Modbus. X provider

Modbus ASCII/RTU/TCP communication

Version 1. 0. 5

User's guide

November 18, 2016

[Remarks]

[Revision history]

Version	Date	Contents
1. 0. 0. 0	2015-1-10	First edition
1. 0. 1. 0	2015-4-22	Added server mode
1. 0. 2. 0	2015-6-08	Addressing of DI/DO variables were fixed to 1 bit, regardless of the data width. Addressing of Holding/Input register were fixed to 16 bits, regardless of the data width.
	2015-8-18	Errors were corrected.　Added supplemental explanations.
1. 0. 3. 0	2015-8-21	Added "OffsetAddressZero" and "Endian" as options of CaoController::AddExtension.
1. 0. 4. 0	2016-6-2	Modified sample programs.
1. 0. 5. 0	2016-9-6	Added "VT" and "Elem" as options of CaoExtension:AddVariable.
1. 0. 5.1	2016-9-19	Added term definition. Added List of Comparison with Old Modbus Command names.
1. 0. 5.2	2016-11-18	Added "RcvPacketLen" and "SndPacketLen" as options of CaoExtension:AddVariabl.

[Supported device]

Model	Version	Description

Contents

1. Introduction

This is a user’s guide of Modbus. X provider.

Using Modbus. X provider allows CAO client to send and receive Modbus protocol[1] without complicated programming.

This document describes the function of Modbus. X provider and the implemented methods.

1.1. Term definition

The following names are defined (standardized) to using in this document.

- “Coil”: standardized as “DO（Discrete Output）”.
- “Input Status”: standardized as “DI（Discrete Input）”.

[1] Modbus is a serial communications protocol originally published by Modicon (now Schneider Electric) in 1979 for use with its programmable logic controllers (PLCs). Simple and robust, it has since become a de facto standard communication protocol, and it is now a commonly available means of connecting industrial electronic devices. (reference from *Wikipedia*)

For about the protocol specification of Modbus ASCII/RTU, refer to “Modicon Modbus Protocol Reference Guide PI–MBUS–300 Rev. J”. For about the protocol specification of Modbus TCP, refer to “OPEN MODBUS/TCP SPECIFICATION Release 1.0, 29 March 1999”.

2. Overview of Modbus. X provider

2.1. Overview

Modbus. X provider sends and receives Modbus protocol.

When a communication device is “com” (a serial device (EIA-485), such as RS232C/RS485), if the communication mode is client mode, the provider works as a Modbus master in order to perform a serial communication against Modbus slave devices. If a communication mode is server mode, the provider works as a Modbus slave in order to respond to a serial communication from a Modbus client device.

When a communication device is “eth” (Ethernet), if the communication mode is client mode, the provider performs TCP/IP communication with Modbus server devices. If a communication mode is server mode, the provider responds to TCP/IP communication from a Modbus client device.

Hereafter, a **master** is called a **Client** and a **slave** is called a **Server**, regardless of a communication device.

Table 2-1 Communication device and communication mode

		Communication mode	
Modbus communication protocol	Communication device	Client	Server
ASCII, RTU	com	✓	✓
TCP	eth	✓	✓*

*In TCP protocol, up to 16 clients can be connected.

The file format of Modbus. X provider is DLL (Dynamic Link Library). Table 2-2 shows the details.

Table 2-2 Modbus. X provider

File name	CaoProvModbusX. dll
ProgID	CaoProv. Modbus. X
Registry registration[2]	regsvr32 CaoProvModbusX. dll
Delete registry registration	regsvr32 /u CaoProvModbusX. dll

[2]. It is not necessary manual registration/deregistration when installing it with ORiN SDK.

2.2. Execution mode

Modbus. X provider has two execution modes; Synchronous mode and Asynchronous mode. To change the execution mode, specify Sync option of AddController.

2.2.1. Asynchronous mode

2.2.1.1. Client mode

Send a Modbus request (query) message by executing *CaoExtension: : Execute() Modbus* function compatible command (see Table 2-13) or by accessing *CaoExtension::Cao(user variable)* (see Table 2-15). Once a response message arrives from a server device, OnMessage event is generated.

2.2.1.2. Server mode

Once a request (query) message arrives from a client device, OnMessage event is generated. After that, the provider sends a response message to the client device by using *CaoMessage::Reply()* method of CaoMessage object, which has been obtained by OnMessage event.

2.2.2. Synchronous mode

2.2.2.1. Client mode

Send and receive Modbus communication message by executing *CaoExtension:: Execute() Modbus* function compatible command (see Table 2-13) or by accessing *CaoExtension : CaoVariable(user variable)*(see Table 2-15).

2.2.2.2. Server mode

To receive a request (query) message from a client device, use ReceiveQuery command. To send a response message to a client device, use SendReply command.

2.3. Method and Property

2.3.1. CaoWorkspace: : AddController method

Modbus. X provider establishes communication by setting connection parameters when AddController is executed. At that time, specify option items (communication style, connection parameters, and time-out).

Syntax AddController(<bstrCtrlName: BSTR>, <bstrProvName: BSTR>,
<bstrPcName: BSTR > [, <bstrOption: BSTR>])

bstrCtrlName : [in] Controller name

bstrProvName : [in] Provider name. Fixed value =” CaoProv. Modbus. X”.

bstrPcName : [in] Computer name where provider operates

bstrOption : [in] Option character string

The following shows the list of option character string and the description. In this table, communication devices that shows “-“ will be ignored when the corresponding option is selected.

Table 2-3 Option character string of CaoWorkspace: : AddController

Option	Description	Server		Client	
		eth	com	eth	com
Client [=True / False]	Specify communication mode (client /server). True: Client (default) False: Server Note: In Server mode, the number of connectable clients is 16.	✓	✓	✓	✓
ServerUnitAddress [=<Server device address>]	Specify a server device address or unit identifier. For com: Server device address (Range : 1 to 255) Default : 1 For eth: Unit identifier (Range : -1, 0 to 255) Default : -1 Note: When the communication device is “eth”, if “-1“ is selected, unit identifier will be arbitrary. The provider will send a reception notification for all unit identifiers arrived from client devices. Note: This option is ignored in the client mode.	✓	✓	-	-
Sync [=True / False]	Enable or disable the synchronous mode. True: Synchronous mode (default) False: Asynchronous mode	✓	✓	✓	✓

Conn =<Connection parameter >	Mandatory. Enter the communication style and connection parameter. (See 2.3.1.1)	✓	✓	✓	✓
PacketType [=<Packet parameter>]	Enter a data type of Modbus communication protocol. 0: RTU (default) 1: ASCII Note: If “eth” is selected for the communication device in Conn option, the internal data type of TCP/IP is fixed to RTU. In this case, this option is ignored.	-	✓	-	✓
TcpConnectionTimeout [=<TCP connection timeout>]	Specify a TCP connection timeout period [ms]. TCP connection is disconnected if a request (query) message does not arrive within the specified time. Range : 0 to 3600000 (default : 0) If 0 is entered, the timeout will be invalid. Note: If the machine is in client mode or if “com” is selected for the communication device in Conn option, this option is ignored.	✓	-	-	-
ReceiveQueryTimeout [=<Query reception timout]	Specify a query reception timeout period [ms]. Range : 0 to 100000 (default : 0) Note: This option is ignored in the client mode.	✓	✓	-	-
SendReplyTimeout [=<Reply timeout>]	Specify reply timeout period [ms]. Range : 1 to 100000 (default : 1000) Note: This option is ignored in the client mode.	✓	✓	-	-
Timeout [=<Send/Receive timeout>]	Specify sending and receiving timeout period [ms]. Range : 1 to 100000 (default : 1000) Note: This option is ignored in the server mode.	-	-	✓	✓
Retry [=<Retry count>]	Specify the number of communication retry at data sending/receiving. Range : 0 to 10 (default : 0) Note: This option is ignored in the server mode.	-	-	✓	✓

OffsetAddressZero [=<True/False>]	Specify the register address value where the Execute command parameter specifies or the register address value where the letter *?* written in the end of each user variable name specifies. True: Offset value of register address starts from 0. False: Offset value of egister address starts from 1. (default : False) Note: This option is ignored in the server mode.	-	-	✓	✓
RtsTransmitDelayTime [=<Sending/receiving switching delay time>]	Specify the sending/receiving switching delay by RTS signal [ms]. 0: RTS signal is always ON (default) 1 to 100000: RTS signal affects sending/receiving circuit. Turn the RTS signal ON at immedately before the sending → Data sending start → *Data sending completion※1* → Turn the RTS signal OFF once the specified period passes. Note: ・This option is mainly used in the following hardware configuration conditions; - Transmission mode is half duplex, and, - Change of sending/receiving requires software. Specifying 1 ms or larger allows to change sending/receiving by RTS signal. ・The judgment of *Data sending completion*1* of inside of this provider is earlier than the sending completion on the actual transmission line. This is because the sending completion in this provider is determined based on the sending completion notification from the communication device driver. (The calculation of the delay time for the actual sending completion differs depending on the vendor if FIFO on the communication hardware is used. It is recommended not to use FIFO for delay time calculation.) ・If “com” is selected for the communication device in Conn option, this option is ignored	-	✓	-	✓
PollDelayTime [=<Polling delay time>]	Specify the polling delay time [ms]. Range : 0 to 100000 (default : 0) Note: This option is ignored in the server mode.	-	-	✓	✓

Endian [=<Int>[: <Float>]]	Specify the data transmission order (endian) for 32-bit command <Int> 1: 32-bit integer/ floating point big endian (from the left-most word to the right-most word) 0: 32-bit integer/floating point little endian (from the right-most word to the left-most word) (default) If the following <Float> entry is omitted, the value of <Float> will be the same as the one entered in <Int>. <Float> 1: 32-bit floating point type big endian 0: 32-bit floating point type little endian Note: This option is ignored in the server mode.	－	－	✓	✓

2.3.1.1. Conn option

The following shows the connection parameter character string of Conn option. Items enclosed by square brackets are omittable. The underlined value in each parameter description represents the default value which will be used when any option is not specified.

- **RS232C/RS485 device**

"Conn=com: <COM Port>[: <BaudRate>[: <Parity>: <DataBits>: <StopBits>]]"

<COM Port> : COM port number. '1' -COM1, '2'-COM2, …

<BaudRate> : Baud rate. 1200, 4800, 9600, 19200, 38400, 57600, 115200, Max (The baud rate is determined depending on the UART hardware specification used.)

<Parity> : Parity. 'N'-NONE, 'E'-EVEN, 'O'-ODD.

<DataBits> : Data bit count. '7'-7 bit, '8'-8 bit.

Note: When the PacketType option is RTU mode, the data bit count is always "8", therefore, an error occurs if other than "8" is specified.

<StopBits> : Stop bit count. '1'-1 bit, '2'-2 bit.

※Flow control setting is fixed to NONE because RtsTransmitDelayTime option uses RTS signal.

- **Ethernet device**

"Conn=eth: <IP Address>[: <Port No>]"

<IP Address> : <For Client mode>

IP address of destination server device

<For Server mode>

IP address of the target client device

Note: Enter an arbitral IP address. Up to 16 clients are connected when (0. 0. 0. 0) is specified.

<Port No> : TCP connection port number

Default : 502

2.3.2. CaoController: : Execute method

For available command names and details, refer to 2.4.1.

Syntax Execute(< bstrCommand: BSTR > [, <vntParam: VARIANT>[, < pVal: VARIANT>]])

bstrCommand	:	[in] Command name
vntParam	:	[in] Parameter
pVal	:	[out] Obtained data

2.3.3. CaoController: : AddVariable method

Create a variable object. For variable names, you can use variables written in 2.5.1 only.

Syntax AddVariable(<bstrName: BSTR > [, <bstrOption: BSTR>])

bstrName	:	[in] Arbitral name
bstrOption	:	[in] Option character string (unused)

2.3.4. CaoController: : GetVariableNames property

Create a variable name list described in 2.5.1.

2.3.5. CaoController: : AddExtension method (for client mode only)

In the Client mode, create a CaoExtension that communicates with a Modbus server device.

Syntax AddExtension (<bstrName: BSTR > [, <bstrOption: BSTR>])

bstrName	:	[in] Arbitral name
bstrOption	:	[in] Option character string

The following table shows available "Option character string ".

Table 2-4 Option character string of CaoController: AddExtension

Option	Description
UnitAddress[=<Device address>]	Specify a server device address (for com) or a unit identifier (for eth) of communication destination. For com: Server device address (Range : 0 to 255) default : 1 For eth: Unit identifier (Range : 0 to 255) default : 0

OffsetAddressZero [=<True/False>]	For each server device address, specify the followings (0 or higher): :the register address value which is specified by a parameter of Execute command, and, : the offset value of "?" (register address) which is placed on the end of each user variable name. For details, refer to “OffsetAddressZero” option on Table 2-3. Default: The setting value of OffsetAddressZero option at the execution of CaoWorkspace::AddController.
Endian [=<Int>[:<Float]>]	For each server device address, set the order of the data transmission (endian) for a 32-bit command. For details, refer to “Endian” option on Table 2-3. Default: The setting value of Endian option at the execution of CaoWorkspace::AddController.

2.3.6. CaoExtension: : GetID method (for client mode only)

In the Client mode, obtain a server device address that is specified by “UnitAddress”.

2.3.7. CaoExtension: : Execute method (for client mode only)

This method is for the Client mode. It executes a *Execute* command that communicates with a server device. Enter an actual command executed for the server device in the first argument (bstCommand). Command names are defined according to the Modbus function codes described in Table 2-12.

Syntax Execute(< bstrCommand: BSTR > [, <vntParam: VARIANT>[, < pVal: VARIANT>]])

bstrCommand	:	[in] Command name
vntParam	:	[in] Parameter
pVal	:	[out] Obtained data

For about available command names, refer to Table 2-13.

2.3.8. CaoExtension: : AddVariable method (for client mode only)

In the Client mode, create a variable object that communicates with a Modbus server device. For a variable name, only variables written in Table 2-15 are available. If other variable names are specified, an error occurs.

Syntax AddVariable(<bstrName: BSTR > [, <bstrOption: BSTR>])

bstrName	:	[in] Arbitral name
bstrOption	:	[in] Option character string

The following table lists option character strings.

Table 2-5 Option character string of CaoExtension: AddVariable

Option	Description
UserVarWidth[=<User variable data width >]	Specify the data width [bit] of user variables. [Range] ・<bstrName> = For *DO?* or *DI?*: 1(default), 8, 16, 32 [bit] ・<bstrName> = For *HRI?* or *IRI?*: 16(default), 32 [bit] ・<bstrName> = For *HRF?* or *IRF?*: fixed to 32 [bit] (default) Note: Without being specified this option, if ”VT” option is specified, the ”VT” option is ignored.
VT[=<Variable type>]	Specify the data type. (For details, refer to Table 2-6) [corresponding bstrName] <bstrName> = DO? or DI? or HRI? or IRI? Note: When specifying HRF? or IRF?, this option is ignored. Variable type is VT_R4-fixed. When omitted: It will become invalid, and ”UserVarWidth” option is given priority. Note: When ”UserVarWidth” is specified, this option is ignored.
Elem[=<Number of element>]	Specify the number of elements of the data. For the range of <bstrName> and Number of element, refer to Table 2-6. “ VT ” option which is variable type other than BSTR becomes array-type (VT_ARRAY), and specifies the number of element. “VT”option which is variable type of BSTR specifies String variable (in bytes). (Note: This is not Number of element of array-type (VT_ARRAY). Number of elements can be specified with decimal or hexadecimal e.g., 0x0A, &h0A, 0AH [corresponding bstrName] <bstrName> = DO? or DI? or HRI? or IRI? or HRF? or IRF? When omitted: It will become invalid. (Variable type disables assigning.) ※Data type which is VT option omitted is as follows: For the range of Number of element of each data type and/or ?(address), refer to Table 2-6. [<bstrName> = DO? or DI?] UserVarWidth= 1： VT_ARRAY | VT_BOOL UserVarWidth= 8： VT_ARRAY | VT_UI1 UserVarWidth=16： VT_ARRAY | VT_UI2 UserVarWidth=32： VT_ARRAY | VT_UI4 [<bstrName> = HRI? or IRI?] UserVarWidth=16： VT_ARRAY | VT_UI2 UserVarWidth=32： VT_ARRAY | VT_UI4 ・・・”Endian”option ：valid [<bstrName> = HRF? or IRFI?] VT_ARRAY | VT_R4 ・・・”Endian”option ：valid
RcvPacketLen[=<Receive Packet Length >]	The maximum value of the length of receive packet is specified in units of WORD when receiving the data. Range: 4 to 125 Omitted or out of the range: 125

SndPacketLen[=<Send Packet Length >]	The maximum value of the length of send packet is specified in units of WORD when sending the data. Range: 4 to 123 Omitted or out of the range: 123

Table 2-6 List of Data types available to VT option

VT	Data Types		Description
	“Elem“ non-specified	“Elem“ specified	
BIT	VT_UI1	VT_ARRAY | VT_UI1 [Range of the number of elemts] “DO?”or“DI?: 1 to 65536 “HRI?”or“IRI?”: 1 to 131072	The data is convered into 0 or 1, and written/ read. Val==0：0, Val!=0：1 [?(Address)Range] “OffsetAddressZero=False”: 1 to 65536 “OffsetAddressZero=True” : 0 to 65535
BOOL	VT_BOOL	VT_ARRAY | VT_ BOOL [Range of the number of elemts] “DO?”or“DI?”: 1 to 65536 “HRI?”or“IRI?”: 1 to 65536	The data is convered into 0 or -1, and written/ read. Val==VARIANT_FALSE：0,Val!=VARIANT_FALSE：-1 [?(Address)Range] “OffsetAddressZero=False” : 1 to 65536 “OffsetAddressZero=True” : 0 to 65535
BSTR	VT_BSTR	VT_BSTR [Range of the number of elemts] “DO?”or“DI?”: 1 to 8192 “HRI?”or“IRI?”: 1 to 131072	Write/Read BSTR as ASCII. When “Elem“non-specified: One character (byte) When “Elem” specified: The number of designated characters (in units of byte) [?(Address)Range] <“DO?”or“DI?”> “OffsetAddressZero=False” : 1 to 65529 “OffsetAddressZero=True” : 0 to 65528 <“HRI?”or“IRI?”> “OffsetAddressZero=False” : 1 to 65536 “OffsetAddressZero=True” : 0 to 65535
I1	VT_I1	VT_ARRAY | VT_I1 [Range of the number of elemts] “DO?”or“DI?”: 1 to 8192 “HRI?”or“IRI?”: 1 to 131072	It can be written/ read as Signed 1-byte data. [?(Address)Range] <“DO?”or“DI?”> “OffsetAddressZero=False”：1 to 65529 “OffsetAddressZero=True”：0 to 65528 <“HRI?”or“IRI?”> “OffsetAddressZero=False”：1 to 65536 “OffsetAddressZero=True”：0 to 65535
I2	VT_I2	VT_ARRAY | VT_ I2	It can be written/ read as Signed 2-byte data.

		[Range of the number of elemts] “DO?”or“DI?”: 1 to 4096 “HRI?”or“IRI?”: 1 to 65536	[?(Address)Range] <“DO?”or“DI?”> “OffsetAddressZero=False” : 1 to 65521 “OffsetAddressZero=True” : 0 to 65520 <“HRI?”or“IRI?”> “OffsetAddressZero=False” :1 to 65536 “OffsetAddressZero=True” : 0 to 65535
I4	VT_I4	VT_ARRAY | VT_I4 [Range of the number of elemts] “DO?”or“DI?” : 1 to 2048 “HRI?”or“IRI?” :1 to 32768	It can be written/ read as Signed 4-byte data. [?(Address)Range] <“DO?”or“DI?”> ･･･”Endian”option :invalid “OffsetAddressZero=False” :1 to 65505 “OffsetAddressZero=True” : 0 to 65504 <“HRI?”or“IRI?”> ･･･”Endian”option :valid “OffsetAddressZero=False” :1 to 65535 “OffsetAddressZero=True” : 0 to 65534
I8	VT_I8	VT_ARRAY | VT_I8 [Range of the number of elemts] “DO?”or“DI?” : 1 to 1024 “HRI?”or“IRI?” : 1 to 16384	It can be written/ read as Signed 8-byte data. [?(Address)Range] <“DO?”or“DI?”> ･･･”Endian”option :invalid “OffsetAddressZero=False” :1 to 65473 “OffsetAddressZero=True” : 0 to 65472 <“HRI?”or“IRI?”> ･･･”Endian”option :valid “OffsetAddressZero=False” :1 to 65533 “OffsetAddressZero=True” : 0 to 65532
UI1	VT_UI1	VT_ARRAY | VT_UI1 [Range of the number of elemts] “DO?”or“DI?”: 1 to 8192 “HRI?”or“IRI?”: 1 to 131072	It can be written/ read as Unsigned 1-byte data. [?(Address)Range] <“DO?”or“DI?”> “OffsetAddressZero=False” :1 to 65529 “OffsetAddressZero=True” :0 to 65528 <“HRI?”or“IRI?”> “OffsetAddressZero=False” : 1 to 65536 “OffsetAddressZero=True” : 0 to 65535
UI2	VT_UI2	VT_ARRAY | VT_UI2 [Range of the number of elemts] “DO?”or“DI?” : 1 to 4096 “HRI?”or“IRI?” : 1 to 65536	It can be written/ read as Unsigned 2-byte data. [?(Address)Range] <“DO?”or“DI?”> “OffsetAddressZero=False” : 1 to 65521 “OffsetAddressZero=True” : 0 to 65520 <“HRI?”or“IRI?”> “OffsetAddressZero=False” : 1 to 65536 “OffsetAddressZero=True” : 0 to 65535
UI4	VT_UI4	VT_ARRAY | VT_UI4 [Range of the number of	It can be written/ read as Unsigned 4-byte data. [?(Address)Range]

		elemts] “DO?”or“DI?” : 1 to 2048 “HRI?”or“IRI?” : 1 to 32768	<“DO?”or“DI?”> ・・・”Endian”option :invalid “OffsetAddressZero=False” : 1 to 65505 “OffsetAddressZero=True” : 0 to 65504 <“HRI?”or“IRI?”> ・・・”Endian”option :valid “OffsetAddressZero=False” : 1 to 65535 “OffsetAddressZero=True” : 0 to 65534
UI8	VT_UI8	VT_ARRAY | VT_UI8 [Range of the number of elemts] “DO?”or“DI?” : 1 to 1024 “HRI?”or“IRI?” : 1 to 16384	It can be written/ read as Unsigned 8-byte data. [?(Address)Range] <“DO?”or“DI?”> ・・・”Endian”option :invalid “OffsetAddressZero=False” : 1 to 65473 “OffsetAddressZero=True” : 0 to 65472 <“HRI?”or“IRI?”> ・・・”Endian”option :valid “OffsetAddressZero=False” : 1 to 65533 “OffsetAddressZero=True” : 0 to 65532
R4	VT_R4	VT_ARRAY | VT_R4 [Range of the number of elemts] “DO?”or“DI?” : 1 to 2048 “HRI?”or“IRI?” :1 to 32768	It can be written/ read as the data of Single Precision floating point number (4-byte). [?(Address)Range] <“DO?”or“DI?”> ・・・”Endian”option :invalid “OffsetAddressZero=False” : 1 to 65505 “OffsetAddressZero=True” : 0 to 65504 <“HRI?”or“IRI?”時> ・・・”Endian”option :valid “OffsetAddressZero=False” : 1 to 65535 “OffsetAddressZero=True” : 0 to 65534
R8	VT_R8	VT_ARRAY | VT_R8 [Range of the number of elemts] “DO?”or“DI?” : 1 to 1024 “HRI?”or“IRI?” : 1 to 16384	It can be written/ read as the data of Double Precision floating point number (8-byte). [?(Address)Range] <“DO?”or“DI?”> ・・・”Endian”option :invalid “OffsetAddressZero=False” : 1 to 65473 “OffsetAddressZero=True” : 0 to 65472 <“HRI?”or“IRI?”>・・・”Endian”option :valid “OffsetAddressZero=False” : 1 to 65533 “OffsetAddressZero=True” : 0 to 65532

2.3.9. CaoExtension: : GetVariableNames property (for client mode only)

In the Client mode, obtain the variable name list written in Table 2-15 .

2.3.10. CaoVariable: : get_Value property

Obtain information corresponding to variables. For about implementation status and data to be obtained, refer to 2.5.

2.3.11. CaoVariable: : put_Value property

Set information corresponding to variables. For about implementation status and data to be set, refer to 2.5.

2.3.12. CaoController::OnMessage event

OnMessage events in the following table will occur.

Table 2-7 CaoController: : OnMessage event

No	Desciption	Function	Availability on each communication mode								Page
			Server				Client				
			Synchronous		Asynchronous		Synchronous		Asynchronous		
			eth	com	eth	com	eth	com	eth	com	
1	REPLY_MSG	Notify the reception of the reply message from the server device.	−	−	−	−	−	−	✓	✓	P. 20
2	QUERY_MSG	Notify the reception of the request (query) message from the client device	−	−	✓	✓	−	−	−	−	P. 21
3	IPINFO_MSG	Notify the connection/disconnection of the TCP client device.	✓	−	✓	−	−	−	−	−	P. 23

REPLY_MSG

Occurrence condition A reply message from a server device is received.

Number 1

Description REPLY_MSG

Source

Destination CaoExtension name or ”*User Variable name*”

Value VT_ARRAY | VT_VARIANT

Array[0]: VT_BSTR

”*Modbus function-compatible Execute command name*”

Array[1]: VT_BSTR

Source IP address Example) ”192.168.0.1”

If a communication device is “com”, this will be blank (””).

Array[2]: VT_I4

Server device address or Unit identifier (Range : 1 to 255)

Array[3]: VT_I4

Function code (Range : 1 to 127)

Array[4]: VT_I4

Function sub code ・・・ This will be 0 if it is not used.

Array[5]: VT_I4

Execution result (0 or 1: Normal, Less than 0: Error code)

Array[6]: VT_ARRAY | VT_**

Same as the return value of each function code.(If there is no return value, VT_EMPTY). For details, see 2.4.4.

Description Notify that the reply message from a server device is received.

An execution result for a request (query) function is stored in the Value property.

Note: The availability of this command differs depending on the communication mode. For details, see Table 2-7.

QUERY_MSG

Occurrence condition A request (query) message is received from a client device.

Number 2

Description QUERY_MSG

Source

Destination

Value VT_ARRAY | VT_VARIANT

Array[0]: VT_BSTR

Source IP address Example) ”192.168.0.1”

Array[1]: VT_I4

Server device address or Unit identifier (Range : 0 to 255)

Array[2]: VT_I4

Modbus protocol Function Code. For details, refer to Table 2-8.

Array[3]: VT_ARRAY|VT_VARIANT

Parameters for respective Modbus protocol Function Code.

For details, refer to Table 2-8.

Table 2-8

Function	Array[2]	Array[3][0]	Array[3][1]	Array[3][2]
DO (Discrete Output) multiple reading	1 (0x01)	VT_I4: Starting address	VT_I4: Bit count to read	VT_EMPTY: none
DI (Discrete Input) multiple reading	2 (0x02)	VT_I4: Starting address	VT_I4: Bit count to read	VT_EMPTY: none
Holding register (16 bits) multiple reading	3 (0x03)	VT_I4: Starting address	VT_I4: Data count to read	VT_EMPTY: none
Input register (16 bits) multiple reading	4 (0x04)	VT_I4: Starting address	VT_I4: Data count to read	VT_EMPTY: none
Read Exception status	7 (0x07)	VT_EMPTY: none		
DO (Discrete Output) multiple writing	15 (0x0F)	VT_I4: Starting address	VT_I4: Bit count to write	VT_ARRAY | VT_BOOL Writing data
Holding register (16 bits) multiple writing	16 (0x10)	VT_I4: Starting address	VT_I4: Data count to write	VT_ARRAY | VT_I2 Writing data

Description Notify that a request (query) message from the client device is received. Table 2-9 shows Modbus protocol Function Code which the reception notification is available.

In general, after finishing the processing for the request function in *Value property*, CAO client needs to send a reply message within the period specified by *SendReplyTimeout* option by using *Message::Reply()* method. Argument of the reply message is same as *CaoController::Execute "SendReply"* command. (see 27.)
If *Message::Reply()* method is executed after the timeout period specified by *SendReplyTimeout* option has passed, an error is returned.

Note: The availability of this command differs depending on the communication mode.
For details, see Table 2-7.

Table 2-9 Reception notification available Modbus function code (For server mode only)

Function	On the Modbus protocol				Notification message (QUERY_MESSAGE) or Reception command (ReceiveQuery)			Auto reply	
	Broadcast		Function Code (HEX)	Sub Code	Reception notification availability	Notification Function Code (HEX)	Notification Sub Code	Normal reply*1	Exception response *2
	TCP	ASCII / RTU							
DO (Discrete Output) multiple reading	N/A	N/A	1(0x01)		✓	1(0x01)			✓
DI (Discrete Input) multiple reading	N/A	N/A	2(0x02)		✓	2(0x02)			✓
Holding register (16 bits) multiple reading	N/A	N/A	3(0x03)		✓	3(0x03)			✓
Input register (16 bits) multiple reading	N/A	N/A	4(0x04)		✓	4(0x04)			✓
DO (Discrete Output) single writing	N/A	✓	5(0x05)		✓	15(0x0F)※6			✓
Holding register (16 bits) single writing	N/A	✓	6(0x06)		✓	16(0x10)※3			✓
Exception status reading	N/A	N/A	7(0x07)		✓	7(0x07)			✓
Diagnostic Echo back of query data	N/A	N/A	8(0x08)	0		—		✓	✓
Others	N/A	N/A		1~		—			✓
DO (Discrete Output) multiple writing	N/A	✓	15(0x0F)		✓	15(0x0F)			✓
Holding register (16 bits) multiple writing	N/A	✓	16(0x10)		✓	16(0x10))			✓

Holding register AND/OR mask writing	N/A	N/A	22(0x16)		✓	Notify twice (3(0x03), and then 16(0x10))※4			✓
Holding register multiple reading and writing	N/A	N/A	23(0x17)		✓	Notify twice (16(0x10), and then 3(0x03))※5			✓
Others			other than above						✓

※1: Send a normal response (reply) message automatically without notification.

※2: Send an exception response (reply) message automatically without notification.

※3: Function code on the Modbus protocol is 6, however, the notification function code is 16. Number of writing data (count) is fixed to 1.

※4: Although the function code on the Modbus protocol is 22, the notification function code is divided into two function codes; function code 3 and 16. Function code 3, which is arrived first, shows the result (value) of AND/OR operation against the reading value. Function code 16, which is arrived next, performs writing. Number of writing data (count) is fixed to 1.

※5: Although the function code on the Modbus protocol is 23(0x17), the notification function code is divided into two function codes; function code 16 and 3.

※6: Although the function code on the Modbus protocol is 5, the notification function code is 10. Number of writing data (count) is fixed to 1.

IPINFO_MSG

Occurrence condition A TCP client device is connected or disconnected.

Number 3

Description IPINFO_MSG

Source

Destination

Value VT_ARRAY | VT_VARIANT

Array[0]: VT_BOOL

Connection status: Connect (VARIANT_TRUE), Disconnect (VARIANT_FALSE)

Array[1]: VT_BSTR

Connected or disconnected IP address Example) ”192.168.0.1”

Array[2]: VT_ARRAY|VT_VARIANT

Just like the system variable of “@IpInfo”, obtain the currently connected client device’s IP address and port number.

For details, see “@IpInfo” System variable (Table 2-14).

Description Inform the connection or disconnection of a TCP client device.

[Condition for connection]

The following two conditions shall be met;

- a TCP client device requests the connection, and,

- the number of currently connected TCP client devices is 16 or less.

[Condition for disconnection]

・Disconnection request is arrived from a TCP client device.

・In “TcpConnectionTimeout” option, a request (query) message is not received within the specified period of time from the TCP client device connection timing when a TCP connection timeout period [ms] is other than “Disable (0)”.

・The reception length of a request (query) message is less than 6 bytes.

・Transmission of a response (reply) message fails.

Note: The availability of this command differs depending on the communication mode. For details, see Table 2-7.

2.4. Command list

2.4.1. CaoController class

Table 2-10 CaoController: : Execute command list

Command name	Function	Availability on each communication mode								page
		Server				Client				
		Synchronous		Asynchronous		Synchronous		Asynchronous		
		eth	com	eth	com	eth	com	eth	com	
ProviderCancel	Set to Cancel state	✓	✓	✓	✓	✓	✓	✓	✓	P. 24
ProviderClear	Clear the Cancel state	✓	✓	✓	✓	✓	✓	✓	✓	P. 25
ReceiveQuery	Receive a request (query) message	✓	✓	—	—	—	—	—	—	P. 25
SendReply	Send a reply message	✓	✓	—	—	—	—	—	—	P. 27

2.4.2. Details of CaoController: : Execute command

ProviderCancel

Available to all communication modes

Syntax *object*. ProviderCancel()

Argument none

Return value none

Description Set a provider to a Cancel state.

Sending and receiving execution are suspended during a Cancel state.

To release the cancel state, execute a “ProviderClear” command.

ProviderClear

Available to all communication modes

Syntax *object*. ProviderClear()

Argument none

Return value none

Description Release a Cancel state of a provider.

ReceiveQuery

Available in the Server mode only

Syntax *object*. ReceiveQuery()

Argument none

Return value <Data> = VT_ARRAY | VT_VARIANT or VT_EMPTY

Array[0]: VT_BSTR

Source IP address Example) ”192.168.0.1”

Array[1]: VT_I4

server device address or unit identifier (Range : 0 to 255)

Array[2]: VT_I4

Modbus protocol Function Code. For details, see Table 2-11.

Array[3]: VT_ARRAY|VT_VARIANT

The parameter differs depending on the applicable Modbus protocol Function Code. For details, refer to Table 2-11.

Table 2-11

Function	Array[2]	Array[3][0]	Array[3][1]	Array[3][2]
DO (Discrete Output) multiple reading	1 (0x01)	VT_I4: Starting address	VT_I4: Bit count to read	VT_EMPTY: none
DI (Discrete Input) multiple reading	2 (0x02)	VT_I4: Starting address	VT_I4: Bit count to read	VT_EMPTY: none
Holding register (16 bits)	3 (0x03)	VT_I4: Starting	VT_I4: Data count to	VT_EMPTY: none

multiple reading		address	read	
Input register (16 bits) multiple reading	4 (0x04)	VT_I4: Starting address	VT_I4: Data count to read	VT_EMPTY: none
Read Exception status	7 (0x07)	VT_EMPTY: none		
DO (Discrete Output) multiple writing	15 (0x0F)	VT_I4: Starting address	VT_I4: Bit count to write	VT_ARRAY | VT_BOOL Writing data
Holding register (16 bits) multiple writing	16 (0x10)	VT_I4: Starting address	VT_I4: Data count to write	VT_ARRAY | VT_I2 Writing data

Description

Receive a request (query) message from a client device.

Table 2-9 shows the list of Modbus protocol Function Code that the reception notification is available.

If the initial byte of a request (query) message is not received within the timeout period specified by ReceiveQueryTimeout option has passed, the processing will return. In this case, <Data> will be VT_EMPTY.

In general, after finishing the processing for the request function in *<Data>*, CAO client needs to send a reply message within the period specified by *SendReplyTimeout* option, by using Send*Reply()* command. For details, see 27.

Note: The availability of this command differs depending on the communication mode. For details, see Table 2-10.

SendReply

Available in the Server mode only.

Syntax

`object. SendReply( <Data> )`

Argument

<Data> = VT_ARRAY | VT_VARIANT

Array[0]: VT_I4

Notification Modbus FunctionCode (see Table 2-11)

Array[1]: VT_BOOL

Execution result (VARIANT_TRUE: Normal, VARIANT_FALSE: Error)

Array[2]: If an execution result is normal, a data type on the table below will be decided based on the Notification Modbus Function Code. If the execution results abnormal, set VT_EMPTY.

Table 2-12

Function	Notification Function Code (HEX)	Data type	Description
DO (Discrete Output) multiple reading	1(0x01)	VT_ARRAY | VT_BOOL	Reading data
DI (Discrete Input) multiple reading	2(0x02)	VT_ARRAY | VT_BOOL	Reading data
Holding register (16 bits) multiple reading	3(0x03)	VT_ARRAY | VT_I2	Reading data
Input register (16 bits) multiple reading	4(0x04)	VT_ARRAY | VT_I2	Reading data
Read Exception status	7(0x07)	VT_UI1	Exception status
DO (Discrete Output) multiple writing	15(0x0F)	VT_EMPTY	No data
Holding register (16 bits) multiple writing	16(0x10)	VT_EMPTY	No data

Return value

none

Description

Send a reply messge to a client device.

In general, after finishing the processing for the request function received by *ReceiveQuery()* command, CAO client needs to send a reply message within the period specified by *SendReplyTimeout* option by using this command.

If this command is executed after the timeout period specified by *SendReplyTimeout* option has passed, an error is returned. .

When the communication device is com, if the server device address received by ReceiveQuery() command is broadcast(0), you do not need to execute this command. The reply message will not be sent to a client device, even if this command is executed.

Also, if processing result (Array[1]) is set to abnormal (VARIANT_FALSE), the reply message will not be sent.

Note: The availability of this command differs depending on the communication mode. For details, see Table 2-10.

2.4.3. CaoExtension class (for client mode only)

The following table shows the Modbus function compatible command at the client mode.

Table 2-13 CaoExtension: : Execute Modbus function compatible command list (for client mode only)

Command name (Old names of Modbus provider compatible commands)	Modbus protocol				Function	Page
	Broadcast		Function Code (HEX)	Sub Code		
	TCP	ASCII / RTU				
ReadMultipleDiscreteOutputs (ReadCoilStatus)	N/A	N/A	1(0x01)		DO (Discrete Output) multiple reading	P. 29
ReadMultipleDiscreteInputs (ReadInputStatus)	N/A	N/A	2(0x02)		DI (Discrete Input) multiple reading	P. 29
ReadMultipleHoldingRegisters (ReadHoldingRegister)	N/A	N/A	3(0x03)		Holding register (16 bits) multiple reading	P. 30
ReadMultipleHoldingRegistersLongInt (none)					Holding register (32 bits Integer type) multiple reading	P. 30
ReadMultipleHoldingRegistersFloat (none)					Holding register (32 bits Floating point type) multiple reading	P. 30
ReadMultipleInputRegisters (ReadInputRegister)	N/A	N/A	4(0x04)		Input register (16 bits) multiple reading	P. 31
RreadMultipleInputRegistersLongInt (none)					Input register (32 bits Integer type) multiple reading	P. 31
RreadMultipleInputRegistersFloat (none)					Input register (32 bits Floating point type) multiple reading	P. 32
WriteSingleDiscreteOutput (ForceSingleCoil)	N/A	✓	5(0x05)		DO (Discrete Output) single writing	P. 32
WriteSingleHoldingRegister (PresetSingleRegister)	N/A	✓	6(0x06)		Holding register (16 bits) single writing	P. 32
ReadExceptionStatus (ReadExceptionStatus)	N/A	N/A	7(0x07)		Read Exception status	P. 33
DiagnosticsReturnQueryData (DiagnosticsReturnQueryData)	N/A	N/A	8(0x08)	0	Diagnostic :Echo back of query data	P. 33
DiagnosticsRestartCommunicationsOption (DiagnosticsRestartCommunicationsOption)				1	Diagnostic : Initialization of communication port	P. 34
WriteMultipleDiscreteOutputs (ForceMultipleCoils)	N/A	✓	15(0x0F)		DO (Discrete Output) multiple writing	P. 34
WriteMultipleHoldingRegisters (PresetMultipleRegisters)	N/A	✓	16(0x10)		Holding register (16 bits) multiple writing	P. 34
WriteMultipleHoldingRegistersLongInt (none)					Holding register (32 bits Integer type) multiple writing	P. 35
WriteMultipleHoldingRegistersFloat (none)					Holding register (32 bits Floating point type) multiple writing	P. 35
MaskWriteHoldingRegister (MaskWrite4XRegister)	N/A	N/A	22(0x16)		Write the AND mask/OR mask result of bits of a Holding register	P. 36
ReadWriteMultipleHoldingRegisters	N/A	N/A	23(0x17)		Holding register multiple reading and	P.

(ReadWrite4XRegisters)					writing	36
AnotherFunctionCode (none)	N/A	Depend on a Function Code	1-127 (0x01-0x7F)		Other function codes	P. 37

2.4.4. Details of CaoExtension: : Execute Modbus function compatible command (for client mode only)

ReadMultipleDiscreteOutputs

Function Code	Broad cast
1(0x01)	-

Syntax *object*. ReadMultipleDiscreteOutputs(<Address>, <Count>)

Argument <Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default)

When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Bit count to read Range : 1 to 2000

Return value <Data> = VT_ARRAY | VT_BOOL: Reading data

VARIANT_TRUE: DO (Discrete Output) is ON-state

VARIANT_FALSE: DO (Discrete Output) is OFF-state

Description Starting from the starting address, read the ON/OFF-state of **DO (Discrete Output)**.

ReadMultipleDiscreteInputs

Function Code	Broad cast
2(0x02)	-

Syntax *object*. ReadMultipleDiscreteInputs(<Address>, <Count>)

Argument <Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Bit count to read Range : 1 to 2000

Return value <Data> = VT_ARRAY | VT_BOOL: Reading data

VARIANT_TRUE: DI (Discrete Input) is ON

VARIANT_FALSE: DI (Discrete Input) is OFF

Description Starting from the starting address, read the ON/OFF-state of **DI (Discrete Input)**.

ReadMultipleHoldingRegisters

Function Code	Broad cast
3(0x03)	-

Syntax `object. ReadMultipleHoldingRegisters( <Address>, <Count> )`

Argument

<Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).
When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to read Range : 1 to 125

Return value <Data> = VT_ARRAY | VT_UI2: Reading data

Description Starting from the starting address, read the contents of a sequence of **Holding register (16 bits)**.

ReadMultipleHoldingRegistersLongInt

Function Code	Broad cast
3(0x03)	-

Syntax `object. ReadMultipleHoldingRegistersLongInt( <Address>, <Count> )`

Argument

<Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).
When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to read Range : 1 to 62

Return value <Data> = VT_ARRAY | VT_I4: Reading data

Description Starting from the starting address, read the contents of a sequence of **Holding register (32 bits Integer type)**.

ReadMultipleHoldingRegistersFloat

Function Code	Broad cast
3(0x03)	-

Syntax `object. ReadMultipleHoldingRegistersFloat( <Address>, <Count> )`

Argument

<Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).
When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to read Range : 1 to 62

Return value <Data> = VT_ARRAY | VT_R4: Reading data

Description Starting from the starting address, read the contents of a sequence of **Holding register (32 bits Floating point type)**.

ReadMultipleInputRegisters

Function Code	Broadcast
4(0x04)	-

Syntax `object. ReadMultipleInputRegisters( <Address>,  <Count> )`

Argument <Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to read (Range : 1 to 125)

Return value <Data> = VT_ARRAY | VT_UI2: Reading data

Description Starting from the starting address, read the contents of a sequence of **Input register (16 bits)**.

ReadMultipleInputRegistersLongInt

Function Code	Broadcast
4(0x04)	-

Syntax `object. ReadMultipleInputRegistersLongInt( <Address>,  <Count> )`

Argument <Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data coutn to read (Range : 1 to 62)

Return value <Data> = VT_ARRAY | VT_I4: Reading data

Description Starting from the starting address, read the contents of a sequence of **Input register (32 bits Integer type)**.

ReadMultipleInputRegistersFloat

Function Code	Broadcast
4(0x04)	-

Syntax *object*. ReadMultipleInputRegistersFloat(<Address>, <Count>)

Argument

<Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to read (Range : 1 to 62)

Return value

<Data> = VT_ARRAY | VT_R4: Reading data

Description

Starting from the starting address, read the contents of a sequence of **Input register (32 bits Floating point type)**.

WriteSingleDiscreteOutput

Function Code	Broadcast	
	eth	com
5(0x05)	-	✓

Syntax *object*. WriteSingleDiscreteOutput(<Address>, <Data>)

Argument

<Address> = VT_I4: Address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Data> = VT_BOOL: Writing data

ON: VARIANT_TRUE, OFF: VARIANT_FALSE

Return value

none

Description

Update the content of **DO (Discrete Output)** of the specified address.

WriteSingleHoldingRegister

Function Code	Broadcast	
	eth	com
6(0x06)	-	✓

Syntax *object*. WriteSingleHoldingRegister(<Address>, <Data>)

Argument

<Address> = VT_I4: Address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Data> = VT_UI2: Writing data

Return value none

Description Update the content of **Holding register (16 bits)** of the specified address.

ReadExceptionStatus

Function Code	Broad cast
7(0x07)	-

Syntax *object*. ReadExceptionStatus( )

Argument none

Return value <Data> = VT_UI1: Exception status

Description Read the state of the exception status.

The exception status obtained are assigned from the lowest bit per bit.

DiagnosticsReturnQueryData

Function Code-Sub	Broad cast
8(0x08)-0	-

Syntax *object*. DiagnosticsReturnQueryData(<Count>, <Query Data>)

Argument <Count> = VT_I4: Number of query data (Range : 1 to 250)

<Query Data > = ARRAY|VT_UI1: Query data

Return value < Echo Data > = ARRAY|VT_UI1: Echo data

Description Diagnose a communication line by responding query data, which has been sent from the client, as an echo back with the server as-is.

If the diagnosis completes successfully, the value of <Echo Data> of Return value will be the same as the value of <Query Data> that has been specified by Argument.

DiagnosticsRestartCommunicationsOption

Function Code-Sub	Broad cast
8(0x08)-1	-

Syntax `object. DiagnosticsRestartCommunicationsOption( <ClearEventLog> )`

Argument

<ClearEventLog> = VT_BOOL: Specify whether if an event log is cleared.

Clear an event log: VARIANT_TRUE

Keep an event log: VARIANT_FALSE

Return value none

Description Initialize the communication port on the server side, and specify whether if an event log is cleared or kept.

WriteMultipleDiscreteOutputs

Function Code	Broadcast	
	eth	com
15(0x0F)	-	✓

Syntax `object. WriteMultipleDiscreteOutputs( <Address>, <Count>, <Data> )`

Argument

<Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Bit count to write (Range : 1 to 1968)

<Data> = VT_ARRAY | VT_BOOL: ON/OFF data

ON: VARIANT_TRUE, OFF: VARIANT_FALSE

Return value none

Description Starting from the starting address, update the contents of a sequence of **DO (Discrete Output)**.

WriteMultipleHoldingRegisters

Function Code	Broadcast	
	eth	com
16(0x10)	-	✓

Syntax `object. WriteMultipleHoldingRegisters( <Address>, <Data> )`

Argument

<Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to write (Range : 1 to 123)

<Data> = VT_ARRAY | VT_UI2: Writing data

Return value none

Description Starting from the starting address, update the contents of a sequence of **Holding register (16 bits)**.

WriteMultipleHoldingRegistersLongInt

Function Code	Broadcast	
	eth	com
16(0x10)	-	✓

Syntax `object. WriteMultipleHoldingRegistersLongInt( <Address>,  <Data> )`

Argument <Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).
When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to write (Range : 1 to 61)

<Data> = VT_ARRAY | VT_I4: Writing data

Return value none

Description Starting from the starting address, update the contents of a sequence of **Holding register (32 bits Integer type)**.

WriteMultipleHoldingRegistersFloat

Function Code	Broadcast	
	eth	com
16(0x10)	-	✓

Syntax `object.  WriteMultipleHoldingRegistersFloat( <Address>,  <Count>, <Data> )`

Argument <Address> = VT_I4: Starting address

Range : When OffsetAddressZero is False, 1 to 65536 (default).
When OffsetAddressZero is True, 0 to 65535.

<Count> = VT_I4: Data count to write (Range : 1 to 61)

<Data> = VT_ARRAY | VT_R4: Writing data

Return none

value

Description Starting from the starting address, update the contents of a sequence of **Holding register (32bits Floating point type)**.

MaskWriteHoldingRegister

Function Code	Broad cast
22(0x16)	-

Syntax *object*. MaskWriteHoldingRegister(<Address>, <AND_Mask>, <OR_Mask>)

Argument <Address> = VT_I4: address

Range : When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<AND_Mask> = VT_UI2: AND masking bit

<OR_Mask> = VT_UI2: OR masking bit

Return value none

Description Modify the **Holding register (16 bits)** of a specified address using a combination of an AND mask, an OR mask, and the register's current contents.

(Reference) The value to update will be processed (calculated) on the server device side based on the following formula.

[Update value]=([current value]AND<AND_Mask>) OR (<OR_Mask> AND <NOT AND_Mask>)

ReadWriteMultipleHoldingRegisters

Function Code	Broad cast
23(0x17)	-

Syntax *object*. ReadWriteMultipleHoldingRegisters(<ReadAddress>, <ReadCount>, <WriteAddress>, <WriteCount>, <WriteData>)

Argument <ReadAddress> = VT_I4: Starting address of the reading

Range: When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<ReadCount> = VT_I4: Data count to read (Range : 1 to 125)

<WriteAddress> = VT_I4: Starting address of the writing

Range: When OffsetAddressZero is False, 1 to 65536 (default).

When OffsetAddressZero is True, 0 to 65535.

<WriteCount> = VT_I4: Data count to write (Range : 1 to 121)

<WriteData> = VT_ARRAY | VT_UI2: Writing data

Return value <Data> = VT_ARRAY | VT_UI2: Reading data

Description Perform the writing and reading for the **Holding register (16 bits)**.
Write <WriteCount> number of <WriteData> into the address specified by <WriteAddress>, and then, read <ReadCount> number of data from the address specified by <ReadAddress>.

AnotherFunctionCode

Function Code	Broadcast	
	eth	com
1-127 (0x01-0x7F)	-	Depend on a Function Code

Syntax *object*. AnotherFunctionCode(<FunctionCode>, <RequestCount>, <RequestData>)

Argument <FunctionCode> = VT_I4: Function code (Range : 1 to 127)
<RequestCount> = VT_I4: Data count of sent data on the data field (Range : 0 to 252)
<RequestData> = VT_ARRAY | VT_UI1: Content of sent data on the data field
<ResponseCount> = VT_I4: Data count of response data on the data field (Range : 0 to 252)

Return value <ResponseData> = VT_ARRAY | VT_UI1: Response data on the data field

Description Execute Modbus function codes that have not been defined as CaoController::Execute command.
For <RequestData>, specify <RequestCount> number of binary data that correspond to the data field of the Modbus query message. The contents of data field depends on the <FunctionCode> (Error check code is automatically added.).
The <ResponseCount> number of binary data that corresponds to the data field of the Modbus reply message will be stored in <ResponseData>. (Error check code is automatically deleted.).
[Note]
1) When com is selected, if the device address is Broadcast address (UnitAddress=0), there is no response data from the server device. Therefore, <ResponseCount> will be ignored and the<ResponseData> will be VT_EMPTY.
2) If 0 is set to <ResponseCount>, <ResponseData> will be VT_EMPTY.
3) Other than above 1) and 2), if <ResponseCount> <> Actual data count of <ResponseData>, the execution result will differ depending on the communication devices as follows.
If com is selected, it will end with an error.
If eth is selected, the execution will end normally, but the return value is

<ResponseCount> <> Actual data count of <ResponseData>

2.5. Variable list

2.5.1. CaoController class

Table 2-14 CaoController class System variable list

Variable name	Data type	Description	Attribute	
			get	put
@Version	VT_BSTR	Version information	✓	—
@Error	VT_I4	Obtain an error code that has occurred last.	✓	—
@IpInfo	VT_ARRAY| VT_VARIANT or VT_EMPTY	Obtain an IP address and port number of a currently connected client device. Array[0]: VT_I4. Number of client connected (0 to 16) Array[1]: If the first client device is connected, VT_ARRAY|VT_VARIANT. If the first client device is not connected, VT_EMPTY. Array[1][0]: VT_BSTR. IP address of the first client device. Array[1][1]: VT_I4. Connection port number of the first client device. ・ ・ ・ Array[17]: If the 16th client device is connected, VT_ARRAY|VT_VARIANT. If the 16th client device is not connected, VT_EMPTY Array[17][0]: VT_BSTR IP address of the 16th client device. Array[17][1]: VT_I4 Connection port number of the 16th client device. Note: When the client mode or serial mode is selected, this will be VT_EMPTY.	✓	—

2.5.2. CaoExtension class (for client mode only)

The following table shows Modbus function compatible user variables at the client mode.

Table 2-15 CaoExtension class Modbus function compatible user variable list

<table>
<tr><th rowspan="2">Variable name</th><th rowspan="2">Data type</th><th rowspan="2">Description</th><th colspan="2">Attribute</th><th rowspan="2">Remark</th></tr>
<tr><th>get</th><th>put</th></tr>
<tr><td>DO?※1</td><td>VT_I4※2</td><td><b>Set/Get</b> the state (value) of <b>DO (Discrete Output)</b> of the number “?”. For a question mark (?), which is placed on the end of a variable name, specify a logical number. The logical number stands for an address.
Example) “DO1”
- If the offset address (“OffsetAddressZero=False”) is 1 or larger, “DO1” corresponds to the DO address “0” on the server device side.
- If the offset address (“OffsetAddressZero=True”) is 0 or larger, “DO1” corresnponds to the DO address “1” on the server device side.

・As the following table shows, the range of the logical number is determined by the combination of the Offset address (“OffsetAddressZero”) option which is a connection parameter at AddController and the Data width (“UserVarWidth”) option at AddController.
<table>
<tr><th rowspan="2">Offset address (OffsetAddressZero)</th><th colspan="4">Data width (UserVarWidth)</th></tr>
<tr><th>1</th><th>8</th><th>16</th><th>32</th></tr>
<tr><td>True</td><td>0 to 65535</td><td>0 to 65528</td><td>0 to 65520</td><td>0 to 65504</td></tr>
<tr><td>False (default)</td><td>1 to 65536</td><td>1 to 65529</td><td>1 to 65521</td><td>1 to 65505</td></tr>
</table>

・As the following table shows, the range of setting value/getting value is determined by the Data width (“UserVarWidth”) option.
If the Data width is other than 1, the data notation will be MSB (Most Significant Bit).
<table>
<tr><th rowspan="2"></th><th colspan="4">Data width (UserVarWidth)</th></tr>
<tr><th>1</th><th>8</th><th>16</th><th>32</th></tr>
<tr><td>Range of setting/getting values</td><td>0 to 1</td><td>0 to 255</td><td>0 to 65535</td><td>-2147483648 to +2147483647</td></tr>
</table>

(Reference)
As the following table shows, a function code that is sent/received on the Modbus communication protocol is determined by the combination of the Get/Set and the Data width (“UserVarWidth”) option.
<table>
<tr><th rowspan="2"></th><th colspan="4">Data width (UserVarWidth)</th></tr>
<tr><th>1</th><th>8</th><th>16</th><th>32</th></tr>
<tr><td>Get (get)</td><td>1(0x01)</td><td>1(0x01)</td><td>1(0x01)</td><td>1(0x01)</td></tr>
<tr><td>Set (put)</td><td>5(0x05)</td><td>15(0x0F)</td><td>15(0x0F)</td><td>15(0x0F)</td></tr>
</table>
</td><td>✓</td><td>✓</td><td></td></tr>
</table>

<table>
<tr>
<td>DI?※1</td>
<td>VT_I4※2</td>
<td>
Set/Get the state (value) of D<b>I (Discrete Input)</b> of the number “?”.
For a question mark (?), which is placed on the end of a variable name, specify a logical number. The logical number stands for an address.

Example) “DI1”
- If the offset address (”OffsetAddressZero=False”) is 1 or larger,
“DI1” corresponds to the <b>DI</b> address “0” on the server device side.
- If the offset address (”OffsetAddressZero=True”) is 0 or larger,
“DI1” corresponds to the <b>DI</b> address “1” on the server device side.

・As the following table shows, the range of the logical number is determined by the combination of the Offset address (“OffsetAddressZero”) option which is a connection parameter at AddController and the Data width (“UserVarWidth”) option at AddController.
<table>
<tr><th rowspan="2">Offset address
(OffsetAddressZero)</th><th colspan="4">Data width (UserVarWidth)</th></tr>
<tr><th>1</th><th>8</th><th>16</th><th>32</th></tr>
<tr><td>True</td><td>0 to 65535</td><td>0 to 65528</td><td>0 to 65520</td><td>0 to 65504</td></tr>
<tr><td>False
(default)</td><td>1 to 65536</td><td>1 to 65529</td><td>1 to 65521</td><td>1 to 65505</td></tr>
</table>

・As the following table shows, the range of setting/getting value is determined by the Data width (“UserVarWidth”) option.
If the Data width is other than 1, the data notation will be MSB (Most Significant Bit).
<table>
<tr><th rowspan="2"></th><th colspan="4">Data width (UserVarWidth)</th></tr>
<tr><th>1</th><th>8</th><th>16</th><th>32</th></tr>
<tr><td>Range of setting/getting values</td><td>0 to 1</td><td>0 to 255</td><td>0 to 65535</td><td>-2147483648 to +2147483647</td></tr>
</table>

(Reference)
As the following table shows, a function code that is sent/received on the Modbus communication protocol is determined by the combination of the Get/Set and the Data width (“UserVarWidth”) option.
<table>
<tr><th rowspan="2"></th><th colspan="4">Data width (UserVarWidth)</th></tr>
<tr><th>1</th><th>8</th><th>16</th><th>32</th></tr>
<tr><td>Get (get)</td><td>2(0x02)</td><td>2(0x02)</td><td>2(0x02)</td><td>2(0x02)</td></tr>
<tr><td>Set (put)</td><td>—</td><td>—</td><td>—</td><td>—</td></tr>
</table>
</td>
<td>✓</td>
<td>-</td>
<td></td>
</tr>
</table>

<table>
<tr>
<td>HRI?※1</td>
<td>VT_I4※2</td>
<td>
<p><b>Set/Get</b> the state (value) of the <b>Holding register</b> of the number "?" by <b>Integer type</b>. For a question mark (?), which is placed on the end of a variable name, specify a logical number. The logical number stands for a register address.</p>
<p>Example) "HRI 1"
- If the offset address ("OffsetAddressZero=False") is 1 or larger, "HRI1" corresponds to the Holding register address "0" on the server device side.
- If the offset address ("OffsetAddressZero=True") is 0 or larger, "HRI1" corresponds to the Holding register address "1" on the server device side.</p>
<p>・As the following table shows, the range of the logical number is determined by the combination of the Offset address ("OffsetAddressZero") option which is a connection parameter at AddController and the Data width ("UserVarWidth") option at AddController.</p>
<table>
<tr><th rowspan="2">Offset address
(OffsetAddressZero)</th><th colspan="2">Data width (UserVarWidth)</th></tr>
<tr><th>16</th><th>32</th></tr>
<tr><td>True</td><td>0 to 65535</td><td>0 to 65534</td></tr>
<tr><td>False (default)</td><td>1 to 65536</td><td>1 to 65535</td></tr>
</table>
<p>・As the following table shows, the range of setting/getting value is determined by the Data width ("UserVarWidth") option.</p>
<table>
<tr><th rowspan="2"></th><th colspan="2">Data width (UserVarWidth)</th></tr>
<tr><th>16</th><th>32</th></tr>
<tr><td>Range of setting/getting values</td><td>0 to 65535</td><td>-2147483648 to +2147483647</td></tr>
</table>
<p>(Reference)
As the following table shows, a function code that is sent/received on the Modbus communication protocol is determined by the combination of the Get/Set and the Data width ("UserVarWidth") option.</p>
<table>
<tr><th rowspan="2"></th><th colspan="2">Data width (UserVarWidth)</th></tr>
<tr><th>16</th><th>32</th></tr>
<tr><td>Get (get)</td><td>3(0x03)</td><td>3(0x03)</td></tr>
<tr><td>Set (put)</td><td>6(0x06)</td><td>16(0x10)</td></tr>
</table>
</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</table>

<table>
<tr>
<td>HRF?</td>
<td>VT_R4</td>
<td><b>Set/Get</b> the state (value) of the <b>Holding register</b> of the number “?” by <b>32 bits Floating point type</b>. For a question mark (?), which is placed on the end of a variable name, specify a logical number. The logical number stands for a register address.

Example) “HRF 1”
- If the offset address (“OffsetAddressZero=False”) is 1 or larger, “HRF1” corresponds to the Holding register address “0” on the server device side.
- If the offset address (“OffsetAddressZero=True”) is 0 or larger, “HRF1” corresponds to the Holding register address “1” on the server device side.

・As the following table shows, the range of the logical number is determined by the combination of the Offset address (“OffsetAddressZero”) option which is a connection parameter at AddController and the Data width (“UserVarWidth”) option at AddController.
<table>
<tr><td>Offset address (OffsetAddressZero)</td><td>Data width (UserVarWidth)</td></tr>
<tr><td></td><td>32 (fixed)</td></tr>
<tr><td>True</td><td>0 to 65534</td></tr>
<tr><td>False (default)</td><td>1 to 65535</td></tr>
</table>

(Reference)
As the following table shows, a function code that is sent/received on the Modbus communication protocol is determined by the combination of the Get/Set and the Data width (“UserVarWidth”) option.
<table>
<tr><td></td><td>Data width (UserVarWidth)</td></tr>
<tr><td></td><td>32 (fixed)</td></tr>
<tr><td>Get (get)</td><td>3(0x03)</td></tr>
<tr><td>Set (put)</td><td>16(0x10)</td></tr>
</table>
</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</table>

<table>
<tr>
<td>IRI?※1</td>
<td>VT_I4※2</td>
<td>
<b>Get</b> the value of <b>Input register</b> of the number “?” by Integer type. For a question mark (?), which is placed on the end of a variable name, specify a logical number.　The logical number stands for an address.

Example)　“IRI 1”
- If the offset address (“OffsetAddressZero=False”) is 1 or larger, “IRI1” corresponds to the <b>Input register</b> address “0” on the server device side.
- If the offset address (“OffsetAddressZero=True”) is 0 or larger, “IRI1” corresponds to the <b>Input register</b> address “1” on the server device side.

・As the following table shows, the range of the logical number is determined by the combination of the Offset address (“OffsetAddressZero”) option which is a connection parameter at AddController and the Data width (“UserVarWidth”) option at AddController.
<table>
<tr><th rowspan="2">Offset address
(OffsetAddressZero)</th><th colspan="2">Data width (UserVarWidth)</th></tr>
<tr><th>16</th><th>32</th></tr>
<tr><td>True</td><td>0 to 65535</td><td>0 to 65534</td></tr>
<tr><td>False (default)</td><td>1 to 65536</td><td>1 to 65535</td></tr>
</table>

・As the following table shows, the range of setting value/getting value is determined by the Data width (“UserVarWidth”) option.
<table>
<tr><th rowspan="2"></th><th colspan="2">Data width (UserVarWidth)</th></tr>
<tr><th>16</th><th>32</th></tr>
<tr><td>Range of setting/getting values</td><td>0 to 65535</td><td>-2147483648 to +2147483647</td></tr>
</table>

(Reference)
As the following table shows, a function code that is sent/received on the Modbus communication protocol is determined by the combination of the Get/Set and the Data width (“UserVarWidth”) option.
<table>
<tr><th rowspan="2"></th><th colspan="2">Data width (UserVarWidth)</th></tr>
<tr><th>16</th><th>32</th></tr>
<tr><td>Get (get)</td><td>4(0x04)</td><td>4(0x04)</td></tr>
<tr><td>Set (put)</td><td>—</td><td>—</td></tr>
</table>
</td>
<td>✓</td>
<td>-</td>
<td></td>
</tr>
</table>

<table>
<tr>
<td>IRF?</td>
<td>VT_R4</td>
<td>Get the value of <b>Input register</b> of the number “?” by <b>32 bits Floating point type</b>. For a question mark (?), which is placed on the end of a variable name, specify a logical number. The logical number stands for an address.

Example) “IRF 1”
- If the offset address (“OffsetAddressZero=False”) is 1 or larger, “IRF1” corresponds to the <b>Input register</b> address “0” on the server device side.
- If the offset address (“OffsetAddressZero=True”) is 0 or larger, “IRF1” corresponds to the <b>Input register</b> address “1” on the server device side.

・As the following table shows, the range of the logical number is determined by the combination of the Offset address (“OffsetAddressZero”) option which is a connection parameter at AddController and the Data width (“UserVarWidth”) option at AddController.

<table>
<tr><td rowspan="2">Offset address (OffsetAddressZero)</td><td>Data width (UserVarWidth)</td></tr>
<tr><td>32 (fixed)</td></tr>
<tr><td>True</td><td>0 to 65534</td></tr>
<tr><td>False (default)</td><td>1 to 65535</td></tr>
</table>

(Reference)
As the following table shows, a function code that is sent/received on the Modbus communication protocol is determined by the combination of the Get/Set and the Data width (“UserVarWidth”) option.

<table>
<tr><td rowspan="2"></td><td>Data width (UserVarWidth)</td></tr>
<tr><td>32 (fixed)</td></tr>
<tr><td>Get (get)</td><td>4(0x04)</td></tr>
<tr><td>Set (put)</td><td>—</td></tr>
</table>
</td>
<td>✓</td>
<td>-</td>
<td></td>
</tr>
</table>

※1: When “VT” option is valid, the range of logical number varies according to Variable types. For details, see Table 2-6.

※2: When “VT” or “Elem” option is valid, varies according to Variable type. For details, see Table 2-5, Table 2-6.

2.6. Error code

Modbus. X provider defines the following original error codes.

Table 2-16 Original error code list

Error name	Code	Description
E_CAOP_SYNC_ONLY	0x80100001	Excutable only in Synchronous mode.
E_CAOP_ASYNC_ONLY	0x80100002	Excutable only in Asynchronous mode.
E_CAOP_CLIENT_ONLY	0x80100003	Excutable only in Client mode.
E_CAOP_SERVER_ONLY	0x80100004	Excutable only in Server mode.
E_CAOP_SERVER_SEND_REPLY_TIMEOUT	0x80100005	This error is issuend under the server mode once a request (query) message has been received and if a reqply is sent after the predetermined period of time specified by "SendReplyTimeout" option.
E_CAOP_ILLEGAL_ARGUMENT	0x80100F01	Argument error. Argument handed to a command is invalid or out of the range.
E_CAOP_ILLEGAL_STATE	0x80100F02	Status error. A function is invoked by abnormal state. If the protocol has not been opened normally, this return code will be returned by all functions.
E_CAOP_ILLEGAL_SLAVE_ADDRESS	0x80100F05	Invalid server device address. Address 0 is used by a function that does not support broadcasting.
E_CAOP_OPEN	0x80100F42	Port open error or socket open error. Failed to open TCP/IP socket or serial port. If the error is serial port open error, the specified serial

		port may not exist in the system.
E_CAOP_FTALK_PORT_ALREADY_OPEN	0x80100F43	Serial port has already opened. A serial port designated for open operation has already been taken by other application.
E_CAOP_FTALK_TCPIP_CONNECT	0x80100F44	TCP/IP connection error. Failed to establish TCP/IP connection. This error occurs when a host exists on the network or on IP address, or the name of host is incorrect. Remote host needs to listen appropriate port number.
E_CAOP_CONNECTION_WAS_CLOSED	0x80100F45	Remote pier closed the TCP/IP connection. This error notifies that TCP/IP connection was closed or damaged by a remote pier.
E_CAOP_SOCKET_LIB	0x80100F46	Socket library error. Failed to load TCP/IP socket library (such as WINSOCK). DLL may not be found or may not be installed.
E_CAOP_PORT_ALREADY_BOUND	0x80100F47	TCP port has already been bound. This error notifies that the specified TCP port cannot be bound. A port may already been taken by other application or may not been released by TCP/IP stack for reuse.
E_CAOP_LISTEN_FAILED	0x80100F48	Failed to listen. Failed to listen the specified TCP port.
E_CAOP_FILEDES_EXCEEDED	0x80100F49	File descriptor exceeds the available range. File descriptor exceeds the maximum limit.
E_CAOP_PORT_NO_ACCESS	0x80100F4A	There is no permission to access the serial port or TCP port. For a serial port error, change the access permission. If this cause of the error is

		TCP/IP, the TCP port number is out of the IPPORT_RESERVED range.
E_CAOP_PORT_NOT_AVAIL	0x80100F4B	TCP port is not available. The specified TCP port is not available in this operation environment.
E_CAOP_LINE_BUSY	0x80100F4C	Serial line is busy. Serial line receives any noise or other signals although it should not have any traffic.
E_CAOP_CHECKSUM	0x80100F81	Checksum error. Checksum of received frame is invalid.
E_CAOP_INVALID_FRAME	0x80100F82	Invalid frame error. The received frame does not correspond to any structure or content in communication protocol or does not match with the frame of query that has been sent before.
E_CAOP_INVALID_REPLY	0x80100F83	Invalid reply error. The received reply frame does not correspond to the communication protocol.
E_CAOP_REPLY_TIMEOUT	0x80100F84	Timeout error. This error may occur when a server device does not respond within the specified time or does not respond completely. Incorrect server device address may cause this error.
E_CAOP_SEND_TIMEOUT	0x80100F85	Send timeout error. This error notifies that the data transmission has been time-out. This error may occur when the handshake line is not configured properly.
E_CAOP_INVALID_MBAP_ID	0x80100F86	Invalid identifier. Protocol or transaction identifier is invalid. TCP server device needs to return the identifier received from TCP client.
E_CAOP_MBUS_EXCEPTION_RESPONSE	0x80100FA0	ModbusException response This error notifies that a message has been received.

E_CAOP_MBUS_ILLEGAL_FUNCTION_RESPONSE	0x80100FA1	This error notifies that an exception response (code 01) of Modbus invalid function is received.
E_CAOP_MBUS_MBUS_ILLEGAL_ADDRESS_RESPONSE	0x80100FA2	This error notifies that an exception response (code 02) of Modbus invalid data address is received.
E_CAOP_MBUS_ILLEGAL_VALUE_RESPONSE	0x80100FA3	This error notifies that an exception response (code 03) of Modbus invalid value is received.
E_CAOP__MBUS_SLAVE_FAILURE_RESPONSE	0x80100FA4	This error notifies that an exception reponse (code 04) of Modbus slave failure is received.

・When a system error of Windows occurs, the error number that is masked by "0x801000" will be returned.

Example) System error of Windows: 2(0x002) >> Error of CAO API : 0x80100002

For about ORiN2 common errors, refer to the error code section on "ORiN2 Programming Guide".

3. Sample program

3.1. Client mode

For the condition of this sample program, communication is established by RS232C/RS485 device: COM1.

This sample program will;

- define a user variable “DO1” for the device of “Server device address = 1”, and then output (set) the ON/OFF-state into Address 1 of DO (DiscreteOutput), and,
- define a user variable “DI1” with 8 bit-width for the device of “Server device address = 2”, and then obtain the status of the DI (DiscreteInput) address 1 to 8 as a byte data type in the MSB (Most Significant Bit).

List 3-1 **Sample31. frm**

```
Private caoEng As CaoEngine
Private caoCntl As CaoController
Private caoExt As CaoExtension
Private caoVarS1DO1 As CaoVariable
Private caoVarS2DI1 As CaoVariable

Private Sub Form_Load()

    Set caoEng = New CaoEngine
    Set caoCntl = caoEng. Workspaces(0). AddController(″″,  ″CaoProv. Modbus. X″,  ″″,
″Conn=COM1″)
    Set caoExt1 = caoCntl. AddExtension(″MbSlave1″)
    Set caoExt2 = caoCntl. AddExtension(″MbSlave2″,  ″UnitAddress=2” )

    Set caoVarS1DO1 = caoExt1. AddVariable(″DO1″, ″″)
    Set caoVarS2DI1 = caoExt2. AddVariable(″DI1″, ” UserVarWidth=8″)

End Sub

Private Sub cmdS1DO1 _ON_Click ()

    caoVarS1DO1. Value = True

End Sub

Private Sub cmdS1DO1_OFF_Click ()

    caoVarS1DO1. Value = False

End Sub

Private Sub cmdS2DI1_In_Click ()

    Ret = caoVarS2DI1. Value

    Text1. Text = Ret

End Sub
```

3.2. Server mode

3.2.1. Sample for Synchronous mode

For the condition of this sample program, the communication is established as Server mode, Synchronous mode, TCP communication mode, and with the IP address “192.168.0.1”.

This sample program will:

: with CaoController:: Execute”ReceiveQuery” command, receive a request (query) message from the client device by using Timer event (Timer 1) by a 100ms interval, and

: reply the result with “SendReply” command.

List 3-2-1 **Sample321. frm**

```
Private m_caoEng As CaoEngine
Private m_caoCntl As CaoController

' Modbus memory map
Private Const MEM_ARRAY_SIZE As Long = (65536)
Private m_bDO(MEM_ARRAY_SIZE - 1) As Boolean   ’ DO (Discrete Output)
Private m_bDI(MEM_ARRAY_SIZE - 1) As Boolean   ’ DI (Discrete Input)
Private m_iHR(MEM_ARRAY_SIZE - 1) As Integer   ’ Holding register (16 bits)
Private m_iIR(MEM_ARRAY_SIZE - 1) As Integer   ’ Holding register (16 bits)
Private m_byExpSts As Byte

Private Sub Form_Load()

    Set m_caoEng = New CaoEngine
    Set m_caoCntl = m_caoEng. Workspaces(0). AddController(″″,  ″CaoProv. Modbus. X″,  ″″, _
                                           ″Client=False,  Sync=True,  eth: 192.168.0.1″)
    Timer1. Interval = 100
    Timer1. Enabled = True

End Sub

Private Sub Sub Timer1_Timer()

    Dim vntArg As Variant
    Dim vntQueryData As Variant

    ' ″ReceiveQuery″ command execution
    vntQueryData = m_caoCtrl. Execute(″ReceiveQuery″,  vntArg)

    If IsEmpty(vntQueryData) Then
        Exit Sub
    End If

    ' Query data analysis & generating Reply data
    vntArg = AnalyzeQueryDataToCreateReplyData(vntQueryData)

    If VarType(vntArg) <> vbEmpty Then
        ' ″SendReply″command execution
        M_caoCtrl. Execute ″SendReply″,  vntArg
    End If

End Sub

Private Function AnalyzeQueryDataToCreateReplyData(vntQueryData As Variant) As Variant

    ' Query data processing
    Dim i As Long
    Dim lSrvAddress As Long
    Dim lFuncCode As Long
    Dim lAddress As Long
    Dim lCount As Long
    Dim bArray() As Boolean
    Dim iArray() As Integer
    Dim bResult As Boolean
    Dim vntData As Variant
    Dim lArrSize As Long

    lSrvAddress = CLng(vntQueryData(1))
    lFuncCode = CLng(vntQueryData(2))

    Select Case lFuncCode
    ' DO (Discrete Output)multiple reading (1)
    ' DI (Discrete Input)multiple reading (2)
    Case 1,  2
        If (m_bTCP Or ((Not m_bTCP And lSrvAddress <> 0) And (lSrvAddress = m_lUnitAddr))) Then
            lAddress = CLng(vntQueryData(3)(0))
            lCount = CLng(vntQueryData(3)(1))
            If 0 < lCount Then
                If lAddress + lCount <= MEM_ARRAY_SIZE Then
                    ReDim bArray(lCount - 1)
                    If lFuncCode = 1 Then
                        For i = 0 To lCount - 1
                           bArray(i) = m_bDO(lAddress + i)
                        Next
                    Else
                        For i = 0 To lCount - 1
                           bArray(i) = m_bDI(lAddress + i)
                        Next
                    End If
                    vntData = bArray
                    bResult = True
                Else
                    m_byExpSts = 3      ' Exception status 3: Address range is abnormal
                    bResult = False
                End If
            Else
                m_byExpSts = 2      ' Exception status 2: Bit count is abnormal
                bResult = False
            End If
        Else
            Exit Function
        End If

    ' Holding register (16 bits)multiple reading (3)
    ' Input register (16 bits)multiple reading (4)
    Case 3,  4
        If (m_bTCP Or ((Not m_bTCP And lSrvAddress <> 0) And (lSrvAddress = m_lUnitAddr))) Then
            lAddress = CLng(vntQueryData(3)(0))
            lCount = CLng(vntQueryData(3)(1))
            If 0 < lCount Then
                If lAddress + lCount <= MEM_ARRAY_SIZE Then
                    ReDim iArray(lCount - 1)
                    If lFuncCode = 3 Then
                        For i = 0 To lCount - 1
                           iArray(i) = m_iHR(lAddress + i)
                        Next
                    Else
                        For i = 0 To lCount - 1
                           iArray(i) = m_iIR(lAddress + i)
                        Next
                    End If
                    vntData = iArray
                    bResult = True
                Else
                    m_byExpSts = 3      ' Exception status 3: Address range is abnormal
                    bResult = False
                End If
            Else
                m_byExpSts = 2      ' Exception status 2: Bit count is abnormal
                bResult = False
            End If
        Else
            Exit Function
        End If

    ' Read Exception status (7)
    Case 7
        If (m_bTCP Or ((Not m_bTCP And lSrvAddress <> 0) And (lSrvAddress = m_lUnitAddr))) Then
            vntData = m_byExpSts
            bResult = True
        Else
            Exit Function
        End If

    ' DO (Discrete Output) multiple writing (15)
    ' Holding register (16 bits)multiple writing (16)
    Case 15,  16
        lAddress = CLng(vntQueryData(3)(0))
        lCount = CLng(vntQueryData(3)(1))
        If 0 < lCount Then
            If lAddress + lCount <= MEM_ARRAY_SIZE Then
                Dim vt As Variant
                vt = VarType(vntQueryData(3)(2))
                If ((lFuncCode = 15) And vt = (vbArray Or vbBoolean)) Or _
                   ((lFuncCode = 16) And vt = (vbArray Or vbInteger)) Then
                    If lCount = UBound(vntQueryData(3)(2)) + 1 Then
                        If lFuncCode = 15 Then
                            For i = 0 To lCount - 1
                               m_bDO(lAddress + i) = vntQueryData(3)(2)(i)
                            Next
                        Else
                            For i = 0 To lCount - 1
                               m_iHR(lAddress + i) = CInt(vntQueryData(3)(2)(i))
                            Next
                        End If
                        bResult = True
                    Else
                       m_byExpSts = 5      ' Exception status 5: Incorrect bit count for writing
                        bResult = False
                    End If
                Else
                    m_byExpSts = 4      ' Exception status 4: Incorrect data type for writing
                    bResult = False
                End If
            Else
                m_byExpSts = 3      ' Exception status 3: Address range is abnormal
                bResult = False
            End If
        Else
            m_byExpSts = 2      ' Exception status 2: Bit count is abnormal
            bResult = False
        End If
    Case Else
            m_byExpSts = 1      ' Exception status 1: Notification function code is abnormal
            bResult = False
    End Select

    AnalyzeQueryDataToCreateReplyData = Array(lFuncCode,  bResult,  vntData)
```

```
End Function
```

3.2.2. Sample for Asynchronous mode

For the condition of this sample program, the communication is established as Server mode, Asynchronous mode, TCP communication mode, and with the IP address "192.168.0.1".

This sample program will:

: with CaoController::OnMessage"QUERY_MSG", receive a request (query) message from the client device, and,

: reply the result with Message: : Reply() method.

List 3-2-2 **Sample322. frm**

```
Private m_caoEng As CaoEngine
Private WithEvents m_caoCntl As CaoController

' Modbus memory map
Private Const MEM_ARRAY_SIZE As Long = (65536)
Private m_bDO(MEM_ARRAY_SIZE - 1) As Boolean   ' DO (Discrete Output)
Private m_bDI(MEM_ARRAY_SIZE - 1) As Boolean   ' DI (Discrete Input)
Private m_iHR(MEM_ARRAY_SIZE - 1) As Integer   ' Holding register (16 bits)
Private m_iIR(MEM_ARRAY_SIZE - 1) As Integer   ' Holding register (16 bits)
Private m_byExpSts As Byte

Private Sub Form_Load()

    Set m_caoEng = New CaoEngine
    Set m_caoCntl = m_caoEng. Workspaces(0). AddController("",  "CaoProv. Modbus. X",  "", _
                                          "Client=False,  Sync=False,  eth: 192.168.0.1")

End Sub

Private Sub m_caoCtrl_OnMessage(ByVal pICaoMess As CAOLib. ICaoMessage)

    Select Case pICaoMess. Number
    Case MSG_ID_QUERY_MSG

        ' Query data analysis & generating Reply data
        Dim vntReply As Variant
        vntReply = AnalyzeQueryDataToCreateReplyData(pICaoMess. Value)
        PutLogQueryMsg pICaoMess

        If VarType(vntReply) <> vbEmpty Then
            ' Reply (response) processing
            pICaoMess. Reply vntReply
            PutLogReplyData vntReply
        End If

    End Select

    Exit Sub

End Sub
```

4. Appendixes

4.1. Comparison with Old Modbus Command names

Old Modbus Command names compared with the new ones are as follows.

Table 1　List of Comparison with Old Modbus Command names

Old Command names	New Command names	Function Code (HEX)	Page
ReadCoilStatus	ReadMultipleDiscreteOutputs	1(0x01)	P.29
ReadInputStatus	ReadMultipleDiscreteInputs	2(0x02)	P.29
ReadHoldingRegister	ReadMultipleHoldingRegisters	3(0x03)	P.29
ReadInputRegister	ReadMultipleInputRegisters	4(0x04)	P.31
ForceSingleCoil	WriteSingleDiscreteOutput	5(0x05)	P.32
PresetSingleRegister	WriteSingleHoldingRegister	6(0x06)	P.32
ReadExceptionStatus	the same as the old command name	7(0x07)	P.32
DiagnosticsReturnQueryData	↑	8(0x08) - 0	P.33
DiagnosticsRestartCommunicationsOption	↑	8(0x08) - 1	P.33
ForceMultipleCoils	WriteMultipleDiscreteOutputs	15(0x0F)	P.34
PresetMultipleRegisters	WriteMultipleHoldingRegisters	16(0x10)	P.34
MaskWrite4XRegister	MaskWriteHoldingRegister	22(0x16)	P.36
ReadWrite4XRegisters)	ReadWriteMultipleHoldingRegisters	23(0x17)	P.36
FetchCommEventCounter	use AnotherFunctionCode as a substitute	11(0x0B)	P.37
FetchCommEventLog	↑	12(0x0C)	P.37
ReportSlaveID	↑	17(0x11)	P.37
Read General Reference	↑	20(0x14)	P.37
Write General Reference	↑	21(0x15)	P.37
ReadFIFOQueue	↑	22(0x16)	P.37