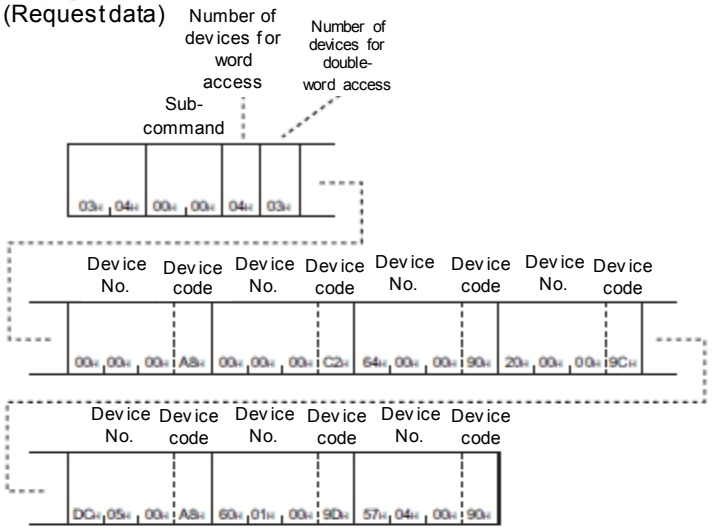




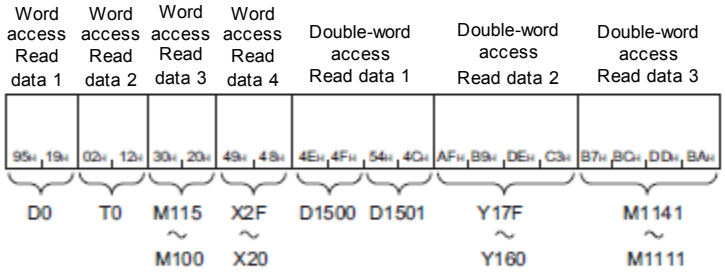


- When performing data communication in binary code

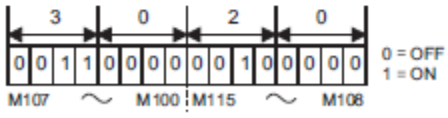
(Request data)



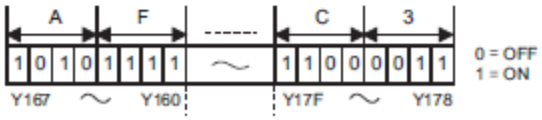
(Response data)



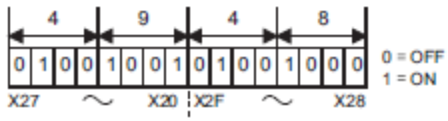
Word access read data 3



Double-word access read data 2



Word access read data 4

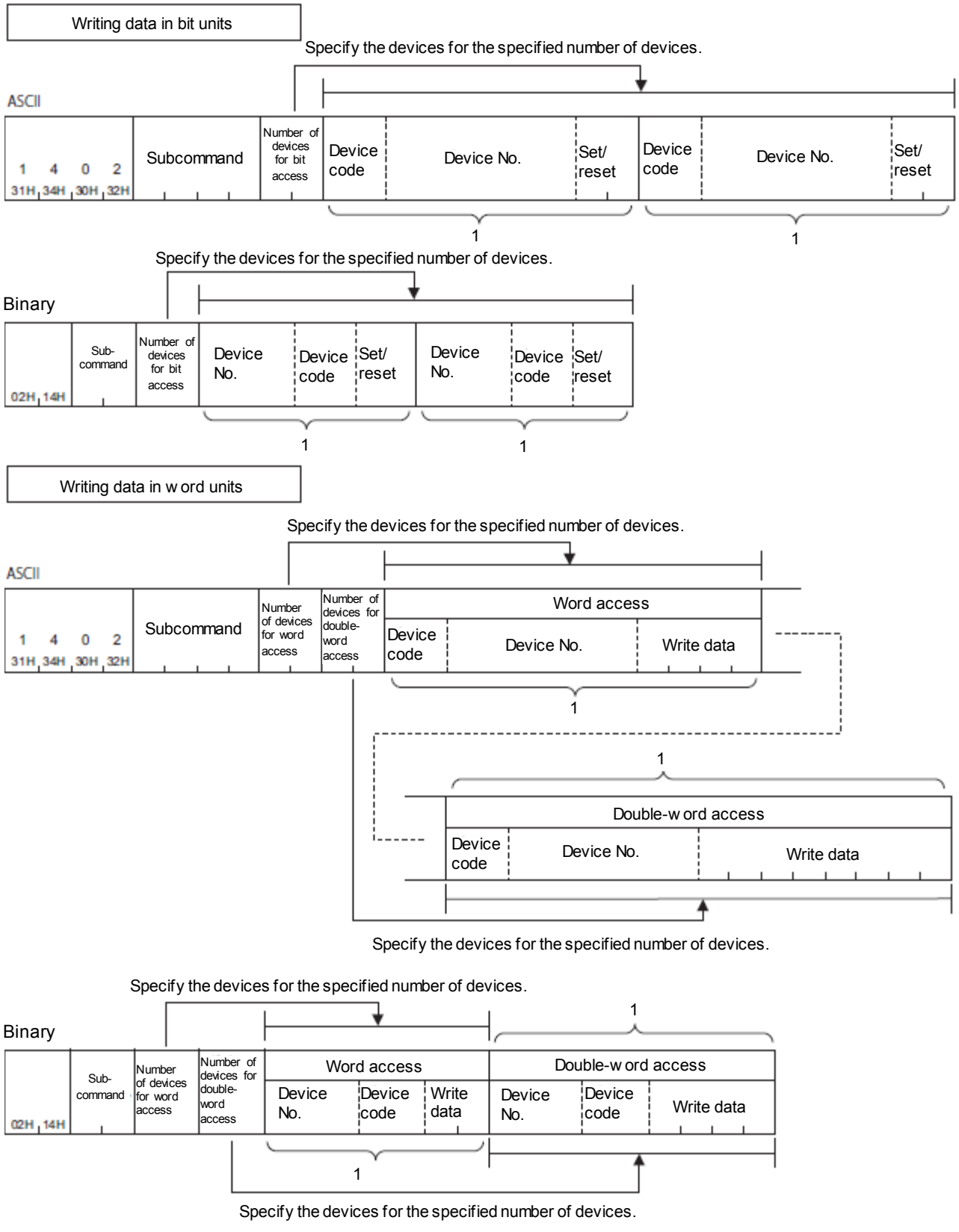


Double-word access read data 3

6.5.2.5. Write Random (Command: 1402)

The value is written in the devices with the specified numbers. The devices with non-consecutive numbers can be specified.

■ Request data



- Subcommand

Item	Subcommand *1	
	ASCII code	Binary code
When writing data in bit units	0 0 0 1 (30H 30H 30H 31H) or 0 0 8 1 (30H 30H 38H 31H)	(01H 00H) or (81H 00H)
	0 0 0 3 (30H 30H 30H 33H) or 0 0 8 3 (30H 30H 38H 33H)	(03H 00H) or (83H 00H)
When writing data in word units	0 0 0 0 (30H 30H 30H 30H) or 0 0 8 0 (30H 30H 38H 30H)	(00H 00H) or (80H 00H)
	0 0 0 2 (30H 30H 30H 32H) or 0 0 8 2 (30H 30H 38H 32H)	(02H 00H) or (82H 00H)

*1: Use subcommand 008□ if accessing a link direct device, module access device, or CPU buffer memory access device. When the subcommand is set to 008□, the message format will differ. (Reading, writing by specifying device extension)

- Number of devices for bit access, Number of devices for word access, Number of devices for double-word access
  Specify the number of target devices.

Subcommand	Item	Description
0003 0002	Number of devices for bit access	Specify the number of bit devices in 1-bit units.
	Number of devices for word access	Specify the number of devices for 1 word access. The applicable units are 16-bit units for bit devices, and 1-word units for word devices.
	Number of devices for double-word access	Specify the number of devices for 2 word access. The applicable units are 32-bit units for bit devices, and 2-word units for word devices.
0001 0000	Number of devices for bit access	Specify the number of bit devices in 1-bit units.
	Number of devices for word access	Specify the number of devices for 1 word access. The applicable units are 16-bit units for bit devices, and 1-word units for word devices.
	Number of devices for double-word access	Specify the number of devices for 2 word access. The applicable units are 32-bit units for bit devices, and 2-word units for word devices.

- Device code, Device No., Write data

  Specify devices to be written.

  When writing data in bit units, specify bit devices.

  Specify write data in hexadecimal.

Item	Description
Word access	Specify devices based on the number set in the request data for word access. It is not necessary to specify devices when "0" is set.
Double-word access	Specify devices based on the number set in the request data for double-word access. It is not necessary to specify devices when "0" is set.

- Set/reset

  Specify ON/OFF for bit devices.

Item	Subcommand	Data to be written		Remarks
		When turned ON	When turned OFF	
ASCII code	0003 0002	"0001"	"0000"	Send 4 digits in order from "0"
	0001 0000	"01"	"00"	Send 2 digits in order from "0"
Binary code	0003 0002	0100H	0000H	Either of the 2-byte numerical values on the left is sent.
	0001 0000	01H	00H	Either of the 1-byte numerical values on the left is sent.

■ Response data

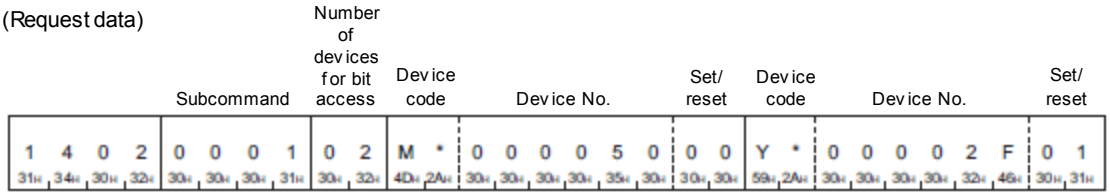
There is no Write Random command response data.

■ Communication example (when writing data in bit units)

Turn M50 OFF, and turn Y2F ON.

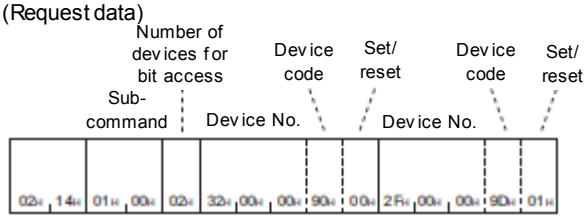
- When performing data communication in ASCII code

(Request data)



- When performing data communication in binary code

(Request data)



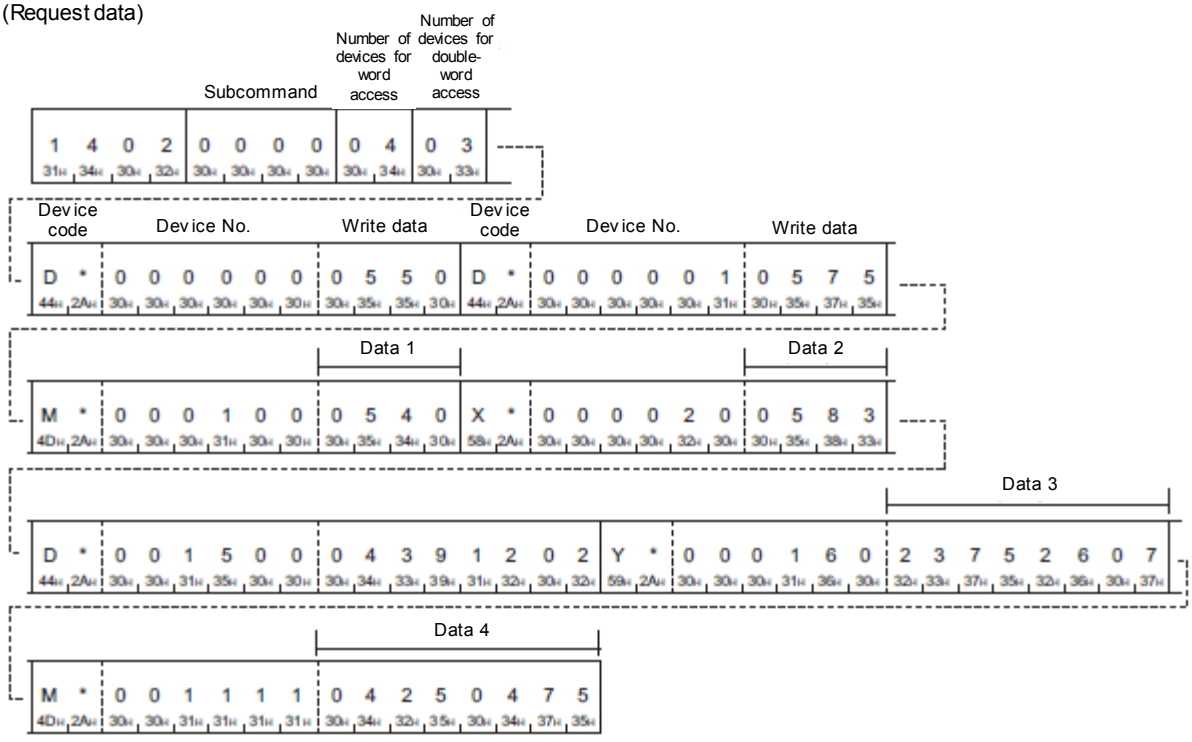
■ Communication example (when writing data in word units)

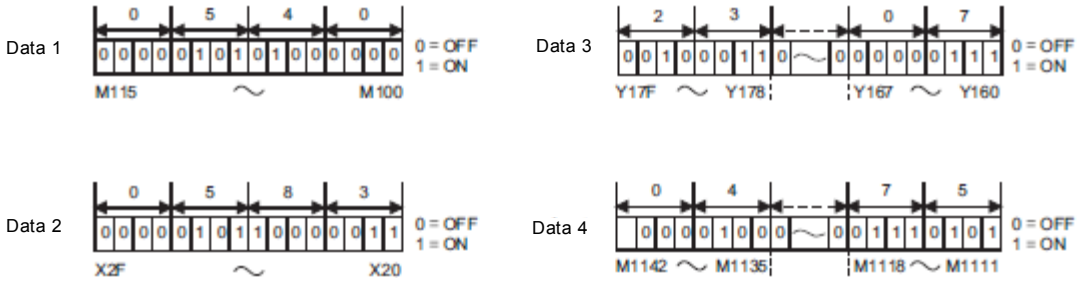
Write values to devices as follows.

Item	Device to be written
Word access	D0, D1, M100 to M115, X20 to X2F
Double-word access	D1500 to D1501, Y160 to Y17F, M1111 to M1142

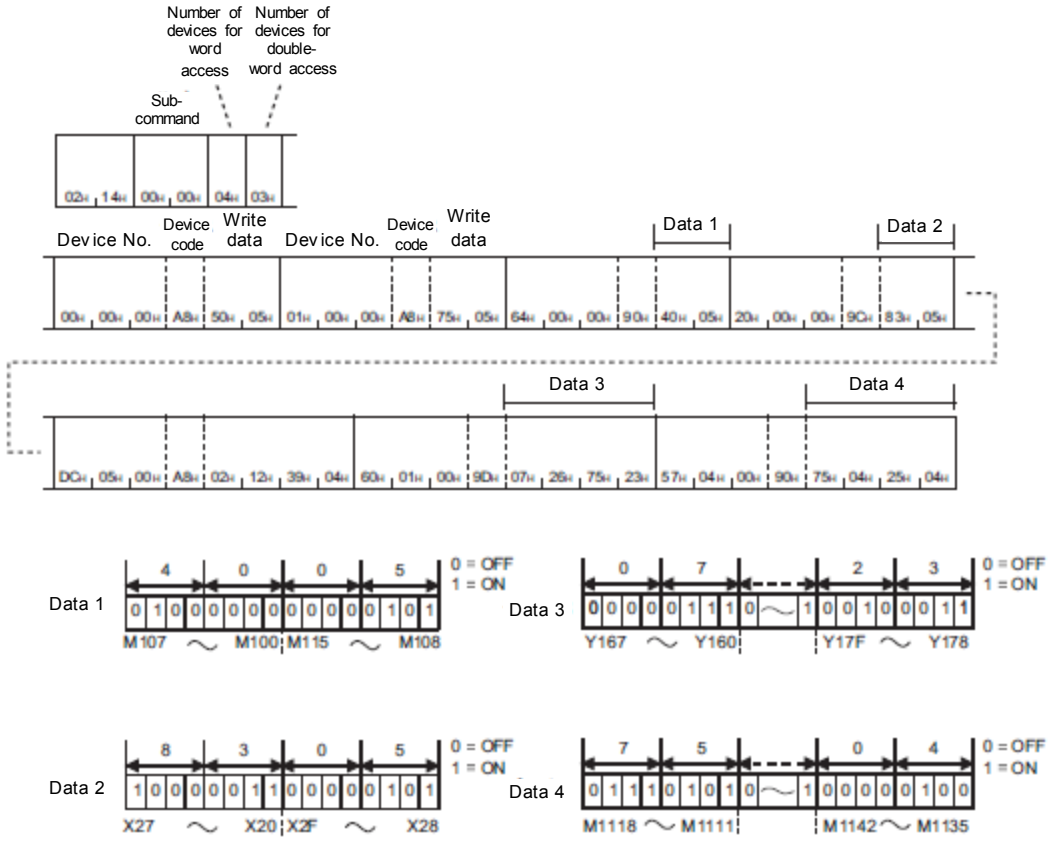
- When performing data communication in ASCII code

(Request data)





- When performing data communication in binary code
  (Request data)

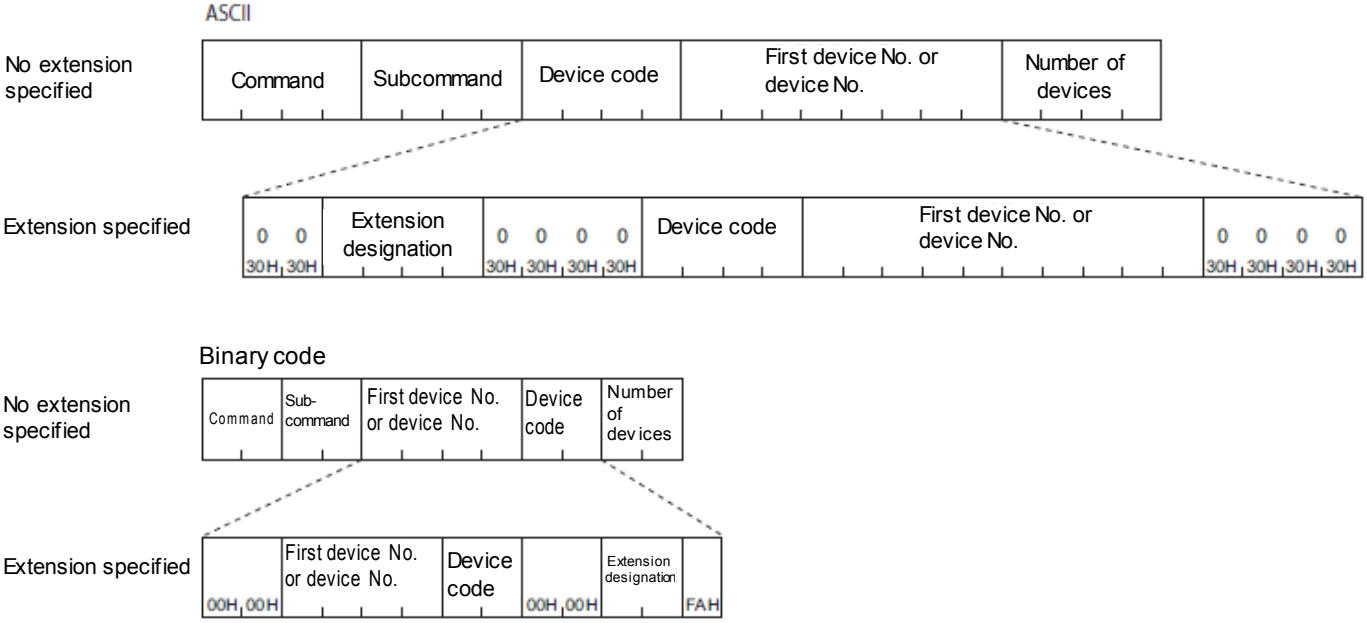


6.5.2.6. Accessing CPU Buffer Memory Access Devices

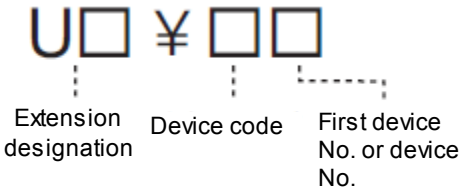
Access RCPU buffer memory.

■ Request data

For Read (Command: 0401), refer to the following example. For other commands, with the exception of device codes, start device numbers, and device numbers, access based on the format for each command.



CPU module access device and request data compatibility is as follows.

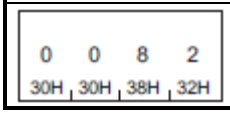
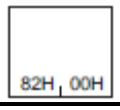


- Command

Access is possible with the following commands.

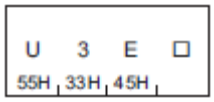
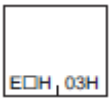
Item		Command
Category	Operation	
Device	Read	0401
	Write	1401
	Read Random	0403
	Write Random	1402

- Subcommand

ASCII code	Binary code
	

- Extension designation

  Specify the CPU module start I/O number.

ASCII code	Binary code
Specify the start I/O number in hexadecimal (3 ASCII code digits). When the start I/O number is expressed with 4 digits, specify with the first 3 digits. 	Specify the start I/O number in hexadecimal (2 bytes). When the start I/O number is expressed with 4 digits, specify with the first 3 digits. 

The start I/O numbers for the specified CPU modules are as follows.

CPU module CPU No.	Start I/O number
CPU No.1	03E0H
CPU No.2	03E1H
CPU No.3	03E2H
CPU No.4	03E3H

- Device code
  Specify the following device codes.

Device	Category	Device code		Device No. range	
		ASCII code	Binary code		
		MELSEC iQ-R series *1	MELSEC iQ-R series		
CPU buffer memory	Word	G***	00ABH	Specify within the range of device numbers held by the access destination unit.	Decimal
CPU buffer memory fixed cycle area		HG**	002EH		

*1: For ASCII code, specify device codes with 4 digits. For device codes with 3 digits or less, add an asterisk (*) (ASCII code: 2AH) or a space (ASCII code: 20H) after the device code.

- Start device or device number
  Specify the start device or device number in decimal.
  Specify the subcommand 0032/0082 device number (ASCII code) with 10 bytes (10 digits).

■ Response data
The same applies if no extension is specified.

■ Communication example
The start I/O number accesses the 03E0H CPU module buffer memory (address: 1).
Show the request data when performing communication with ASCII code.

- When performing data communication in ASCII code
  (Request data)

Subcommand	Extension designation	Device code	First device No. or device No.	
0 0 8 2	0 0	U 3 E 0	0 0 0 0	G * * *	0 0 0 0 0 0 0 0 0 0 1	0 0 0 0
30H 30H 38H 32H	30H 30H	55H 33H 45E 30H	30H 30H 30H 30H	47H 2AH 2AH 2AH	30H 30H 30H 30H 30H 30H 30H 30H 30H 31H	30H 30H 30H 30H

- When performing data communication in binary code
  (Request data)

Subcommand	First device No. or device No.	Device code	Extension designation	
82H 00H	00H 00H	01H 00H 00H 00H	ABH 00H	00H 00H	E0H 03H	FAH

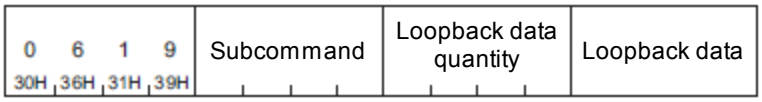
6.5.3. Self Test (Loopback Test) (Command: 0619)

Perform a test to determine whether communication between external devices and Ethernet-equipped modules is normal. By performing a loopback test, it is possible to confirm whether the connection with the external device is correct, and whether data communication is functioning properly.

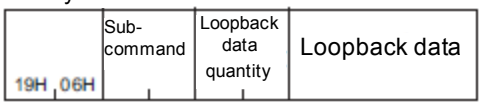
* Loopback tests can only be performed for Ethernet-equipped modules connected to external devices. Loopback tests cannot be used for other station modules connected via a network.

■ Request data

ASCII



Binary code



- Subcommand

Subcommand	
ASCII code	**Binary code**
0 0 0 0 30H 30H 30H 30H	00H 00H

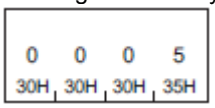
- Loopback data quantity

  Specify the "Loopback data" quantity in bytes. The range that can be specified is 1 to 960 bytes.

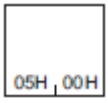
  Example

  When the loopback data quantity is 5 bytes

  When using ASCII, convert the number of bytes to 4 ASCII code digits (hexadecimal), and send them in the order from higher-order byte to lower-order byte.

  When using binary code, send the number of 2-byte characters indicating the number of bytes in the order from lower-order byte to higher-order byte.

- Loopback data

  Specify the data sent and received when performing a loopback test.

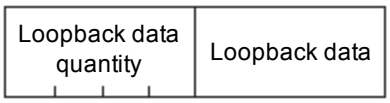
  When performing data communication in ASCII code, specify a 1-byte character string ("0" to "9", "A" to "F") for data with maximum of 960 characters, and send from the start.

  When performing data communication in binary code, convert the 1-byte character ("0" to "9", "A" to "F") code to a 1-byte numerical value, and send data with maximum of 960 bytes from the starting character code.

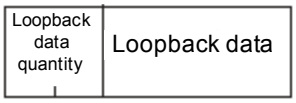
■ Request data

The same content as that in the "Loopback data quantity" and "Loopback data" specified in the request message is stored.

ASCII



Binary code

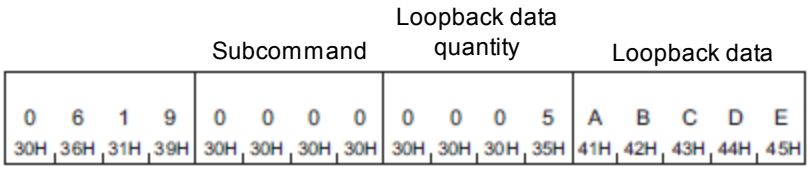


■ Communication example

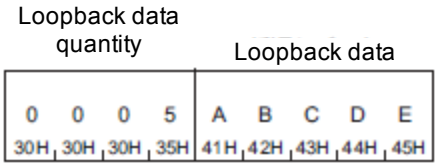
Perform a loopback test with loopback data "ABCDE".

- When performing data communication in ASCII code

(Request data)

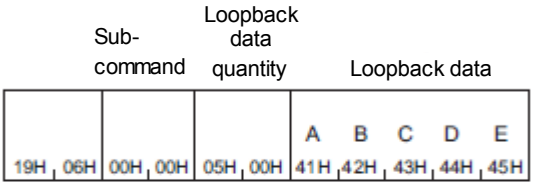


(Response data)

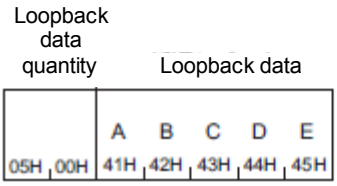


- When performing data communication in binary code

(Request data)



(Response data)



6.6. End Code

The following is a list of stored end codes.

Code category	End code	Description	Processing details
Processing success	0000H	The request was successfully processed.	Indicates that the request was correctly processed.
Standard error	C059H	• The command or subcommand is specified incorrectly. • A command other than the prescribed sequence was received.	Review the command and subcommand, and send again.
	C05CH	The request message has an error.	Review the request content, and send again.
	C061H	The request data length is inconsistent with the number of data.	Review the request data content or request data length, and send again.
	CEE1H	The request message size exceeds the allowable range.	Review the request content, and send again.
	CEE2H	The response message size exceeds the allowable range.	Review the request content, and send again.

7. Appendix

7.1. Error list

The errors which occur only when the Ethernet interface is used are listed as follows.

Error No.	Error causes and remedies
7810	■ Parameter ***** setting error of Ethernet interface parameter. Cause) ***** parameter is wrongly set. (The parameter name is input in *****.) Measures) Check the setting content of the parameter.
7820	■ MXT Command time out. Cause) The time set in parameter MXTTOUT was exceeded. Measures) Check parameter MXTTOUT.
7840	■ Received MXT command data illegal. Cause) The command argument and data type do not match. Measures) Check the contents of the command and the communication data packet to be transmitted.

For the other errors except these, refer to the errors list of the instruction manual of the controller.

7.2. Sample program

This is the sample program of the Ethernet function.

7.2.1. Sample program of data link

The sample program to do the data link with Microsoft Visual Studio Express Visual Basic (hereafter written as VB) is herein described.

The program creation is briefly introduced with the following procedure.

For details of VB operation and application producing method, refer to the instruction manual of this software.

(1) Preparation of Winsock control

(2) Production of form screen

(3) Program (Form1.frm)

There is the program following 2 passages. Use either according to the customer's system.

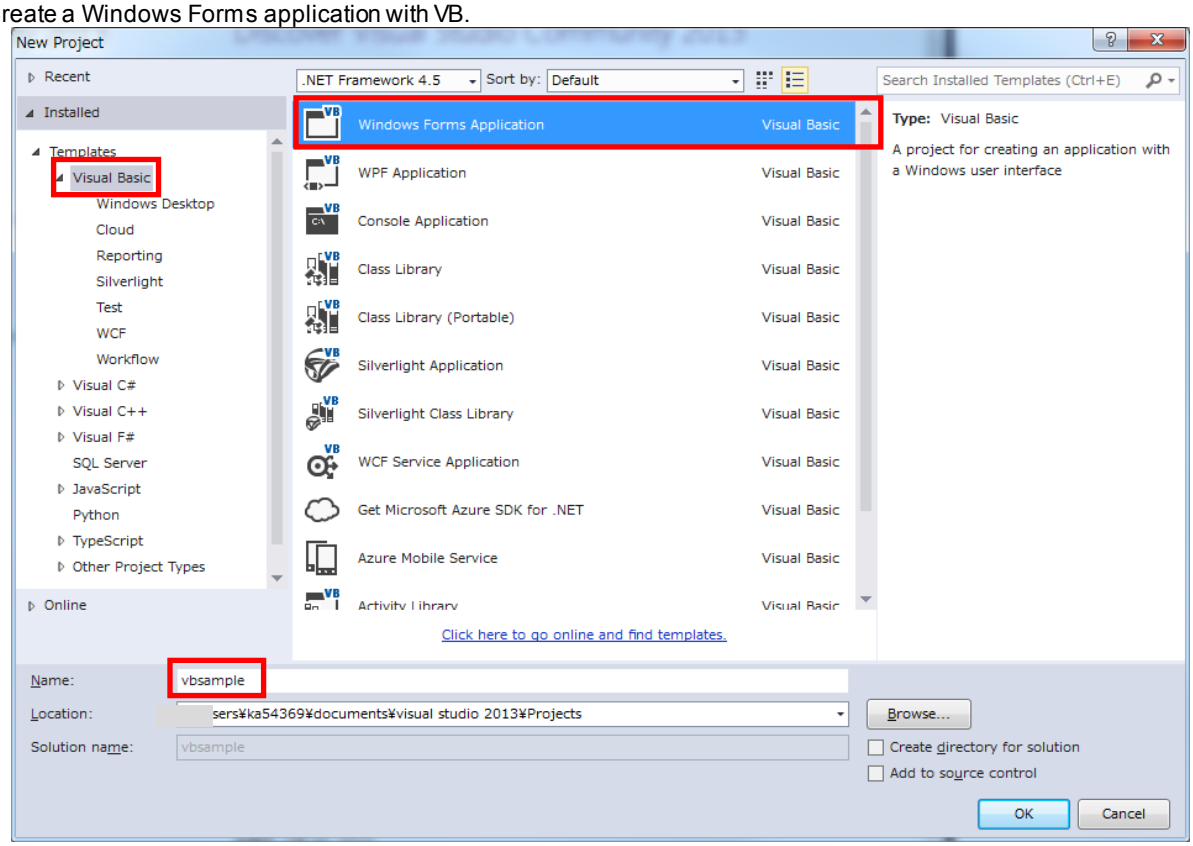
1) Program for the clients (when using the personal computer as the client and using the controller as the server).

2) Program for the server (when using the personal computer as the server and using the controller as the client).

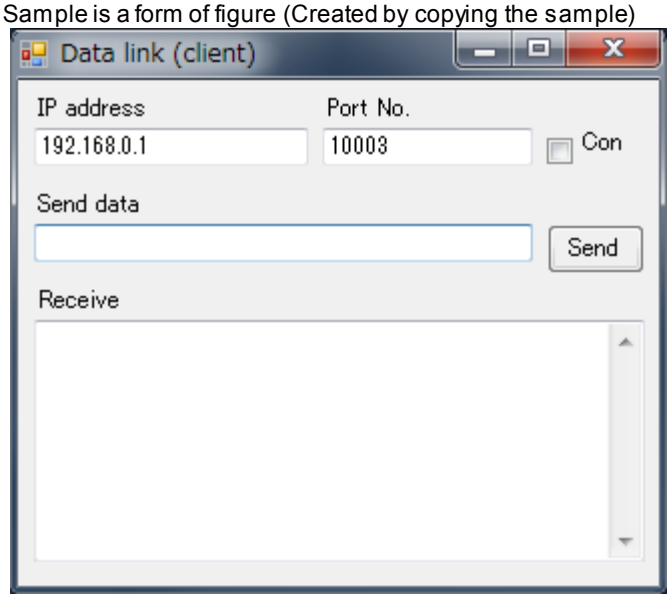
* About the work of 1) 2), the client and the server are the same.

(1) Preparation of project

Create a Windows Forms application with VB.



(2) Sample is a form of figure (Created by copying the sample)



Copy files to vbsample folder.

• Form1.Designer.vb

• Form1.vb

Be careful not to confuse the client and the server.

Each text files saved from pdf manual.

■ Form1.Designer.vb (Form for the client)

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class Form1
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
    Private Sub InitializeComponent()
        Me.components = New System.ComponentModel.Container
        Me.Button1 = New System.Windows.Forms.Button
        Me.Check1 = New System.Windows.Forms.CheckBox
        Me.Text4 = New System.Windows.Forms.TextBox
        Me.Text3 = New System.Windows.Forms.TextBox
        Me.Text2 = New System.Windows.Forms.TextBox
```

```
Me.Text1 = New System.Windows.Forms.TextBox
Me.Label4 = New System.Windows.Forms.Label
Me.Label3 = New System.Windows.Forms.Label
Me.Label2 = New System.Windows.Forms.Label
Me.Label1 = New System.Windows.Forms.Label
Me.Timer1 = New System.Windows.Forms.Timer(Me.components)
Me.SuspendLayout()
'
'Button1
'
Me.Button1.BackColor = System.Drawing.SystemColors.Control
Me.Button1.Cursor = System.Windows.Forms.Cursors.Default
Me.Button1.ForeColor = System.Drawing.SystemColors.ControlText
Me.Button1.Location = New System.Drawing.Point(264, 72)
Me.Button1.Name = "Button1"
Me.Button1.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Button1.Size = New System.Drawing.Size(49, 25)
Me.Button1.TabIndex = 16
Me.Button1.Text = "Send"
Me.Button1.UseVisualStyleBackColor = False
'
'Check1
'
Me.Check1.BackColor = System.Drawing.SystemColors.Control
Me.Check1.Cursor = System.Windows.Forms.Cursors.Default
Me.Check1.ForeColor = System.Drawing.SystemColors.ControlText
Me.Check1.Location = New System.Drawing.Point(264, 24)
Me.Check1.Name = "Check1"
Me.Check1.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Check1.Size = New System.Drawing.Size(49, 25)
Me.Check1.TabIndex = 14
Me.Check1.Text = "Connection"
Me.Check1.UseVisualStyleBackColor = False
'
'Text4
'
Me.Text4.AcceptsReturn = True
Me.Text4.AcceptsTab = True
Me.Text4.BackColor = System.Drawing.SystemColors.Window
Me.Text4.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text4.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text4.Location = New System.Drawing.Point(8, 120)
Me.Text4.MaxLength = 0
Me.Text4.Multiline = True
Me.Text4.Name = "Text4"
Me.Text4.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text4.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
Me.Text4.Size = New System.Drawing.Size(305, 121)
Me.Text4.TabIndex = 17
'
'Text3
'
Me.Text3.AcceptsReturn = True
Me.Text3.BackColor = System.Drawing.SystemColors.Window
Me.Text3.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text3.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text3.Location = New System.Drawing.Point(8, 72)
Me.Text3.MaxLength = 0
Me.Text3.Name = "Text3"
Me.Text3.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text3.Size = New System.Drawing.Size(249, 19)
```

```
Me.Text3.TabIndex = 15
'
'Text2
'
Me.Text2.AcceptsReturn = True
Me.Text2.BackColor = System.Drawing.SystemColors.Window
Me.Text2.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text2.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text2.Location = New System.Drawing.Point(152, 24)
Me.Text2.MaxLength = 0
Me.Text2.Name = "Text2"
Me.Text2.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text2.Size = New System.Drawing.Size(105, 19)
Me.Text2.TabIndex = 13
Me.Text2.Text = "10003"
'
'Text1
'
Me.Text1.AcceptsReturn = True
Me.Text1.BackColor = System.Drawing.SystemColors.Window
Me.Text1.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text1.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text1.Location = New System.Drawing.Point(8, 24)
Me.Text1.MaxLength = 0
Me.Text1.Name = "Text1"
Me.Text1.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text1.Size = New System.Drawing.Size(137, 19)
Me.Text1.TabIndex = 12
Me.Text1.Text = "192.168.0.1"
'
'Label4
'
Me.Label4.BackColor = System.Drawing.SystemColors.Control
Me.Label4.Cursor = System.Windows.Forms.Cursors.Default
Me.Label4.ForeColor = System.Drawing.SystemColors.ControlText
Me.Label4.Location = New System.Drawing.Point(8, 104)
Me.Label4.Name = "Label4"
Me.Label4.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Label4.Size = New System.Drawing.Size(65, 13)
Me.Label4.TabIndex = 19
Me.Label4.Text = "Receive data"
'
'Label3
'
Me.Label3.BackColor = System.Drawing.SystemColors.Control
Me.Label3.Cursor = System.Windows.Forms.Cursors.Default
Me.Label3.ForeColor = System.Drawing.SystemColors.ControlText
Me.Label3.Location = New System.Drawing.Point(8, 56)
Me.Label3.Name = "Label3"
Me.Label3.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Label3.Size = New System.Drawing.Size(65, 13)
Me.Label3.TabIndex = 18
Me.Label3.Text = "Send data"
'
'Label2
'
Me.Label2.BackColor = System.Drawing.SystemColors.Control
Me.Label2.Cursor = System.Windows.Forms.Cursors.Default
Me.Label2.ForeColor = System.Drawing.SystemColors.ControlText
Me.Label2.Location = New System.Drawing.Point(152, 8)
Me.Label2.Name = "Label2"
```

```
        Me.Label2.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Label2.Size = New System.Drawing.Size(65, 13)
        Me.Label2.TabIndex = 11
        Me.Label2.Text = "Port No."
        '
        'Label1
        '
        Me.Label1.BackColor = System.Drawing.SystemColors.Control
        Me.Label1.Cursor = System.Windows.Forms.Cursors.Default
        Me.Label1.ForeColor = System.Drawing.SystemColors.ControlText
        Me.Label1.Location = New System.Drawing.Point(8, 8)
        Me.Label1.Name = "Label1"
        Me.Label1.RightToLeft = System.Windows.Forms.RightToLeft.No
        Me.Label1.Size = New System.Drawing.Size(73, 17)
        Me.Label1.TabIndex = 10
        Me.Label1.Text = "IP address"
        '
        'Timer1
        '
        Me.Timer1.Interval = 50
        '
        'Form1
        '
        Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 12.0!)
        Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
        Me.ClientSize = New System.Drawing.Size(320, 253)
        Me.Controls.Add(Me.Button1)
        Me.Controls.Add(Me.Check1)
        Me.Controls.Add(Me.Text4)
        Me.Controls.Add(Me.Text3)
        Me.Controls.Add(Me.Text2)
        Me.Controls.Add(Me.Text1)
        Me.Controls.Add(Me.Label4)
        Me.Controls.Add(Me.Label3)
        Me.Controls.Add(Me.Label2)
        Me.Controls.Add(Me.Label1)
        Me.Name = "Form1"
        Me.Text = "Data link (client)"
        Me.ResumeLayout(False)
        Me.PerformLayout()

    End Sub
    Public WithEvents Button1 As System.Windows.Forms.Button
    Public WithEvents Check1 As System.Windows.Forms.CheckBox
    Public WithEvents Text4 As System.Windows.Forms.TextBox
    Public WithEvents Text3 As System.Windows.Forms.TextBox
    Public WithEvents Text2 As System.Windows.Forms.TextBox
    Public WithEvents Text1 As System.Windows.Forms.TextBox
    Public WithEvents Label4 As System.Windows.Forms.Label
    Public WithEvents Label3 As System.Windows.Forms.Label
    Public WithEvents Label2 As System.Windows.Forms.Label
    Public WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents Timer1 As System.Windows.Forms.Timer

End Class
```

■ Form1.vb (Program for the client)

```
Imports System
Imports System.Net.Sockets

Public Class Form1

    Private Client As TcpClient

    Private Sub Check1_CheckStateChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles Check1.CheckStateChanged
        ' Process for Connect or Disconnect
        Try
            If Check1.CheckState = CheckState.Checked Then
                Client = New TcpClient()
                Client.Connect(Text1.Text, Convert.ToInt32(Text2.Text)) 'Connect
                Button1.Enabled = Client.Connected
                Timer1.Enabled = Client.Connected
            Else
                Timer1.Enabled = False
                Button1.Enabled = False
                Client.GetStream().Close()      'Disconnect
                Client.Close()
            End If
        Catch ex As Exception
            Check1.Checked = False
            MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,
            MessageBoxDefaultButton.Button1)
        End Try
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        'Send process
        Try
            Dim SendBuf As Byte() = System.Text.Encoding.Default.GetBytes(Text3.Text)
            Dim Stream As NetworkStream = Client.GetStream()
            Stream.Write(SendBuf, 0, SendBuf.Length)
        Catch ex As Exception
            Client = Nothing
            Timer1.Enabled = False
            Button1.Enabled = False
            Check1.Checked = False
            MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,
            MessageBoxDefaultButton.Button1)
        End Try
    End Sub

    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick
        'Receive process
        Try
            Dim Stream As NetworkStream = Client.GetStream()
            If Stream.DataAvailable Then
                Dim bytes(1000) As Byte
                Dim strReceivedData As String = ""
                Dim datalength = Stream.Read(bytes, 0, bytes.Length)
                strReceivedData = System.Text.Encoding.Default.GetString(bytes).Substring(0, datalength)
                Text4.AppendText(strReceivedData)
                Text4.AppendText(System.Environment.NewLine)
            End If
        Catch ex As Exception
```

```
            Client = Nothing
            Timer1.Enabled = False
            Button1.Enabled = False
            Check1.Checked = False
            '
            MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,
            MessageBoxDefaultButton.Button1)
        End Try
    End Sub
End Class
```

■ Form1.Designer.vb (Form for the server)

```
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
Partial Class Form1
    Inherits System.Windows.Forms.Form

    'Form overrides dispose to clean up the component list.
    <System.Diagnostics.DebuggerNonUserCode()> _
    Protected Overrides Sub Dispose(ByVal disposing As Boolean)
        Try
            If disposing AndAlso components IsNot Nothing Then
                components.Dispose()
            End If
        Finally
            MyBase.Dispose(disposing)
        End Try
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

        'NOTE: The following procedure is required by the Windows Form Designer
        'It can be modified using the Windows Form Designer.
        'Do not modify it using the code editor.
        <System.Diagnostics.DebuggerStepThrough()> _
        Private Sub InitializeComponent()
            Me.components = New System.ComponentModel.Container
            Me.Button1 = New System.Windows.Forms.Button
            Me.Check1 = New System.Windows.Forms.CheckBox
            Me.Text4 = New System.Windows.Forms.TextBox
            Me.Text3 = New System.Windows.Forms.TextBox
            Me.Text2 = New System.Windows.Forms.TextBox
            Me.Text1 = New System.Windows.Forms.TextBox
            Me.Label4 = New System.Windows.Forms.Label
            Me.Timer1 = New System.Windows.Forms.Timer(Me.components)
            Me.Label3 = New System.Windows.Forms.Label
            Me.Label2 = New System.Windows.Forms.Label
            Me.Label1 = New System.Windows.Forms.Label
            Me.SuspendLayout()
            '
            'Button1
            '
            Me.Button1.BackColor = System.Drawing.SystemColors.Control
            Me.Button1.Cursor = System.Windows.Forms.Cursors.Default
            Me.Button1.ForeColor = System.Drawing.SystemColors.ControlText
            Me.Button1.Location = New System.Drawing.Point(264, 72)
            Me.Button1.Name = "Button1"
            Me.Button1.RightToLeft = System.Windows.Forms.RightToLeft.No
            Me.Button1.Size = New System.Drawing.Size(49, 25)
```

```
Me.Button1.TabIndex = 26
Me.Button1.Text = "Send"
Me.Button1.UseVisualStyleBackColor = False
'
'Check1
'
Me.Check1.BackColor = System.Drawing.SystemColors.Control
Me.Check1.Cursor = System.Windows.Forms.Cursors.Default
Me.Check1.ForeColor = System.Drawing.SystemColors.ControlText
Me.Check1.Location = New System.Drawing.Point(264, 24)
Me.Check1.Name = "Check1"
Me.Check1.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Check1.Size = New System.Drawing.Size(49, 25)
Me.Check1.TabIndex = 24
Me.Check1.Text = "Connection"
Me.Check1.UseVisualStyleBackColor = False
'
'Text4
'
Me.Text4.AcceptsReturn = True
Me.Text4.BackColor = System.Drawing.SystemColors.Window
Me.Text4.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text4.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text4.Location = New System.Drawing.Point(8, 120)
Me.Text4.MaxLength = 0
Me.Text4.Multiline = True
Me.Text4.Name = "Text4"
Me.Text4.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text4.ScrollBars = System.Windows.Forms.ScrollBars.Vertical
Me.Text4.Size = New System.Drawing.Size(305, 121)
Me.Text4.TabIndex = 27
'
'Text3
'
Me.Text3.AcceptsReturn = True
Me.Text3.BackColor = System.Drawing.SystemColors.Window
Me.Text3.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text3.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text3.Location = New System.Drawing.Point(8, 72)
Me.Text3.MaxLength = 0
Me.Text3.Name = "Text3"
Me.Text3.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text3.Size = New System.Drawing.Size(249, 19)
Me.Text3.TabIndex = 25
'
'Text2
'
Me.Text2.AcceptsReturn = True
Me.Text2.BackColor = System.Drawing.SystemColors.Window
Me.Text2.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text2.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text2.Location = New System.Drawing.Point(152, 24)
Me.Text2.MaxLength = 0
Me.Text2.Name = "Text2"
Me.Text2.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text2.Size = New System.Drawing.Size(105, 19)
Me.Text2.TabIndex = 23
Me.Text2.Text = "10003"
'
'Text1
'
```

```
Me.Text1.AcceptsReturn = True
Me.Text1.BackColor = System.Drawing.SystemColors.Window
Me.Text1.Cursor = System.Windows.Forms.Cursors.IBeam
Me.Text1.ForeColor = System.Drawing.SystemColors.WindowText
Me.Text1.Location = New System.Drawing.Point(8, 24)
Me.Text1.MaxLength = 0
Me.Text1.Name = "Text1"
Me.Text1.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Text1.Size = New System.Drawing.Size(137, 19)
Me.Text1.TabIndex = 22
'
'Label4
'
Me.Label4.BackColor = System.Drawing.SystemColors.Control
Me.Label4.Cursor = System.Windows.Forms.Cursors.Default
Me.Label4.ForeColor = System.Drawing.SystemColors.ControlText
Me.Label4.Location = New System.Drawing.Point(8, 104)
Me.Label4.Name = "Label4"
Me.Label4.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Label4.Size = New System.Drawing.Size(65, 13)
Me.Label4.TabIndex = 29
Me.Label4.Text = "Receive data"
'
'Timer1
'
Me.Timer1.Interval = 50
'
'Label3
'
Me.Label3.BackColor = System.Drawing.SystemColors.Control
Me.Label3.Cursor = System.Windows.Forms.Cursors.Default
Me.Label3.ForeColor = System.Drawing.SystemColors.ControlText
Me.Label3.Location = New System.Drawing.Point(8, 56)
Me.Label3.Name = "Label3"
Me.Label3.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Label3.Size = New System.Drawing.Size(65, 13)
Me.Label3.TabIndex = 28
Me.Label3.Text = "Send data"
'
'Label2
'
Me.Label2.BackColor = System.Drawing.SystemColors.Control
Me.Label2.Cursor = System.Windows.Forms.Cursors.Default
Me.Label2.ForeColor = System.Drawing.SystemColors.ControlText
Me.Label2.Location = New System.Drawing.Point(152, 8)
Me.Label2.Name = "Label2"
Me.Label2.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Label2.Size = New System.Drawing.Size(65, 13)
Me.Label2.TabIndex = 21
Me.Label2.Text = "Port No."
'
'Label1
'
Me.Label1.BackColor = System.Drawing.SystemColors.Control
Me.Label1.Cursor = System.Windows.Forms.Cursors.Default
Me.Label1.ForeColor = System.Drawing.SystemColors.ControlText
Me.Label1.Location = New System.Drawing.Point(8, 8)
Me.Label1.Name = "Label1"
Me.Label1.RightToLeft = System.Windows.Forms.RightToLeft.No
Me.Label1.Size = New System.Drawing.Size(73, 17)
Me.Label1.TabIndex = 20
```

```
            Me.Label1.Text = "IP address"
            '
            'Form1
            '
            Me.AutoScaleDimensions = New System.Drawing.SizeF(6.0!, 12.0!)
            Me.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
            Me.ClientSize = New System.Drawing.Size(320, 253)
            Me.Controls.Add(Me.Button1)
            Me.Controls.Add(Me.Check1)
            Me.Controls.Add(Me.Text4)
            Me.Controls.Add(Me.Text3)
            Me.Controls.Add(Me.Text2)
            Me.Controls.Add(Me.Text1)
            Me.Controls.Add(Me.Label4)
            Me.Controls.Add(Me.Label3)
            Me.Controls.Add(Me.Label2)
            Me.Controls.Add(Me.Label1)
            Me.Name = "Form1"
            Me.Text = "Data link (server)"
            Me.ResumeLayout(False)
            Me.PerformLayout()

        End Sub
        Public WithEvents Button1 As System.Windows.Forms.Button
        Public WithEvents Check1 As System.Windows.Forms.CheckBox
        Public WithEvents Text4 As System.Windows.Forms.TextBox
        Public WithEvents Text3 As System.Windows.Forms.TextBox
        Public WithEvents Text2 As System.Windows.Forms.TextBox
        Public WithEvents Text1 As System.Windows.Forms.TextBox
        Public WithEvents Label4 As System.Windows.Forms.Label
        Friend WithEvents Timer1 As System.Windows.Forms.Timer
        Public WithEvents Label3 As System.Windows.Forms.Label
        Public WithEvents Label2 As System.Windows.Forms.Label
        Public WithEvents Label1 As System.Windows.Forms.Label

    End Class
```

■ Form1.vb (Program for the server)

```
Imports System
Imports System.Net
Imports System.Net.Sockets
Imports System.Net.NetworkInformation
Imports System.Text

Public Class Form1

    Private Listener As TcpListener
    Private Client As TcpClient

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Text1.Enabled = False     'Disable IP address
        Text3.Enabled = False     'Disable Send data
        Button1.Enabled = False 'Disable Send button
    End Sub

    Private Sub Check1_CheckStateChanged (ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles Check1.CheckStateChanged
        'Process for Connect
        Try
```

```
        If Check1.CheckState = CheckState.Checked Then
            Dim interfaces As NetworkInterface()
            Dim _currentInterface As NetworkInterface

            'Get local IP address
            interfaces = NetworkInterface.GetAllNetworkInterfaces
            For Each NetworkInterface As NetworkInterface In interfaces
                If NetworkInterface.Name = "Local Area Connection" Then
                    _currentInterface = NetworkInterface
                    Dim properties As IPInterfaceProperties
                    properties = _currentInterface.GetIPProperties

                    If properties.UnicastAddresses.Count > 0 Then
                        For Each info As UnicastIPAddressInformation In properties.UnicastAddresses
                            Text1.Text = info.Address.ToString
                        Next
                    End If
                End If
            Next

            'Wait connection from client
            Listener = New TcpListener(IPAddress.Parse(Text1.Text), Convert.ToInt32(Text2.Text))
            Timer1.Start()
            Listener.Start()
        Else
            Client = Nothing
            Timer1.Stop()
            Button1.Enabled = False 'Disable send button
            Text3.Enabled = False
            Listener.Stop()             'Stop listen
        End If
    Catch ex As Exception
        MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,
        MessageBoxDefaultButton.Button1)
    End Try
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    'Send text
    Try
        Dim SendBuf As Byte() = System.Text.Encoding.Default.GetBytes(Text3.Text)
        Dim Stream As NetworkStream = Client.GetStream()
        Stream.Write(SendBuf, 0, SendBuf.Length)
    Catch ex As Exception
        'Disconnect
        Client = Nothing

        MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,
        MessageBoxDefaultButton.Button1)
    End Try
End Sub

Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Timer1.Tick
    'Receive process
    Try
        '
        If Client Is Nothing Then
            '
            If Listener.Pending = False Then
                Text1.Enabled = False    'Disable IP address edit
                Text3.Enabled = False    'Disable send text edit
```

```
                Button1.Enabled = False 'Disable send button
            Else
                Client = Listener.AcceptTcpClient() 'Connect with client
                Text1.Enabled = True     'Enable IP address edit
                Text3.Enabled = True     'Enable   send text edit
                Button1.Enabled = True 'Enable send button
            End If
        Else
            'Receive data
            Try
                Dim Stream As NetworkStream = Client.GetStream
                If Stream.DataAvailable Then
                    Dim bytes(1000) As Byte
                    Dim strReceivedData As String = ""
                    Dim datalength = Stream.Read(bytes, 0, bytes.Length)
                    strReceivedData = System.Text.Encoding.Default.GetString(bytes).Substring(0, datalength)
                    Text4.AppendText(strReceivedData)
                    Text4.AppendText(System.Environment.NewLine)
                End If
            Catch ex As Exception
                'Disconnect
                Client = Nothing
                MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,
                MessageBoxDefaultButton.Button1)
            End Try
        End If

    Catch ex As Exception
        MessageBox.Show(ex.Message, Me.Text, MessageBoxButtons.OK, MessageBoxIcon.Error,
         MessageBoxDefaultButton.Button1)
    End Try
End Sub

End Class
```

7.2.2. Sample program for real-time external control function

A sample program that establishes a data link using Microsoft Visual Studio Express Visual C++ (hereinafter VC) is shown below.

The procedures for creating the program are briefly explained below.

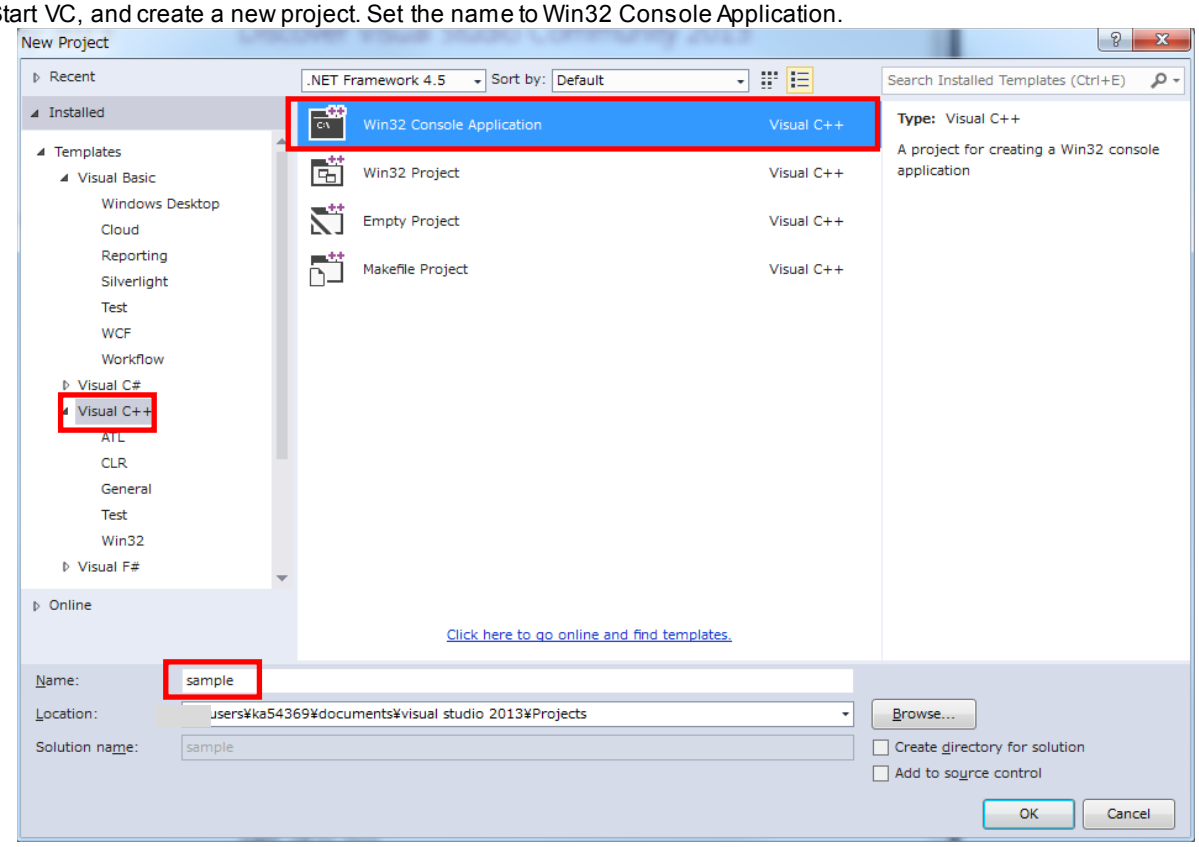
Refer to the software manuals for details on operating VC and creating the application.

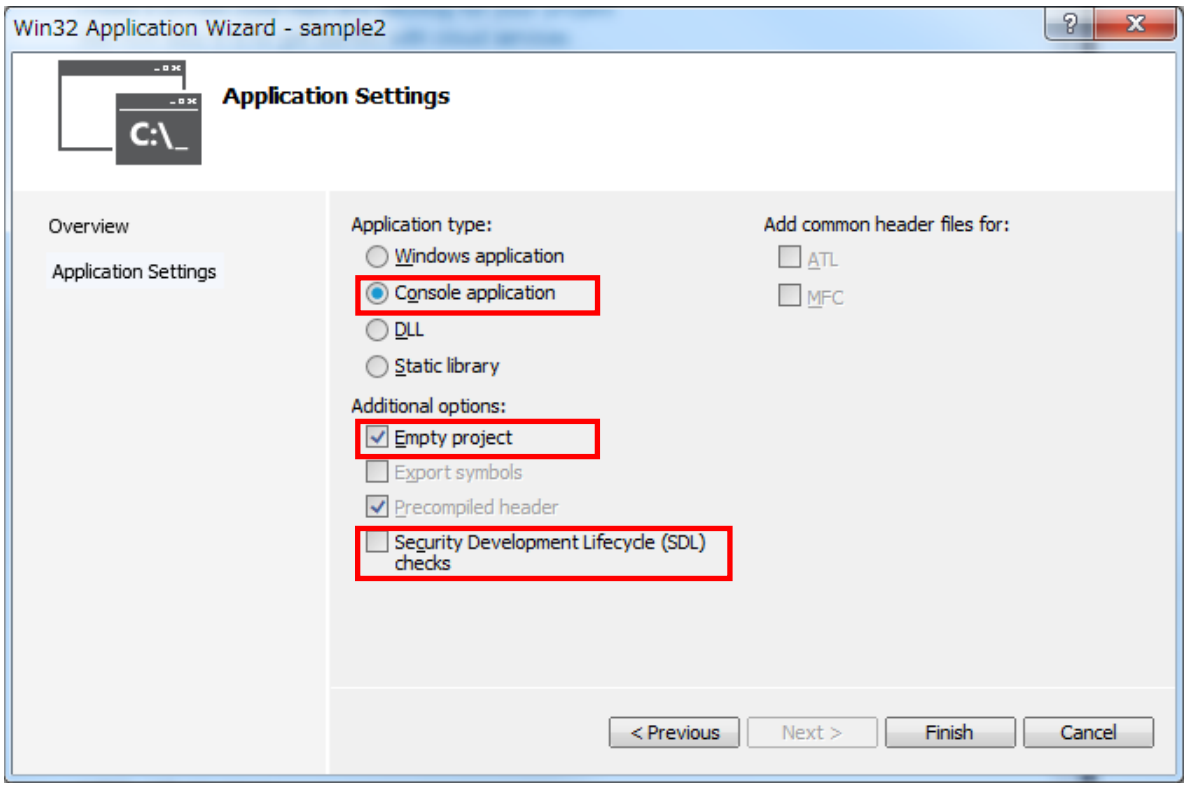
(1) Create new project

(2) Create program sample.cpp/strdef.h

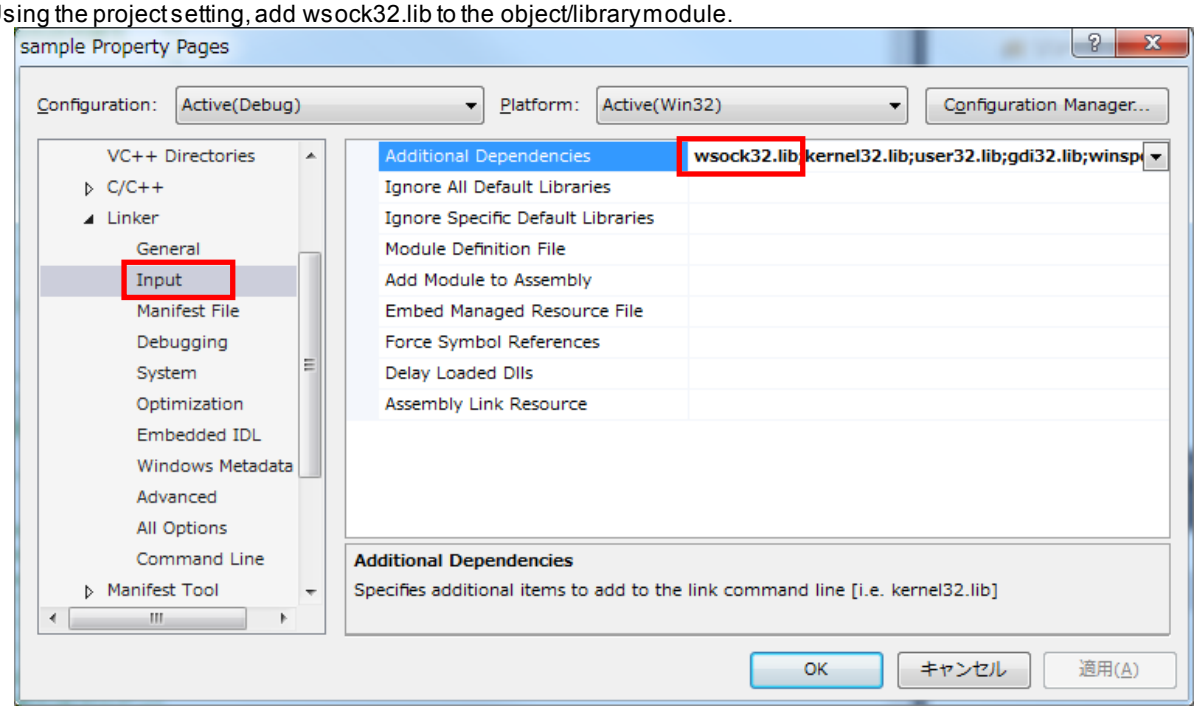
(1) Create new project

Start VC, and create a new project. Set the name to Win32 Console Application.





Using the project setting, add wsock32.lib to the object/library module.



Copy files to sample folder.

• strdef.h

・ sample.cpp

Each text files saved from pdf manual.

■ Header file strdef.h

```
//***************************************************************************
// Real-time control sample program
// Communication packet data structure definition header file
//***************************************************************************
// strdef.h

#define VER_H7

/*********************************************************************/
/* Joint coordinate system (Set unused axis to 0)                    */
/* Refer to the instruction manual enclosed                          */
/* with each robot for details on each element.                      */
/*********************************************************************/
typedef struct{
    float       j1;             // J1 axis angle (radian)
    float       j2;             // J2 axis angle (radian)
    float       j3;             // J3 axis angle (radian)
    float       j4;             // J4 axis angle (radian)
    float       j5;             // J5 axis angle (radian)
    float       j6;             // J6 axis angle (radian)
    float       j7;             // Additional axis 1 (J7 axis angle) (radian)
    float       j8;             // Additional axis 2 (J8 axis angle) (radian)
} JOINT;

/*********************************************************************/
/* XYZ  coordinate system (Set unused axis to 0)                     */
/* Refer to the instruction manual enclosed                          */
/* with each robot for details on each element.                      */
/*********************************************************************/
typedef struct{
    float       x;              // X axis coordinate value (mm)
    float       y;              // Y axis coordinate value (mm)
    float       z;              // Z axis coordinate value (mm)
    float       a;              // A axis coordinate value (radian)
    float       b;              // B axis coordinate value (radian)
    float       c;              // C axis coordinate value (radian)
    float       l1;             // Additional axis 1 (mm or radian)
    float       l2;             // Additional axis 2 (mm or radian)
} WORLD;

typedef struct{
    WORLD w;
    unsigned intsflg1;          // Structural flag 1
    unsigned intsflg2;          // Structural flag 2
} POSE;

/*********************************************************************/
/* Pulse coordinate system (Set unused axis to 0)                    */
/* These coordinates express each joint                              */
/* with a motor pulse value.                                         */
/*********************************************************************/
typedef struct{
    long p1; // Motor 1 axis
    long p2; // Motor 2 axis
    long p3; // Motor 3 axis
    long p4; // Motor 4 axis
    long p5; // Motor 5 axis
    long p6; // Motor 6 axis
    long p7; // Additional axis 1 (Motor 7 axis)
```

```
    long p8; // Additional axis 2 (Motor 8 axis)
} PULSE;

/*********************************************************/
/* Real-time function communication data packet      */
/*********************************************************/
typedef struct enet_rtcmd_str {
    unsigned short Command;             // Command
#define MXT_CMD_NULL        0           // Real-time external command invalid
#define MXT_CMD_MOVE        1           // Real-time external command valid
#define MXT_CMD_END         255         // Real-time external command end

    unsigned short  SendType;           // Command data type designation
    unsigned short  RecvType;           // Monitor data type designation
                                        ////////////  Command or monitor data type ///
#define MXT_TYP_NULL        0           // No data
                                        // For the command and monitor ////////////////////
#define MXT_TYP_POSE        1           // XYZ  data
#define MXT_TYP_JOINT       2           // Joint data
#define MXT_TYP_PULSE       3           // pulse data
                                        ////////////  For position related monitor ///
#define MXT_TYP_FPOSE       4           // XYZ  data (after filter process)
#define MXT_TYP_FJOINT      5           // Joint data (after filter process)
#define MXT_TYP_FPULSE      6           // Pulse data (after filter process)
#define MXT_TYP_FB_POSE     7           // XYZ  data (Encoder feedback value)
#define MXT_TYP_FB_JOINT    8           // Joint data (Encoder feedback value)
#define MXT_TYP_FB_PULSE    9           // Pulse data (Encoder feedback value)
                                        // For current related monitors ////////////////////
#define MXT_TYP_CMDCUR      10          // Electric current command
#define MXT_TYP_FBKCUR      11          // Electric current feedback

    union rtdata {                      // Command data
        POSE pos;                       // XYZ  type [mm/rad]
        JOINT jnt;                      // Joint type [rad]
        PULSE pls;                      // Pulse type [pls]
        long lng1[8];                   // Integer type [% / non-unit]
    } dat;

    unsigned short SendIOType;          // Send input/output signal data designation
    unsigned short RecvIOType;          // Return input/output signal data designation

#define MXT_IO_NULL         0           // No data
#define MXT_IO_OUT          1           // Output signal
#define MXT_IO_IN           2           // Input signal

    unsigned short  BitTop;             // Head bit No.
    unsigned short  BitMask;            // Transmission bit mask pattern designation (0x0001-0xffff)
    unsigned short  IoData;             // Input/output signal data (0x0000-0xffff)

    unsigned short  TCount;             // Timeout time counter value
    unsigned long   CCount;             // Transmission data counter value

    unsigned short  RecvType1;          // Reply data-type specification 1
    union rtdata1 {                     // Monitor data 1
        POSE    pos1;                   // XYZ  type [mm/rad]
        JOINT   jnt1;                   // JOINT type [mm/rad]
        PULSE   pls1;                   // PULSE type [mm/rad]
        long    lng1[8];                // Integer type [% / non-unit]
    } dat1;

    unsigned short  RecvType2;          // Reply data-type specification 2
```

```
        union rtdata2 {                     // Monitor data 2
            POSE    pos2;                   // XYZ type [mm/rad]
            JOINT   jnt2;                   // JOINT type [mm/rad]
            PULSE   pls2;                   // PULSE type [mm/rad] or Integer type [% / non-unit]
            long    lng2[8];                // Integer type [% / non-unit]
        } dat2;

        unsigned short    RecvType3;        // Reply data-type specification 3
        union rtdata3 {                     // Monitor data 3
            POSE    pos3;                   // XYZ type [mm/rad]
            JOINT   jnt3;                   // JOINT type [mm/rad]
            PULSE   pls3;                   // PULSE type [mm/rad] or Integer type [% / non-unit]
            long    lng3[8];                // Integer type [% / non-unit]
        } dat3;

    } MXTCMD;
```

■ Source file sample.cpp

```
// sample.cpp

// Change the definition in the "strdef.h" file by the S/W version of the controller.
// Refer to the "strdef.h" file for details.
//

#define _CRT_SECURE_NO_WARNINGS

#include <windows.h>
#include <iostream>
#include <winsock.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <math.h>
#include "strdef.h"
#define NO_FLAGS_SET 0

#define MAXBUFLEN 512

using namespace std;

INT main(VOID)
{
    WSADATA Data;
    SOCKADDR_IN destSockAddr;
    SOCKET destSocket;
    unsigned long destAddr;
    int status;
    int numsnt;
    int numrcv;
    char sendText[MAXBUFLEN];
    char recvText[MAXBUFLEN];
    char dst_ip_address[MAXBUFLEN];
    unsigned short    port;
    char msg[MAXBUFLEN];
    char buf[MAXBUFLEN];
    char type,type_mon[4];
    unsigned short IOSendType=0;        // Send input/output signal data designation
    unsigned short IORecvType=0;        // Reply input/output signal data designation
    unsigned short IOBitTop=0;
```

```
    unsigned short IOBitMask=0xffff;
    unsigned short IOBitData=0;

    cout << " Input connection destination IP address (192.168.0.20) ->";
    cin.getline(dst_ip_address, MAXBUFLEN);
    if(dst_ip_address[0]==0)    strcpy(dst_ip_address, "192.168.0.20");

    cout << " Input connection destination port No. (10000) -> ";
    cin.getline(msg, MAXBUFLEN);
    if(msg[0]!=0)       port=atoi(msg);
    else                                    port=10000;

    cout << " Use input/output signal?([Y] / [N])-> ";
    cin.getline(msg, MAXBUFLEN);
    if(msg[0]!=0 && (msg[0]=='Y' || msg[0]=='y')) {
        cout << " What is target? Input signal/output signal([I]nput / [O]utput)-> ";
        cin.getline(msg, MAXBUFLEN);
        switch(msg[0]) {
          case 'O':                         // Set target to output signal
          case 'o':
              IOSendType = MXT_IO_OUT;
              IORecvType = MXT_IO_OUT;
              break;
          case 'I':                         // Set target to input signal
          case 'i':
          default:
              IOSendType = MXT_IO_NULL;
              IORecvType = MXT_IO_IN;
              break;
    }

    cout << " Input head bit No. (0 to 32767)-> ";
    cin.getline(msg, MAXBUFLEN);
    if(msg[0]!=0)       IOBitTop = atoi(msg);
    else                IOBitTop = 0;

    if(IOSendType==MXT_IO_OUT) {        // Only for output signal
        cout << " Input bit mask pattern for output as hexadecimal (0000 to FFFF)-> ";
        cin.getline(msg, MAXBUFLEN);
        if(msg[0]!=0)           sscanf(msg,"%4x",&IOBitMask);
        else                    IOBitMask = 0;
        cout << " Input bit data for output as hexadecimal (0000 to FFFF)-> ";
        cin.getline(msg, MAXBUFLEN);
        if(msg[0]!=0)           sscanf(msg,"%4x",&IOBitData);
        else                    IOBitData = 0;
    }
}
cout << "--- Input the data type of command. ---¥n";
cout << "[0: None / 1: XYZ / 2:JOINT / 3: PULSE]¥n";
cout << "-- please input the number -- [0] - [3]-> ";
cin.getline(msg, MAXBUFLEN);
type = atoi(msg);

for(int k=0; k<4; k++) {
    sprintf(msg,"--- input the data type of monitor ( %d-th ) ---¥n", k);
    cout << msg;
    cout << "[0: None]¥n";
    cout << "[1: XYZ / 2:JOINT / 3: PULSE] ..... Command value¥n";
    cout << "[4: XYZ/ 5: JOINT/ 6: PULSE] ..... Command value after the filter process¥n";
    cout << "[7: XYZ/ 5:JOINT/ 6:PULSE] ..... Feedback value.¥n";
    cout << "[10: Electric current value / 11: Electric current feedback] ... Electric current value.¥n";
```

```
        cout << "Input the numeral [0] to [11] -> ";
        cin.getline(msg, MAXBUFLEN);
        type_mon[k] = atoi(msg);
    }
    sprintf(msg, "IP=%s / PORT=%d / Send Type=%d / Monitor Type0/1/2/3=%d/%d/%d/%d", dst_ip_address, port , type,
    type_mon[0], type_mon[1], type_mon[2], type_mon[3]);
    cout << msg << endl;

    cout << "[Enter]= End / [d]= Monitor data display";
    cout << "[z/x]=  Increment/decrement first command data transmitted by the delta amount. ";

    cout << " Is it all right? [Enter] / [Ctrl+C] ";
    cin.getline(msg, MAXBUFLEN);

    // Windows Socket DLL initialization
    status=WSAStartup(MAKEWORD(1, 1), &Data);
    if (status != 0)
    cerr << "ERROR: WSAStartup unsuccessful" << endl;

    // IP address, port, etc., setting
    memset(&destSockAddr, 0, sizeof(destSockAddr));
    destAddr=inet_addr(dst_ip_address);
    memcpy(&destSockAddr.sin_addr, &destAddr, sizeof(destAddr));
    destSockAddr.sin_port=htons(port);
    destSockAddr.sin_family=AF_INET;

    // Socket creation
    destSocket=socket(AF_INET, SOCK_DGRAM, 0);
    if (destSocket == INVALID_SOCKET) {
        cerr << "ERROR: socket unsuccessful" << endl;
        status=WSACleanup();
        if (status == SOCKET_ERROR)
                cerr << "ERROR: WSACleanup unsuccessful" << endl;
        return(1);
    }

    MXTCMD              MXTsend;
    MXTCMD              MXTrecv;
    JOINT               jnt_now;
    POSE                pos_now;
    PULSE               pls_now;

    unsigned long       counter = 0;
    int loop = 1;
    int disp = 0;
    int disp_data = 0;
    int ch;
    float delta=(float)0.0;
    long ratio=1;
    int   retry;
        fd_set          SockSet;        // Socket group used with select
        timeval         sTimeOut;       // For timeout setting

    memset(&MXTsend, 0, sizeof(MXTsend));
    memset(&jnt_now, 0, sizeof(JOINT));
    memset(&pos_now, 0, sizeof(POSE));
    memset(&pls_now, 0, sizeof(PULSE));

    while(loop)             {

        memset(&MXTsend, 0, sizeof(MXTsend));
```

```
memset(&MXTrecv, 0, sizeof(MXTrecv));

// Transmission data creation
if(loop==1){        // Only first time
        MXTsend.Command = MXT_CMD_NULL;
        MXTsend.SendType = MXT_TYP_NULL;
        MXTsend.RecvType = type;
        MXTsend.SendIOType = MXT_IO_NULL;
        MXTsend.RecvIOType = IOSendType;
        MXTsend.CCount = counter = 0;
}
else{                                   // Second and following times
        MXTsend.Command = MXT_CMD_MOVE;
        MXTsend.SendType = type;
        MXTsend.RecvType = type_mon[0];
        MXTsend.RecvType1= type_mon[1];
        MXTsend.RecvType2= type_mon[2];
        MXTsend.RecvType3= type_mon[3];
        switch(type){
                case MXT_TYP_JOINT:
                        memcpy(&MXTsend.dat.jnt, &jnt_now, sizeof(JOINT));
                        MXTsend.dat.jnt.j1 += (float)(delta*ratio*3.141592/180.0);
                        break;
                case MXT_TYP_POSE:
                        memcpy(&MXTsend.dat.pos, &pos_now, sizeof(POSE));
                        MXTsend.dat.pos.w.x += (delta*ratio);
                        break;
                case MXT_TYP_PULSE:
                        memcpy(&MXTsend.dat.pls, &pls_now, sizeof(PULSE));
                        MXTsend.dat.pls.p1 += (long)((delta*ratio)*10);
                        break;
                default:
                        break;
        }
        MXTsend.SendIOType = IOSendType;
        MXTsend.RecvIOType = IORecvType;
        MXTsend.BitTop = IOBitTop;
        MXTsend.BitMask =IOBitMask;
        MXTsend.IoData = IOBitData;
        MXTsend.CCount = counter;
}

// Keyboard input
// [Enter]=End / [d]= Displaythe monitor data, or none / [0/1/2/3]= Change of monitor data display
// [z/x]=Increment/decrement first command data transmitted bythe delta amount
while(_kbhit()!=0){
        ch=_getch();
        switch(ch){
        case 0x0d:
                MXTsend.Command = MXT_CMD_END;
                loop = 0;
                break;
        case 'Z':
        case 'z':
                delta += (float)0.1;
                break;
        case 'X':
        case 'x':
                delta -= (float)0.1;
                break;
        case 'C':
```

```
            case 'c':
                delta = (float)0.0;
                break;
            case 'd':
                disp = ~disp;
                break;
            case '0': case '1':         case '2': case '3':
                disp_data = ch - '0';
                break;
            }
        }

        memset(sendText, 0, MAXBUFLEN);
        memcpy(sendText, &MXTsend, sizeof(MXTsend));
        if(disp){
            sprintf(buf, "Send      (%ld):",counter);
            cout << buf << endl;
        }
        numsnt=sendto(destSocket, sendText, sizeof(MXTCMD), NO_FLAGS_SET, (LPSOCKADDR) &destSockAddr,
        sizeof(destSockAddr));
        if (numsnt != sizeof(MXTCMD)) {
            cerr << "ERROR: sendto unsuccessful" << endl;
            status=closesocket(destSocket);
            if (status == SOCKET_ERROR)
                cerr << "ERROR: closesocket unsuccessful" << endl;
            status=WSACleanup();
            if (status == SOCKET_ERROR)
                cerr << "ERROR: WSACleanup unsuccessful" << endl;
                return(1);
        }

        memset(recvText, 0, MAXBUFLEN);

        retry = 1;                          // No. of reception retries
        while(retry){
            FD_ZERO(&SockSet);              // SockSet initialization
            FD_SET(destSocket, &SockSet); // Socket registration
            sTimeOut.tv_sec = 1;            // Transmission timeout setting (sec)
            sTimeOut.tv_usec = 0;           //                                 (micro sec)
            status = select(0, &SockSet, (fd_set *)NULL, (fd_set *)NULL, &sTimeOut);
            if(status == SOCKET_ERROR){
                return(1);
            }
            if((status > 0) && (FD_ISSET(destSocket, &SockSet) != 0)) {     // If it receives by the time-out
                numrcv=recvfrom(destSocket, recvText, MAXBUFLEN, NO_FLAGS_SET, NULL, NULL);
                if (numrcv == SOCKET_ERROR) {
                    cerr << "ERROR: recvfrom unsuccessful" << endl;
                status=closesocket(destSocket);
                if (status == SOCKET_ERROR)
                    cerr << "ERROR: closesocket unsuccessful" << endl;
                status=WSACleanup();
                if (status == SOCKET_ERROR)
                    cerr << "ERROR: WSACleanup unsuccessful" << endl;
                return(1);
            }
            memcpy(&MXTrecv, recvText, sizeof(MXTrecv));
            char str[10];
            if(MXTrecv.SendIOType==MXT_IO_IN)    sprintf(str,"IN%04x", MXTrecv.IoData);
            else if(MXTrecv.SendIOType==MXT_IO_OUT)    sprintf(str,"OT%04x", MXTrecv.IoData);
            else                                                    sprintf(str,"------");
```

```
int DispType;
void *DispData;
switch(disp_data){
	case 0:
		DispType = MXTrecv.RecvType;
		DispData = &MXTrecv.dat;
		break;
	case 1:
		DispType = MXTrecv.RecvType1;
		DispData = &MXTrecv.dat1;
		break;
	case 2:
		DispType = MXTrecv.RecvType2;
		DispData = &MXTrecv.dat2;
		break;
	case 3:
		DispType = MXTrecv.RecvType3;
		DispData = &MXTrecv.dat3;
		break;
	default:
		break;
}

switch(DispType){
	case MXT_TYP_JOINT:
	case MXT_TYP_FJOINT:
	case MXT_TYP_FB_JOINT:
		if(loop==1){
			memcpy(&jnt_now, DispData, sizeof(JOINT));
			loop = 2;
		}
		if(disp){
			JOINT *j=(JOINT*)DispData;
			sprintf(buf, "Receive (%ld): TCount=%d
				Type(JOINT)=%d¥n %7.2f,%7.2f,%7.2f,%7.2f,%7.2f,%7.2f,%7.2f,%7.2f (%s)"
				,MXTrecv.CCount,MXTrecv.TCount,DispType
				,j->j1, j->j2, j->j3 ,j->j4, j->j5, j->j6, j->j7, j->j8, str);
			cout << buf << endl;
		}
		break;
	case MXT_TYP_POSE:
	case MXT_TYP_FPOSE:
	case MXT_TYP_FB_POSE:
		if(loop==1){
			memcpy(&pos_now, &MXTrecv.dat.pos,  sizeof(POSE));
			loop = 2;
		}
		if(disp){
			POSE  *p=(POSE*)DispData;
			sprintf(buf, "Receive (%ld): TCount=%d
				Type(POSE)=%d¥n %7.2f,%7.2f,%7.2f,%7.2f,%7.2f,%7.2f, %04x,%04x (%s)"
				,MXTrecv.CCount,MXTrecv.TCount,DispType
				,p->w.x, p->w.y, p->w.z, p->w.a, p->w.b, p->w.c, p->sflg1, p->sflg2, str);
			cout << buf << endl;
		}
		break;
	case MXT_TYP_PULSE:
	case MXT_TYP_FPULSE:
	case MXT_TYP_FB_PULSE:
	case MXT_TYP_CMDCUR:
	case MXT_TYP_FBKCUR:
```

```
                        if(loop==1){
                                memcpy(&pls_now, &MXTrecv.dat.pls, sizeof(PULSE));
                                loop = 2;
                        }
                        if(disp){
                                PULSE *l=(PULSE*)DispData;
                                sprintf(buf,"Receive (%ld): TCount=%d
                                        Type(PULSE/OTHER)=%d¥n %ld,%ld,%ld,%ld,%ld,%ld,%ld,%ld (%s)"
                                        ,MXTrecv.CCount,MXTrecv.TCount,DispType
                                        ,l->p1, l->p2, l->p3, l->p4, l->p5, l->p6, l->p7, l->p8, str);
                                cout << buf << endl;
                        }
                        break;
                case MXT_TYP_NULL:
                        if(loop==1){
                                loop = 2;
                        }
                        if(disp){
                                sprintf(buf,"Receive (%ld): TCount=%d    Type(NULL)=%d¥n (%s)"
                                        ,MXTrecv.CCount,MXTrecv.TCount,   DispType, str);
                                cout << buf << endl;
                        }
                        break;
                default:
                        cout << "Bad data type.¥n" << endl;
                        break;
            }
            counter++;                          // Count up only when communication is successful
            retry=0;                            // Leave reception loop
        }
        else{                                   // Reception timeout
            cout << "... Receive Timeout!   <Push [Enter] to stop the program>" << endl;
            retry--;                            // No. of retries subtraction
            if(retry==0) loop=0;                // End program if No. of retries is 0
        }
    } /* while(retry) */
} /* while(loop) */

// End
cout << "/// End /// ";
sprintf(buf,"counter = %ld", counter);
cout << buf << endl;

// Close socket
status=closesocket(destSocket);
if (status == SOCKET_ERROR)
    cerr << "ERROR: closesocket unsuccessful" << endl;
status=WSACleanup();
if (status == SOCKET_ERROR)
    cerr << "ERROR: WSACleanup unsuccessful" << endl;

return 0;
}
```

MITSUBISHI ELECTRIC CORPORATION

HEAD OFFICE: TOKYO BUILDING, 2-7-3, MARUNOUCHI, CHIYODA-KU, TOKYO 100-8310, JAPAN
NAGOYA WORKS: 5-1-14, YADA-MINAMI, HIGASHI-KU NAGOYA 461-8670, JAPAN

Authorised representative:
Mitsubishi Electric Europe B.V. FA - European Business Group
Mitsubishi-Electric-Platz 1, D-40882 Ratingen, Germany
Tel: +49(0)2102-4860

Apr., 2017 MEE Printed in Japan on recycled paper.