Modbus.X providers

Modbus ASCII/RTU/TCP communication

Version 1.0.7

User's Guide

November 9, 2021

[Remark]

This document was created by automatic translation from Japanese.

[Revision History]

Version	Date	Description
1.0.0.0	2015-1-10	First edition
1.0.1.0	2015-4-22	Add Server Mode
1.0.2.0	2015-6-08	DI/DO addressing is fixed at 1 bit regardless of the data width. Hold/input register addressing is fixed to 16 bits regardless of the data width.
1.0.2.1	2015-8-18	Some supplements and errors corrected
1.0.3.0	2015-8-21	Added "OffsetAddressZero" and "Endian" to CaoControlller::AddExtension options.
1. 0. 4. 0	2016-6-2	Sample modification
1. 0. 5. 0	2016-9-6	"VT" and "Elem" added to CaoExtension:AddVariable options.
1. 0. 5.1	2016-9-19	Term definitions added. Comparison with the name of the old Modbus commands was added.
1. 0. 5.2	2016-11-18	Added "RcvPacketLen" and "SndPacketLen" to CaoExtension:AddVariable options.
1. 0. 6. 0	2020-3-12	Optional at CaoWorkspace::AddController: Add UDP-mode to PacketType
1. 0. 7. 0	2021-5-24	Fixed the type mismatch of the result of Execute ("ReceiveQuery").
	2021-11-9	Fixed the description of the Conn option in server mode and Ethernet device. Fixed the sample program.

[Compatible devices]

Model	Version	Notes

Table of Contents

1. Introduction

This user's guide is for Modbus.X providers.

This Modbus.X provider allows CAO clients to easily send and receive Modbus protocols.[1]

This Modbus describes the functionality of the.X providers and the methods that are implemented.

1.1. Definition of Terms

The following terms are defined (unified) in this manual.

- "Coils (Coil)": unified with "DO (Discrete Output)".
- "Input Status" : "DI(Discrete Input)".

[1] Modbus protocol is a serial communication established by Modicon for its programmable logic controllers (PLCs) in 1979. It has become the communication protocol of the de facto standard in the industry, and is now the most common way to connect industrial electronics. (from wikipedia)

For Modbus ASCII/RTU protocol-specification, see Modicon Modbus Protocol Reference Guide PI‐MBUS-300 Rev. J, see OPEN MODBUS/TCP SPECIFICATION Release 1 0, March 1999 29 for protocol-specific information about Modbus TCP.

2. Provider Overview

2.1. Introduction

Modbus.X providers are providers that send and receive Modbus protocols.

When the communication device is com (RS232C/RS485 or other serial device (EIA-485)) and the communication mode is client, it operates as Modbus master and performs serial communication with Modbus slave device. When the communication mode is server, it operates as a Modbus slave. It responds to serial communication from Modbus client device.

When the communication device is eth (Ethernet) and the communication mode is client, TCP/IP communication is performed with Modbus server device, and when the communication mode is server, TCP/IP communication from Modbus client device is responded.

Regardless of the communication device, master = client, slave = server standardizes the names according to the notation described below.

Table 2-1 Communication Devices and Communication Modes

		Communication mode	
Modbus communication protocol	Communication Devices	Client	Server
ASCII,RTU	Com	✓	✓
TCP	Eth	✓	✓※

※The number of clients that can be connected under the TCP protocol is 16.

The file format of Modbus.X provider is DLL (Dynamic Link Library), which is detailed in Table 2-2.

Table 2-2 Modbus.X Providers

File name	CaoProvModbusX.dll
ProgID	CaoProv.Modbus.X
Registry registration[2]	Regsvr32 CaoProvModbusX.dll
Deletion of Registry Registration	Regsvr32 /u CaoProvModbusX.dll

[2] If it is installed by ORiN SDK, it does not need to be registered/deleted manually.

2.2. Execution mode

There are two execution modes for Modbus.X providers: synchronous mode and asynchronous mode. This can be toggled by specifying Sync of AddController.

2.2.1. Asynchronous mode

2.2.1.1. For client mode

A Modbus request (query) message is sent by executing a command that supports CaoExtension::Execute() Modbus function (see Table 2-13) or accessing CaoExtension::CaoVariable (user variable) (see Table 2-15). Then, an OnMessage event is generated when a response message is received from the server device.

2.2.1.2. For server mode

Generates a OnMessage event when a request (query) message is received from a client device. Then, the response message is returned to the client device using CaoMessage::Reply() method of CaoMessage object acquired by OnMessage event.

2.2.2. Synchronous mode

2.2.2.1. For client mode

Modbus communication messages are sent and received by accessing CaoExtension::Execute() Modbus function compatible command (see Table 2-13) or CaoExtension::CaoVariable (user variable) (see Table 2-15).

2.2.2.2. For server mode

CaoController::Execute() "ReceiveQuery command is used to receive a request (query) message from a client device, and CaoController::Execute() "SendReply command is used to send a response message to a client device.

2.3. Method properties

2.3.1. CaoWorkspace::AddController method

In Modbus.X providers, the connection parameters are set on AddController to connect communication.
In this case, specify the communication mode, connection parameters, timeout settings, etc. with the option.
AddController parameter specifications are shown below.

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

BstrCtrlName : [in] Controller name
BstrProvName : [in] Provider name. Fixed value = " CaoProv. Modbus.X"
BstrPcName : [in] Provider's running machine name
BstrOption : [in] Option character string

The following is a listing for Option character string: If "-" is entered in the communication device field, the option will be ignored when the device is specified.

Table 2-3 Option character string in the CaoWorkspace::AddController

Option	Description	Server		Client	
		Eth	Com	Eth	Com
Client [=True / False]	Set the communication mode (client/server). True: Client (default) False: Server NOTE: The number of clients that can be connected in server mode is 16.	✓	✓	✓	✓
ServerUnitAddress [=<server device address>]	Set the server device address or unit identifier. At com: Server device address (range: 1-255) Default: 1 Eth: Unit identifier (range:-1, 0-255) Default:-1 N.B.: If "-1" is set at eth, the unit identifier is arbitrary, and reception is notified to all unit identifiers transmitted from the client device. NOTE: In client mode, this option is ignored.	✓	✓	-	-
Sync [=True / False]	Set the synchronization mode. True: Synchronous mode (default) False: Asynchronous mode	✓	✓	✓	✓

Conn =<connection parameter>	Required. Communication format and connection parameters. (See 2.3.1.1)	✓	✓	✓	✓
PacketType [=<packet parameter>]	Sets Modbus communication protocol data type. When the communication device specified by Conn option is "com": 0: RTU (default) 1: ASCII When the communication device specified by Conn option is "eth" and in client mode: 0: TCP (default) 1: UDP NOTE: This option is ignored when in server mode and the communication device specified by Conn option is "eth".	-	✓	✓	✓
TcpConnectionTimeout [=<TCP connection timeout>]	Set the TCP connection timeout time [ms]. If a request (query) message is not received within the set time, the TCP connection is disconnected. Range: 0 to 3600000 (default: 0) 0 for timeout disabled NOTE: This option is ignored in client mode or when the communication device specified in Conn option is "com".	✓	-	-	-
ReceiveQueryTimeout [=<query reception timeout>]	Set the query reception timeout time [ms]. Range: 0 to 100000 (default: 0) NOTE: In client mode, this option is ignored.	✓	✓	-	-
SendReplyTimeout [=<reply send timeout>]	Set the reply transmission timeout [ms]. Range: 1 to 100000 (default: 1000) NOTE: In client mode, this option is ignored.	✓	✓	-	-
Timeout [=<send/receive timeout>]	Set the send/receive timeout time [ms]. Range: 1 to 100000 (default: 1000) NOTE: In server mode, this option is ignored.	-	-	✓	✓
Retry [=<retry count>]	Set the number of communication retries during transmission/reception. Range: 0 to 10 (default: 0) NOTE: In server mode, this option is ignored.	-	-	✓	✓

OffsetAddressZero [=<True/False>]	Set the register address value specified by the parameters of Execute commands and the offset value of ? (register address value) specified by the end of each user variable name to 0 to. True: The offset value of the register address is 0 ~ False: The offset value of the register address is from 1 to (Defaults to False) NOTE: In server mode, this option is ignored. When setting for each server device, Set the "OffsetAddressZero" option (Table2-4) for CaoController::AddExtension.	-	-	✓	✓
RtsTransmitDelayTime [=<transmit/receive switching delay time>]	Set the transmission/reception switching delay time [ms] using the RTS signal. 0: RTS signal is always ON (default) 1 to 100000: With transmitter/receiver circuit control by RTS signal RTS ON → transmission starts immediately before transmission. → Transmit completion ※1 → is RTS OFF after the set delay has elapsed. NOTE: - This is mainly used when the transmission mode is half-duplex and the hardware configuration requires switching of the transmission/reception circuit by software. When this bit is set to "1" ms or more, the transmission/reception is switched by the RTS signal. - The determination of *1 (transmission completion) in this provider is earlier than that in the actual transmission path because the transmission completion notification from the communication device driver is used as the starting point. (It is recommended that the delay time be calculated when FIFO of the communication hardware is used to calculate the delay time until the actual transmission is completed, because the delay time is vendor-dependent and FIFO is not used.) - This option is ignored when the device specified in Conn option is "eth".	-	✓	-	✓

PollDelayTime [=<polling delay>]	Set the polling delay time [ms]. Range: 0 to 100000 (default: 0) NOTE: In server mode, this option is ignored.	-	-	✓	✓
Endian [=<Int>[:<Float>]]	Sets the data transfer order (endian) for 32-bit commands. <Int> 1: 32-bit integer/floating-point big-endian (upper word → lower word) 0: 32-bit integer/floating-point little-endian (Lower word → upper word) (Default) However, for the floating point type, the following <Float> specification is used Applicable only if not. <Float> 1: 32-bit floating-point big-endian 0: 32-bit floating-point little-endian NOTE: In server mode, this option is ignored. When setting for each server device, "Endian" Optional for CaoController::AddExtension Configure (Table 2-4).	-	-	✓	✓

2.3.1.1. Conn Optional

The following is a Conn optional connection parameter string: Here, brackets ("[]") are optional. In the explanation of each parameter, the underlined part becomes the default value when no option is specified.

- RS232C/RS485 Devices

"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 (UART hardware-dependent)
<Parity>	:	Parity . 'N'-NONE,'E'-EVEN,'O'-ODD
<DataBits>	:	Number of data bits. '7'-7bit,'8'-8bit N.B.: When PacketType option is set to RTU mode, the number of data bits is fixed at 8. Therefore, if a value other than '8' is specified, an error occurs.
<StopBits>	:	No. of Stop Bits . '1'-1bit,'2'-2bit

※ The flow control setting is fixed to None (NONE) because the RTS-signal is used by the "RtsTransmitDelayTime" option.

- Ethernet Devices

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

<IP Address>	:	<In client mode> Connection destination server device IP address <In server mode> 0.0.0.0 Fixed NOTE: The number of clients that can be connected is 16.
<Port No>	:	<In client mode> TCP connection port number to connect to Default: 502 <In server mode> Own TCP connection port number Default: 502

2.3.2. CaoController::Execute method

Refer to 2.4.1 for the available command names and details.

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

BstrCommand : [in] Command name

VntParam : [in] Parameters

pVal : [out] Acquired data

2.3.3. CaoController::AddVariable method

Create a variable object. Only variables 2.5.1 can be used in variable names.

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

BstrName : [in] Any name

BstrOption : [in] Option character string (not used)

2.3.4. CaoController::GetVariableNames Properties

Get the variable name list for 2.5.1.

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

In client mode, a CaoExtension for communicating with Modsbus server devices is generated.

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

BstrName : [in] Any name

BstrOption : [in] Option character string

The available "Option character string" are listed below.

Table 2-4 Option character string in the CaoController: AddExtension

Option	Meaning
UnitAddress [=<device address>]	Set the server device address (at com) or unit identifier (at eth) of the communication destination. At com: Server device address (range: 0-255) Default: 1 Eth: Unit identifier (range: 0-255) Default: 0
OffsetAddressZero [=<True/False>]	For each server device address, set the register address value specified by the parameters of each Execute command and the offset value of ? (register address value) specified by the end of each user variable name to 0 to. See the "OffsetAddressZero" option in Table 2-3 for more information. Default: "OffsetAddressZero" option setting at CaoWorkspace::AddController.

Endian [=<Int>[:<Float>]]	Set the data transfer order (endian) for the 32-bit command for each server device address. See the "Endian" option in Table 2-3 for more information. Default: "Endian" option setting at CaoWorkspace::AddController.

2.3.6. CaoExtension::GetID socket (client mode only)

In client mode, the device address of the server specified by the "UnitAddress" option is acquired.

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

In client mode, Execute command that communicates with the server device is executed according to the command name defined for each function (Modbus function code).

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

BstrCommand : [in] Command name

VntParam : [in] Parameters

pVal : [out] Acquired data

See Table 2-13 for the available "Command Names".

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

In client mode, create a variable-object that communicates with Modsbus server device. Only the variables in Table 2-15 can be used for variable names. If you specify a variable name other than these, the method returns an error.

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

BstrName : [in] Any name

BstrOption : [in] Option character string

The following is a listing for Option character string:

Table 2-5 Option character string in the CaoExtension:AddVariable

Option	Meaning
UserVarWidth[=<user-variable-data-width>]	Set the user variable data width [bit]. [Range] - <bstrName>= DO? or DI? Hour: 1 (default), 8, 16, 32 [bit] - <bstrName>= HRI? or IRI? Hour: 16 (default), 32 [bit] - <bstrName>= HRF? or IRF? Hour: 32 (default) [bit] fixed NOTE: If this option is not specified and the "VT" option is specified, this option is ignored.
VT[=<variable type>]	Specifies the data type. (See Table 2-6 for details.) [Supported bstrName] **<bstrName>= DO? or DI? or HRI? or IRI?** N.B.: HRF? When Or IRF? is specified, this option is ignored and the variable type is fixed to VT_R4. Default: Disabled and the "UserVarWidth" option takes precedence. N.B.: This option is ignored when the "UserVarWidth" option is specified.
Elem[=<number of elements>]	Specifies the number of data elements. See Table 2-6 for the number of elements per <bstrName> and "VT" option. The "VT" option is an array type (VT_ARRAY) for variable types other than BSTR and specifies the number of elements. If the "VT" option is BSTR, the variable type specifies the number of character strings (in bytes) in VT_BSTR. (Note: Not the number of elements of array type (VT_ARRAY)) The number of elements can be specified as a decimal number or as a hexadecimal number. e.g.) 0x0A, &h0A, 0AH [Supported bstrName] <bstrName>= DO? or DI? or HRI? or IRI? or HRF? or IRF? Default: Invalid (variable type is non-array) ※The data type when the VT option is omitted is as follows. See Table 2-6 for the element count range and ? (address) range for each data type. [<bstrName>= DO? or DI? Hour] 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? Hour] UserVarWidth=16: VT_ARRAY | VT_UI2 UserVarWidth = 32: VT_ARRAY | VT_UI4 … Endian option: Enabled [<bstrName>= HRF? or IRFI? Hour] VT_ARRAY | VT_R4... "Endian" option: Enabled
RcvPacketLen[=<receive-packet-length>]	Specifies the maximum receive packet length in WORD when acquiring data. Range: 4 to 125 Omitted or out of range: 125
SndPacketLen[=<send-packet-length>]	Specifies the maximum transmit packet length in WORD when data is set. Range: 4 to 123 When omitted or out of range: 123

Table 2-6 List of data types that can be specified with the VT option

VT	Data Type		Description
	"Elem" Not specified	"Elem" With specification	
BIT	VT_UI1	VT_ARRAY | VT_UI1 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 65536 When "HRI?" or "IRI?" is selected: 1 to 131072	Data is converted to 2-value of 0/1 and written/read. Val==0:0, Val!=0:1 [? (Address) Range] When "OffsetAddressZero=False" is selected: 1 to 65536 When "OffsetAddressZero=True" is selected: 0 to 65535
BOOL	VT_BOOL	VT_ARRAY | VT_ BOOL [Number of elements range] When "DO?" or "DI?" is selected: 1 to 65536 When "HRI?" or "IRI?" is selected: 1 to 65536	Data is converted to two values of 0/-1 and written/read. Val==VARIANT_FALSE:0,Val!=VARIANT_FALSE:-1 [? (Address) Range] When "OffsetAddressZero=False" is selected: 1 to 65536 When "OffsetAddressZero=True" is selected: 0 to 65535
BSTR	VT_BSTR	VT_BSTR [Number of elements range] When "DO?" or "DI?" is selected: 1 to 8192 When "HRI?" or "IRI?" is selected: 1 to 131072	This command writes/reads BSTR as ASCII. When "Elem" is not specified: 1 character (byte) When "Elem" is specified: Specified number of characters (in bytes) [? (Address) Range] <"DO?"or"DI?" When "OffsetAddressZero=False" is selected: 1 to 65529 When "OffsetAddressZero=True" is selected: 0 to 65528 <"HRI?"or"IRI?" When "OffsetAddressZero=False" is selected: 1 to 65536 When "OffsetAddressZero=True" is selected: 0 to 65535
I1	VT_I1	VT_ARRAY | VT_I1 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 8192 When "HRI?" or "IRI?" is selected: 1 to 131072	Write/read as signed 1-byte data. [? (Address) Range] <"DO?"or"DI?" When "OffsetAddressZero=False" is selected: 1 to 65529 When "OffsetAddressZero=True" is selected: 0 to 65528 <"HRI?"or"IRI?" When "OffsetAddressZero=False" is selected: 1 to 65536 When "OffsetAddressZero=True" is selected: 0 to 65535
I2	VT_I2	VT_ARRAY | VT_ I2	Write/read as signed 2-byte data.

		[Number of elements range] When "DO?" or "DI?" is selected: 1 to 4096 When "HRI?" or "IRI?" is selected: 1 to 65536	[? (Address) Range] <"DO?"or"DI?" When "OffsetAddressZero=False" is selected: 1 to 65521 When "OffsetAddressZero=True" is selected: 0 to 65520 <"HRI?"or"IRI?" When "OffsetAddressZero=False" is selected: 1 to 65536 When "OffsetAddressZero=True" is selected: 0 to 65535
I4	VT_I4	VT_ARRAY | VT_I4 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 2048 When "HRI?" or "IRI?" is selected: 1 to 32768	Write/read as signed 4-byte data. [? (Address) Range] <"DO?" or "DI?" Hours … Endian Optional: Disabled> When "OffsetAddressZero=False" is selected: 1 to 65505 When "OffsetAddressZero=True" is selected: 0 to 65504 <When "HRI?" or "IRI?" The "Endian" option: Enabled> When "OffsetAddressZero=False" is selected: 1 to 65535 When "OffsetAddressZero=True" is selected: 0 to 65534
I8	VT_I8	VT_ARRAY | VT_I8 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 1024 When "HRI?" or "IRI?" is selected: 1 to 16384	Write/read as signed 8-byte data. [? (Address) Range] <"DO?" or "DI?" Hours … Endian Optional: Disabled> When "OffsetAddressZero=False" is selected: 1 to 65473 When "OffsetAddressZero=True" is selected: 0 to 65472 <When "HRI?" or "IRI?" The "Endian" option: Enabled> When "OffsetAddressZero=False" is selected: 1 to 65533 When "OffsetAddressZero=True" is selected: 0 to 65532
UI1	VT_UI1	VT_ARRAY | VT_UI1 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 8192 When "HRI?" or "IRI?" is selected: 1 to 131072	Write/read as unsigned 1-byte data. [? (Address) Range] <"DO?"or"DI?" When "OffsetAddressZero=False" is selected: 1 to 65529 When "OffsetAddressZero=True" is selected: 0 to 65528 <"HRI?"or"IRI?" When "OffsetAddressZero=False" is selected: 1 to 65536 When "OffsetAddressZero=True" is selected: 0 to 65535
UI2	VT_UI2	VT_ARRAY | VT_UI2 [Number of elements range]	Write/read as unsigned 2-byte data. [? (Address) Range]

		When "DO?" or "DI?" is selected: 1 to 4096 When "HRI?" or "IRI?" is selected: 1 to 65536	<"DO?"or"DI?" When "OffsetAddressZero=False" is selected: 1 to 65521 When "OffsetAddressZero=True" is selected: 0 to 65520 <"HRI?"or"IRI?" When "OffsetAddressZero=False" is selected: 1 to 65536 When "OffsetAddressZero=True" is selected: 0 to 65535
UI4	VT_UI4	VT_ARRAY | VT_UI4 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 2048 When "HRI?" or "IRI?" is selected: 1 to 32768	Write/read as unsigned 4-byte data. [? (Address) Range] <"DO?" or "DI?" Hours … Endian Optional: Disabled> When "OffsetAddressZero=False" is selected: 1 to 65505 When "OffsetAddressZero=True" is selected: 0 to 65504 <When "HRI?" or "IRI?" The "Endian" option: Enabled> When "OffsetAddressZero=False" is selected: 1 to 65535 When "OffsetAddressZero=True" is selected: 0 to 65534
UI8	VT_UI8	VT_ARRAY | VT_UI8 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 1024 When "HRI?" or "IRI?" is selected: 1 to 16384	Write/read as unsigned 8-byte data. [? (Address) Range] <"DO?" or "DI?" Hours … Endian Optional: Disabled> When "OffsetAddressZero=False" is selected: 1 to 65473 When "OffsetAddressZero=True" is selected: 0 to 65472 <When "HRI?" or "IRI?" The "Endian" option: Enabled> When "OffsetAddressZero=False" is selected: 1 to 65533 When "OffsetAddressZero=True" is selected: 0 to 65532
R4	VT_R4	VT_ARRAY | VT_R4 [Number of elements range] When "DO?" or "DI?" is selected: 1 to 2048 When "HRI?" or "IRI?" is selected: 1 to 32768	Write/read as single precision floating point (4 bytes) data. [? (Address) Range] <"DO?" or "DI?" Hours … Endian Optional: Disabled> When "OffsetAddressZero=False" is selected: 1 to 65505 When "OffsetAddressZero=True" is selected: 0 to 65504 <When "HRI?" or "IRI?" The "Endian" option: Enabled> When "OffsetAddressZero=False" is selected: 1 to 65535 When "OffsetAddressZero=True" is selected: 0 to 65534
R8	VT_R8	VT_ARRAY | VT_R8 [Number of elements range] When "DO?" or "DI?" is	Write/read as double precision floating point (8 bytes) data. [? (Address) Range] <"DO?" or "DI?" Hours … Endian Optional: Disabled>

		selected: 1 to 1024 When "HRI?" or "IRI?" is selected: 1 to 16384	When "OffsetAddressZero=False" is selected: 1 to 65473 When "OffsetAddressZero=True" is selected: 0 to 65472 <When "HRI?" or "IRI?" The "Endian" option: Enabled> When "OffsetAddressZero=False" is selected: 1 to 65533 When "OffsetAddressZero=True" is selected: 0 to 65532

2.3.9. CaoExtension::GetVariableNames Properties (Client Mode Only)

Retrieves the variable name list shown in Table 2-15 in client mode.

2.3.10. CaoVariable::get_Value Property

Retrieves information corresponding to a variable. See 2.5 for the implementation status and retrieval data for each variable.

2.3.11. CaoVariable::put_Value Property

Set the information corresponding to the variable. For the implementation status and setting data of each variable, see 2.5.

2.3.12. CaoController::OnMessage event

OnMessage event shown in the table below occurs.

Table 2-7 CaoController::OnMessage events

No	Desciption	Function	Availability of each operation mode								Page
			Server				Client				
			Synchronization		Asynchronous		Synchronization		Asynchronous		
			Eth	Com	Eth	Com	Eth	Com	Eth	Com	
1	REPLY_MSG	Notification when a response message is received from a server device	-	-	-	-	-	-	✓	✓	P.20
2	QUERY_MSG	Notification when a request (query) message from a client device is received	-	-	✓	✓	-	-	-	-	P.21
3	IPINFO_MSG	TCP client device connection/disconnection notification	✓	-	✓	-	-	-	-	-	P.23

REPLY_MSG

Occurrence condition When a response 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 Supported Execute Command-Name"

Array[1]: VT_BSTR

Source IP address example) "192.168.0.1" …… If you are using com, an empty character ("")

Array[2]: VT_I4

Server device address or unit identifier (range: 1-255)

Array[3]: VT_I4

Function code (Range: 1-127)

Array[4]: VT_I4

Function subcode: Fixed to 0 when 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 (or VT_EMPTY if none). See 2.4.4 for details.

Description

Notifies that a response message from the server device has been received.

The result of executing the request (query) function is stored in Value property.

NOTE: This command can be handled by each operation mode. See Table 2-7 for details.

QUERY_MSG

Occurrence condition When 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: 0-255)

Array[2]: VT_I4

See Table 2-8 below for more information on Modbus protocol Function Code.

Array[3]: VT_ARRAY|VT_VARIANT

Parameters by Modbus protocol Function Code. See Table 2-8 below for details.

Table 2-8

Function	Array[2]	Array[3][0]	Array[3][1]	Array[3][2]
DO(Discrete Output) Multiple reads	1 (0x01)	VT_I4: Start address	VT_I4: Number of points to be read	VT_EMPTY: None
DI(Discrete Input) Multiple reads	2 (0x02)	VT_I4: Start address	VT_I4: Number of points to be read	VT_EMPTY: None
Holding register (16 bits) Multiple reads	3 (0x03)	VT_I4: Start address	VT_I4: Number of read data	VT_EMPTY: None
Input register (16 bits) Multiple reads	4 (0x04)	VT_I4: Start address	VT_I4: Number of read data	VT_EMPTY: None
Exception status Read status	7 (0x07)	VT_EMPTY: None		
DO(Discrete Output) Multiple writing	15 (0x0F)	VT_I4: Start address	VT_I4: Written score	VT_ARRAY | VT_BOOL Write data
Holding register (16 bits) Multiple writing	16 (0x10)	VT_I4: Start address	VT_I4: Number of write data	VT_ARRAY | VT_I2 Write data

Description Notifies that a request (query) message has been received from the client device.

Modbus protocol Function Code of receivable notification is shown in Table 2-9.

After the CAO-client normally handles the requesting function in Value,

Message::Reply() You must use the method to return a response message (the argument is the same as CaoController::Execute "SendReply command on page 27) within the time set by the "SendReplyTimeout" option.
If Message::Reply() method is executed after the time set by the "SendReplyTimeout" option has elapsed, an error is returned.

NOTE: This command can be handled by each operation mode. See Table 2-7 for details.

Table 2-9 Receive notification Modbus function codes (server-mode only)

Function		On Modbus protocols				Notification message (QUERY_MSE) or Receive command (ReceiveQuery)			Automatic Response	
		Broadcast		Function Code (HEX)	Sub Code	Notification (reception)	Notification FunctionCode(HEX)	Notification Sub Code	Normal reply ※1	Exception-response ※2
		TCP	ASCII / RTU							
DO(Discrete Output) Multiple reads		×	×	1 (0x01)	-	✓	1 (0x01)	-	-	✓
DI(Discrete Input) Multiple reads		×	×	2 (0x02)	-	✓	2 (0x02)	-	-	✓
Read multiple holding registers (16 bits)		×	×	3 (0x03)	-	✓	3 (0x03)	-	-	✓
Read multiple input registers (16 bits)		×	×	4 (0x04)	-	✓	4 (0x04)	-	-	✓
DO(Discrete Output) Single output		×	✓	5 (0x05)	-	✓	15 (0x0F) ※6	-	-	✓
Holding register (16-bit) single write		×	✓	6 (0x06)	-	✓	16 (0x10) ※3	-	-	✓
Exception status read		×	×	7 (0x07)	-	✓	7 (0x07)	-	-	✓
Diagnostics	Echo Back Query Data	×	×	8 (0x08)	0	-	-	-	✓	✓
	Other	×	×		1~	-	-	-	-	✓
DO(Discrete Output) Multiple writing		×	✓	15 (0x0F)	-	✓	15 (0x0F)	-	-	✓
Writing multiple holding registers (16 bits)		×	✓	16 (0x10)	-	✓	16 (0x10))	-	-	✓
Holding register AND/OR masking output		×	×	22 (0x16)	-	✓	Two notices (3(0x03) ※4 16(0x10)	-	-	✓
Read/Write multiple holding registers		×	×	23 (0x17)	-	✓	Two notices (16(0x10) After 3 (0x03)) ※5	-	-	✓
Other				Other than the above	-	-	-	-	-	✓

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

※2: Returns an exception response (reply) message automatically without notification.
※3: FunctionCode on Modbus protocol is 6, but the notification FunctionCode is 16. The number of data items to be written (Count) is fixed to 1.
※4: FunctionCode on Modbus protocols is 22, but the notification FunctionCode is divided into two, and the result (value) of AND/OR operation performed on the read value in 3 for the first time is written and notified in 16 for the second time. The number of data items to be written (Count) is fixed to 1.
※5: FunctionCode on Modbus protocol is 23 (0x17), but the notification FunctionCode is divided into two, the first is 16 and the second is 3.
※6: FunctionCode on Modbus protocol is 5, but the notification FunctionCode is 10. The number of data items to be written (Count) is fixed to 1.

IPINFO_MSG

Occurrence condition When a TCP client device connects or disconnects

Number 3

Description IPINFO_MSG

Source

Destination

Value VT_ARRAY | VT_VARIANT

Array[0]: VT_BOOL

Connection Status: Connected (VARIANT_TRUE), Disconnected (VARIANT_FALSE)

Array[1]: VT_BSTR

Connected or disconnected, IP address example) "192.168.0.1"

Array[2]: VT_ARRAY|VT_VARIANT

As with the "@IpInfo" system variable,

Obtain the IP address and port number of the currently connected client device.

For more information, see the "@IpInfo" system variable in Table 2-14.

Description Indicates that a TCP client device is connected or disconnected.

[Connection Conditions]

When a connection request is received from a TCP client device

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

[Cut condition]

- When a disconnection request is received from a TCP client device.
- If TCP connection timeout time [ms] is set to a value other than disable (0) with the "TcpConnectionTimeout" option, and a request (query) message is not received within the set time after the TCP client device connects.

−The length of the request/query message received is less than 6Byte.

−When sending a reply message fails.

NOTE: This command can be handled by each operation mode. See Table 2-7 for details.

2.4. Command list

2.4.1. CaoController classes

Table 2-10 CaoController::Execute Commands

Command name	Function	Availability of each operation mode								Page
		Server				Client				
		Synchronization		Asynchronous		Synchronization		Asynchronous		
		Eth	Com	Eth	Com	Eth	Com	Eth	Com	
ProviderCancel	Set to cancel state	✓	✓	✓	✓	✓	✓	✓	✓	P.25
ProviderClear	Cancel Status Release	✓	✓	✓	✓	✓	✓	✓	✓	P.25
ReceiveQuery	Receive request (query) message	✓	✓	-	-	-	-	-	-	P.26
SendReply	Send response message	✓	✓	-	-	-	-	-	-	P.27

2.4.2. More information on CaoController::Execute commands

ProviderCancel

All operating modes available

Syntax ***Object.ProviderCancel()***

Argument None

Return Value None

Description Set the provider to the canceled state.

Transmission/reception is suspended while the CANCEL state is reached.

To cancel the cancellation status, execute the "ProviderClear" command.

ProviderClear

All operating modes available

Syntax ***Object.ProviderClear()***

Argument None

Return Value None

Description Cancels the cancel state of the provider.

ReceiveQuery

Available only in server mode

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: 0-255)

Array[2]: VT_I4

See Table 2-11 below for more information on Modbus protocol Function Code.

Array[3]: VT_ARRAY|VT_VARIANT

Parameters by Modbus protocol Function Code. See Table 2-11 below for details.

Table 2-11

Function	Array[2]	Array[3][0]	Array[3][1]	Array[3][2]
DO(Discrete Output) Multiple reads	1 (0x01)	VT_I4: Start address	VT_I4: Number of points to be read	VT_EMPTY: None
DI(Discrete Input) Multiple reads	2 (0x02)	VT_I4: Start address	VT_I4: Number of points to be read	VT_EMPTY: None
Holding register (16 bits) Multiple reads	3 (0x03)	VT_I4: Start address	VT_I4: Number of read data	VT_EMPTY: None
Input register (16 bits) Multiple reads	4 (0x04)	VT_I4: Start address	VT_I4: Number of read data	VT_EMPTY: None
Exception status Read status	7 (0x07)	VT_EMPTY: None		
DO(Discrete Output) Multiple writing	15 (0x0F)	VT_I4: Start address	VT_I4: Written score	VT_ARRAY | VT_BOOL Write data
Holding register (16 bits) Multiple writing	16 (0x10)	VT_I4: Start address	VT_I4: Number of write data	VT_ARRAY | VT_I2 Write data

Description

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

Modbus protocol Function Code of receivable notification is shown in Table 2-9.

If the first byte of the request (query) message has not been received even after the time set by the "ReceiveQueryTimeout" option has elapsed, the processing is returned. In this case, <Data> is VT_EMPTY.

After the CAO-client normally handles the requesting function in the <Data>, SendReply() You must use the command to return a response message (see SendReply (page 27) for more information) within the time set by the "SendReplyTimeout" option.

NOTE: This command is available or unavailable in each operation mode. See Table 2-10 for details.

SendReply

Available only in server mode

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

Process (VARIANT_TRUE: Normal, VARIANT_FALSE: Error)

Array[2]: When the execution result is normal, set the data type in the following table by the notification Modbus FunctionCode. If the execution result is abnormal, set VT_EMPTY.

Table 2-12

Function	Notification Function Code (HEX)	Data Type	Description
DO(Discrete Output) Multiple reads	1 (0x01)	VT_ARRAY | VT_BOOL	Read data
DI(Discrete Input) Multiple reads	2 (0x02)	VT_ARRAY | VT_BOOL	Read data
Read multiple holding registers (16 bits)	3 (0x03)	VT_ARRAY | VT_I2	Read data
Read multiple input registers (16 bits)	4 (0x04)	VT_ARRAY | VT_I2	Read data
Exception status read	7 (0x07)	VT_UI1	Exception status
DO(Discrete Output) Multiple writing	15 (0x0F)	VT_EMPTY	No data
Writing multiple holding registers (16 bits)	16 (0x10)	VT_EMPTY	No data

Return Value

None

Description

Send a response message to the client device.

Normally, after the CAO client performs processing for the request function received by ReceiveQuery() command, the CAO client must use this command to return a response message within the time set by the "SendReplyTimeout" option.

If this command is executed after the time set by the "SendReplyTimeout" option has

elapsed, an error is returned.

When the communication device is com and the server device address received by ReceiveQuery() command is broadcast (0), this command does not need to be executed. Even if this command is executed, a response message to the client device is not sent. Also, if the process result (Array[1]) is set to abnormal (VARIANT_FALSE), it will not be transmitted in the same way.

NOTE: This command is available or unavailable in each operation mode. See Table 2-10 for details.

2.4.3. CaoExtension (client mode only)

The table below shows the commands that support Modbus function in client mode.

Table 2-13 Table List of Commands Supported by CaoExtension::Execute Modbus Function (Client Mode Only)

Command name (Former Modbus Provider-Supported Commands)	Modbus protocol				Function	Page
	Broadcast		Function Code (HEX)	Sub Code		
	TCP	ASCII / RTU				
ReadMultipleDiscreteOutputs (ReadCoilStatus)	×	×	1 (0x01)	-	DO(Discrete Output) Multiple reads	P.30
ReadMultipleDiscreteInputs (ReadInputStatus)	×	×	2 (0x02)	-	DI(Discrete Input) Multiple reads	P.30
ReadMultipleHoldingRegisters (ReadHoldingRegister)	×	×	3 (0x03)	-	Read multiple holding registers (16 bits)	P.30
ReadMultipleHoldingRegistersLongInt (None)					Reading multiple holding registers (32-bit integer type)	P.31
ReadMultipleHoldingRegistersFloat (None)					Read multiple holding registers (32-bit floating point type)	P.31
ReadMultipleInputRegisters (ReadInputRegister)	×	×	4 (0x04)	-	Read multiple input registers (16 bits)	P.32
RreadMultipleInputRegistersLongInt (None)					Read multiple input registers (32-bit integer type)	P.32
RreadMultipleInputRegistersFloat (None)					Read multiple input registers (32-bit floating point type)	P.32
WriteSingleDiscreteOutput (ForceSingleCoil)	×	✓	5 (0x05)	-	DO(Discrete Output) Single output	P.33
WriteSingleHoldingRegister (PresetSingleRegister)	×	✓	6 (0x06)	-	Holding register (16-bit) single write	P.33
ReadExceptionStatus The same	×	×	7 (0x07)	-	Exception status read	P.33
DiagnosticsReturnQueryData The same	×	×	8 (0x08)	0	Diagnostics: Echo Back Query Data	P.34
DiagnosticsRestartCommunicationsOption The same				1	Diagnostics: Communication Port Initialization	P.34
WriteMultipleDiscreteOutputs (ForceMultipleCoils)	×	✓	15 (0x0F)	-	DO(Discrete Output) Multiple writing	P.35
WriteMultipleHoldingRegisters (PresetMultipleRegisters)	×	✓	16 (0x10)	-	Writing multiple holding registers (16 bits)	P.35
WriteMultipleHoldingRegistersLongInt (None)					Writing multiple holding registers (32-bit integer type)	P.35
WriteMultipleHoldingRegistersFloat (None)					Writing multiple holding registers (32-bit floating point type)	P.36
MaskWriteHoldingRegister (MaskWrite4XRegister)	×	×	22 (0x16)	-	Holding register AND/OR masking output	P.36
ReadWriteMultipleHoldingRegisters (ReadWrite4XRegisters)	×	×	23 (0x17)	-	Read/Write multiple holding registers	P.37
AnotherFunctionCode (None)	×	Depends on Function Code	1-127 (0x01-0x7F)	-	Other function codes	P.37

2.4.4. Command supported by CaoExtension::Execute Modbus function (client mode only)

ReadMultipleDiscreteOutputs

Function Code	Broad cast
1 (0x01)	×

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

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Number of read points Range: 1-2000

Return Value **<Data>= VT_ARRAY | VT_BOOL: Read data**

Discrete Output (VARIANT_TRUE: DO) ON status

Discrete Output (VARIANT_FALSE: DO) is disabled.

Description Reads ON/OFF status of consecutive DOs (Discrete Output) from the starting address.

ReadMultipleDiscreteInputs

Function Code	Broad cast
2 (0x02)	×

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

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Number of read points: 1 to 2000

Return Value **<Data>= VT_ARRAY | VT_BOOL: Read data**

Discrete Input (VARIANT_TRUE: DI) ON status

Discrete Input (VARIANT_FALSE: DI) is disabled.

Description ON/OFF status of consecutive DIs (Discrete Input) is read from the starting address.

ReadMultipleHoldingRegisters

Function Code	Broad cast
3 (0x03)	×

Syntax ***Object.ReadMultipleHoldingRegisters(<Address>,<Count>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Read data count range: 1-125

Return Value **<Data>= VT_ARRAY | VT_UI2: Read data**

Description The contents of consecutive holding registers (16 bits) are read from the start address.

ReadMultipleHoldingRegistersLongInt

Function Code	Broad cast
3 (0x03)	×

Syntax ***Object.ReadMultipleHoldingRegistersLongInt(<Address>,<Count>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Read data count range: 1-62

Return Value **<Data>= VT_ARRAY | VT_I4: Read data**

Description The contents of consecutive holding registers (32-bit integer type) are read from the start address.

ReadMultipleHoldingRegistersFloat

Function Code	Broad cast
3 (0x03)	×

Syntax ***Object.ReadMultipleHoldingRegistersFloat(<Address>,<Count>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Read data count range: 1-62

Return Value **<Data>= VT_ARRAY | VT_R4: Read data**

Description Reads the contents of consecutive holding registers (32-bit floating point type) from the start address.

ReadMultipleInputRegisters

Function Code	Broad cast
4 (0x04)	×

Syntax ***Object.ReadMultipleInputRegisters(<Address>,<Count>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Number of read data (Range: 1-125)

Return Value **<Data>= VT_ARRAY | VT_UI2: Read data**

Description The contents of consecutive input registers (16 bits) are read from the start address.

ReadMultipleInputRegistersLongInt

Function Code	Broad cast
4 (0x04)	×

Syntax ***Object.ReadMultipleInputRegistersLongInt(<Address>,<Count>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Number of read data (Range: 1-62)

Return Value **<Data>= VT_ARRAY | VT_I4: Read data**

Description The contents of consecutive input registers (32-bit integer type) are read from the start address.

ReadMultipleInputRegistersFloat

Function Code	Broad cast
4 (0x04)	×

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

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Number of read data (Range: 1-62)

Return Value **<Data>= VT_ARRAY | VT_R4: Read data**

Description Reads the contents of consecutive input registers (32-bit floating point type) from the start address.

WriteSingleDiscreteOutput

Function Code	Broadcast	
	Eth	Com
5 (0x05)	×	✓

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

Argument **<Address>= VT_I4: Addresses**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Data>= VT_BOOL: Write data

ON: VARIANT_TRUE, OFF: VARIANT_FALSE

Return Value None

Description Updates the content of the DO (Discrete Output) at the specified address.

WriteSingleHoldingRegister

Function Code	Broadcast	
	Eth	Com
6 (0x06)	×	✓

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

Argument **<Address>= VT_I4: Addresses**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Data>= VT_UI2: Write data

Return Value None

Description Updates the contents of the holding register (16 bits) at the specified address.

ReadExceptionStatus

Function Code	Broad cast
7 (0x07)	×

Syntax ***Object.ReadExceptionStatus()***

Argument None

Return Value **<Data>= VT_UI1: Exception status**

Description Reads the exception status state.
The read exception status is assigned one bit at a time from the low-order 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: 1-250)**
<Query Data > = ARRAY|VT_UI1: Query data

Return Value **< Echo Data > = ARRAY | VT_UI1: Echo data**

Description The server echoes back the query data sent from the client as it is and sends it back to diagnose the communication line.
If successful, the return value <Echo Data> is the same value as the <Query Data> specified in the argument.

DiagnosticsRestartCommunicationsOption

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

Syntax ***Object.DiagnosticsRestartCommunicationsOption(<ClearEventLog>)***

Argument **<ClearEventLog>= VT_BOOL: Event log clear specification**
Clear Event Log: VARIANT_TRUE
Leaving the Event Log: VARIANT_FALSE

Return Value None

Description Initialize the communication port on the server side.
Also, specify whether to clear or leave the event log.

WriteMultipleDiscreteOutputs

Function Code	Broadcast	
	Eth	Com
15 (0x0F)	×	✓

Syntax ***Object.WriteMultipleDiscreteOutputs(<Address>,<Count>,<Data>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Number of exported points (Range: 1-1968)

<Data>= VT_ARRAY | VT_BOOL: ON/OFF

ON: VARIANT_TRUE, OFF: VARIANT_FALSE

Return Value None

Description The content of DO (Discrete Output) of consecutive points is updated from the starting address.

WriteMultipleHoldingRegisters

Function Code	Broadcast	
	Eth	Com
16 (0x10)	×	✓

Syntax ***Object. WriteMultipleHoldingRegisters(<Address>,<Data>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Write data count (Range: 1-123)

<Data>= VT_ARRAY | VT_UI2: Export data

Return Value None

Description Updates the contents of consecutive holding registers (16 bits) from the start address.

WriteMultipleHoldingRegistersLongInt

Function Code	Broadcast	
	Eth	Com
16 (0x10)	×	✓

Syntax ***Object.WriteMultipleHoldingRegistersLongInt(<Address>,<Data>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Write data count (Range: 1-61)

<Data>= VT_ARRAY | VT_I4: Export data

Return Value None

Description Updates the contents of consecutive holding registers (32-bit integer type) from the start address.

WriteMultipleHoldingRegistersFloat

Function Code	Broadcast	
	Eth	Com
16 (0x10)	×	✓

Syntax ***Object. WriteMultipleHoldingRegistersFloat(<Address>,<Count>,<Data>)***

Argument **<Address>= VT_I4: Starting address**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<Count>= VT_I4: Write data count (Range: 1-61)

<Data>= VT_ARRAY | VT_R4: Export data

Return Value None

Description Updates the contents of consecutive holding registers (32-bit floating point type) from the start address.

MaskWriteHoldingRegister

Function Code	Broad cast
22 (0x16)	×

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

Argument **<Address>= VT_I4: Addresses**

Range: When 1-65536 "OffsetAddressZero=False (Default)

0-65535 "OffsetAddressZero=True" hour

<AND_Mask> = VT_UI2: AND masked value

<OR_Mask> = VT_UI2: OR masked value

Return Value None

Description The current value of the holding register (16 bits) at the specified address, the AND mask value, and the OR mask value are combined for updating.
(Reference) Values to be updated are normally processed (calculated) by the server device according to the following equation.
[Updated 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: Read start address
Range: When 1-65536 "OffsetAddressZero=False (Default)
0-65535 "OffsetAddressZero=True" hour
<ReadCount>= VT_I4: Reads (Range: 1-125)
<WriteAddress>= VT_I4: Write start address
Range: When 1-65536 "OffsetAddressZero=False (Default)
0-65535 "OffsetAddressZero=True" hour
<WriteCount>= VT_I4: Writes (Range: 1-121)
<WriteData>= VT_ARRAY | VT_UI2: Export data

Return Value **<Data>= VT_ARRAY | VT_UI2: Read data**

Description **Reads and writes holding registers (16 bits).**
Writes <WriteCount> minutes <WriteData> to the address specified by <WriteAddress>, and reads <ReadCount> minutes of data from the address specified by <ReadAddress>.

AnotherFunctionCode

Function Code	Broadcast	
	Eth	Com
1-127 (0x01-0x7F)	×	Depends on FunctionCode

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

Argument

<FunctionCode>= VT_I4: Function code (Range: 1-127)
<RequestCount>= VT_I4: Number of data units transmitted (Range: 0 to 252)
<RequestData>= VT_ARRAY | VT_UI1: Data section transmit data
<ResponseCount>= VT_I4: Number of response data in the data section (range: 0 to 252)

Return Value

<ResponseData>= VT_ARRAY | VT_UI1: Data part response data

Description

Executes Modbus function code (not defined as a CaoController::Execute command).
For <RequestData>, specify only the information corresponding to the data part in Modbus query message corresponding to <FunctionCode> in binary for the number of <RequestCount> (the message check code is automatically added). Only the information corresponding to the data part in Modbus reply message is read into <ResponseData> in binary format for the number of <ResponseCount> (the message check code is automatically deleted).
[Notes]
1) If the device address at com is Broadcast (UnitAddress=0), the <ResponseCount> is ignored and <ResponseData> is VT_EMPTY because there is no response data from the server device.
2) If <ResponseCount> is set to 0, <ResponseData> is VT_EMPTY.
3) Except for the above cases 1) and 2), when <ResponseCount><> is the actual number of <ResponseData> data, the following is executed depending on the communication device.
On com: ended in error.
When eth is set: Normal termination is performed,
Return parameter is <ResponseCount><> Actual <ResponseData> data count

2.5. Variable list

2.5.1. CaoController classes

Table 2-14 CaoController class system variable list

Variable name	Data Type	Description	Attribute	
			Get	Put
@Version	VT_BSTR	Version information	✓	-
@Error	VT_I4	Gets the last error code that occurred.	✓	-
@IpInfo	VT_ARRAY| VT_VARIANT Or VT_EMPTY	Obtain the IP address and port number of the currently connected client device. Array[0]: VT_I4 Number of connected clients (0 to 16) Array[1]: When the first unit is connected: VT_ARRAY| VT_VARIANT When the first unit is not connected: VT_EMPTY Array[1][0]: VT_BSTR 1st IP-Address Array[1][1]: VT_I4 1st port No. – – – Array[17]: When 16th unit is connected: VT_ARRAY| VT_VARIANT When 16th unit is not connected: VT_EMPTY Array[17][0]: IP-Address of VT_BSTR 16 Array[17][1]: 16th VT_I4 port No. NOTE: In client mode or serial mode It will be VT_EMPTY.	✓	-

2.5.2. CaoExtension (client mode only)

The table below shows the user variables for Modbus function in client mode.

Table 2-15 Table List of User Variables Supported by CaoExtension Class-based Modbus Function

<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">Remarks</th></tr>
<tr><th>Get</th><th>Put</th></tr>
<tr><td>DO?※1</td><td>VT_I4※2</td><td>?Sets/gets the status (value) of the DO (DiscreteOutput) of the No.
A ? at the end of a variable name specifies a logical number as the address.
e.g., "DO1"
- Offset address ("OffsetAddressZero=False") 1 to-
Equivalent to the DO address "0" of the server device
- Offset address ("OffsetAddressZero=True") 0 to-
Equivalent to the DO address "1" of the server device

− The logical number range is the connection parameter of AddController.
With the offset address ("OffsetAddressZero") option
AddVariable Width ("UserVarWidth") options
The table below is based on the combination of these.

<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 - 65535</td><td>0 - 65528</td><td>0 - 65520</td><td>0 - 65504</td></tr>
<tr><td>False (default)</td><td>1 - 65536</td><td>1 - 65529</td><td>1 - 65521</td><td>1 - 65505</td></tr>
</table>

− The range of the set value/acquired value is the data width ("UserVarWidth").
The table below shows the options). When the data width is other than "1"
The notation is 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 set value/acquired value</td><td>0 - 1</td><td>0 - 255</td><td>0 -65535</td><td>-2147483648 - +2147483647</td></tr>
</table>

(Reference)
For FunctionCode sent and received over Modbus communication protocol,
For setting/acquiring and data-width ("UserVarWidth") options
Depending on the combination, the table below is displayed.

<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>Acquire (get)</td><td>1 (0x01)</td><td>1 (0x01)</td><td>1 (0x01)</td><td>1 (0x01)</td></tr>
<tr><td>Setting (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>
?Sets/gets the status (value) of the second DI (DiscreteInput).
A ? at the end of a variable name specifies a logical number as the address.
e.g., "DI1"
- Offset address ("OffsetAddressZero=False") 1 to-
Equivalent to the DI address "0" of the server device
- Offset address ("OffsetAddressZero=True") 0 to-
Equivalent to the DI address "1" of the server device

- The logical number range is the connection parameter of AddController.
With the offset address ("OffsetAddressZero") option
AddVariable Width ("UserVarWidth") options
The table below is based on the combination of these.

<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 - 65535</td><td>0 - 65528</td><td>0 - 65520</td><td>0 - 65504</td></tr>
<tr><td>False (default)</td><td>1 - 65536</td><td>1 - 65529</td><td>1 - 65521</td><td>1 - 65505</td></tr>
</table>

- The range of the set value/acquired value is the data width ("UserVarWidth").
The table below shows the options). When the data width is other than "1"
The notation is 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 set value/acquired value</td><td>0 - 1</td><td>0 - 255</td><td>0 - 65535</td><td>-2147483648 - +2147483647</td></tr>
</table>

(Reference)
For FunctionCode sent and received over Modbus communication protocol,
For setting/acquiring and data-width ("UserVarWidth") options
Depending on the combination, the table below is displayed.

<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>Acquire (get)</td><td>2 (0x02)</td><td>2 (0x02)</td><td>2 (0x02)</td><td>2 (0x02)</td></tr>
<tr><td>Setting (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>
?Sets/gets the value of the second holding register as an integer.
A ? at the end of a variable name specifies a logical number as the register address.
e.g., "HRI1"
- Offset address ("OffsetAddressZero=False") 1 to-
Corresponds to the holding register address "0" of the server device.
- Offset address ("OffsetAddressZero=True") 0 to-
Corresponds to the holding register address "1" of the server device.

− The logical number range is the connection parameter of AddController.
With the offset address ("OffsetAddressZero") option
AddVariable Width ("UserVarWidth") options
The table below is based on the combination of these.
<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 - 65535</td><td>0 -65534</td></tr>
<tr><td>False (Default)</td><td>1 - 65536</td><td>1 -65535</td></tr>
</table>

− The range of the set value/acquired value is the data width ("UserVarWidth").
The table below is based on the options.
<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 set value/acquired value</td><td>0 - 65535</td><td>-2147483648 - +2147483647</td></tr>
</table>

(Reference)
For FunctionCode sent and received over Modbus communication protocol,
For setting/acquiring and data-width ("UserVarWidth") options
Depending on the combination, the table below is displayed.
<table>
<tr><th rowspan="2"></th><th colspan="2">Data-width (UserVarWidth)</th></tr>
<tr><th>16</th><th>32</th></tr>
<tr><td>Acquire (get)</td><td>3 (0x03)</td><td>3 (0x03)</td></tr>
<tr><td>Setting (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>?Sets/gets the value of the second holding register in 32-bit floating point type.
A ? at the end of a variable name specifies a logical number as the register address.
e.g., "HRF1"
- Offset address ("OffsetAddressZero=False") 1 to-
Corresponds to the holding register address "0" of the server device.
- Offset address ("OffsetAddressZero=True") 0 to-
Corresponds to the holding register address "1" of the server device.

- The logical number range is the connection parameter of AddController.
With the offset address ("OffsetAddressZero") option
AddVariable Width ("UserVarWidth") options
The table below is based on the combination of these.

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

(Reference)
For FunctionCode sent and received over Modbus communication protocol,
For setting/acquiring and data-width ("UserVarWidth") options
Depending on the combination, the table below is displayed.

<table>
<tr><th rowspan="2"></th><th>Data-width (UserVarWidth)</th></tr>
<tr><th>32 fixed</th></tr>
<tr><td>Acquire (get)</td><td>3 (0x03)</td></tr>
<tr><td>Setting (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>
? Gets the value of the first input register as an integer.
A ? at the end of a variable name specifies a logical number as the register address.
e.g., "IRI1"
- Offset address ("OffsetAddressZero=False") 1 to-
Corresponds to the input register address "0" of the server device.
- Offset address ("OffsetAddressZero=True") 0 to-
Corresponds to the input register address "1" of the server device.

- The logical number range is the connection parameter of AddController.
With the offset address ("OffsetAddressZero") option
AddVariable Width ("UserVarWidth") options
The table below is based on the combination of these.
<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 - 65535</td><td>0 - 65534</td></tr>
<tr><td>False (Default)</td><td>1 - 65536</td><td>1 - 65535</td></tr>
</table>

- The range of the set value/acquired value is the data width ("UserVarWidth").
The table below is based on the options.
<table>
<tr><th></th><th colspan="2">Data-width (UserVarWidth)</th></tr>
<tr><th></th><th>16</th><th>32</th></tr>
<tr><td>Range of set value/acquired value</td><td>0 - 65535</td><td>-2147483648 - +2147483647</td></tr>
</table>

(Reference)
For FunctionCode sent and received over Modbus communication protocol,
For setting/acquiring and data-width ("UserVarWidth") options
Depending on the combination, the table below is displayed.
<table>
<tr><th></th><th colspan="2">Data-width (UserVarWidth)</th></tr>
<tr><th></th><th>16</th><th>32</th></tr>
<tr><td>Acquire (get)</td><td>4 (0x04)</td><td>4 (0x04)</td></tr>
<tr><td>Setting (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>?Gets the value of the first input register as a 32-bit floating point type.
A ? at the end of a variable name specifies a logical number as the register address.
e.g.) "IRF1"
- Offset address ("OffsetAddressZero=False") 1 to-
Corresponds to the input register address "0" of the server device.
- Offset address ("OffsetAddressZero=True") 0 to-
Corresponds to input register address "1" on the server device side.

- The logical number range is the connection parameter of AddController.
With the offset address ("OffsetAddressZero") option
AddVariable Width ("UserVarWidth") options
The table below is based on the combination of these.

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

(Reference)
For FunctionCode sent and received over Modbus communication protocol,
For setting/acquiring and data-width ("UserVarWidth") options
Depending on the combination, the table below is displayed.

<table>
<tr><th></th><th>Data-width (UserVarWidth)</th></tr>
<tr><th></th><th>32 fixed</th></tr>
<tr><td>Acquire (get)</td><td>4 (0x04)</td></tr>
<tr><td>Setting (put)</td><td>-</td></tr>
</table>
</td>
<td>✓</td>
<td>-</td>
<td></td>
</tr>
</table>

※1: When the "VT" option is enabled, the range of logical numbers varies depending on the variable type. See Table 2-6 for details.

※2: Variable types differ when the "VT" or "Elem" option is enabled. See Table 2-5 and Table 2-6 for details.

2.6. Error code

Modbus.X providers define the following unique error codes:

Table 2-16 List of unique error codes

Error name	Error Number	Description
E_CAOP_SYNC_ONLY	0x80100001	It can only be executed in synchronous mode.
E_CAOP_ASYNC_ONLY	0x80100002	It can only be executed in asynchronous mode.
E_CAOP_CLIENT_ONLY	0x80100003	It can only be executed in client mode.
E_CAOP_SERVER_ONLY	0x80100004	It can only be executed in server mode.
E_CAOP_SERVER_SEND_REPLY_TIMEOUT	0x80100005	This error occurs when a request (query) message is received in server mode and a response (reply) is sent after the time specified by the "SendReplyTimeout" option has elapsed.
E_CAOP_ILLEGAL_ARGUMENT	0x80100F01	Argument error. The parameter passed to the command that returns this error code is invalid or out of range.
E_CAOP_ILLEGAL_STATE	0x80100F02	Status error. The function is called in the wrong state. If the protocol has not yet been opened successfully, this return code is returned by all functions.
E_CAOP_ILLEGAL_SLAVE_ADDRESS	0x80100F05	Invalid server device address. Address 0 was used by a function that does not support broadcast.

E_CAOP_OPEN	0x80100F42	Port or socket open error. Could not open TCP/IP socket or serial port. For a serial port, the serial port may not exist in the system.
E_CAOP_FTALK_PORT_ALREADY_OPEN	0x80100F43	The serial port is already open. The serial port defined for the open operation is already opened in another application.
E_CAOP_FTALK_TCPIP_CONNECT	0x80100F44	TCP/IP connect failure. Could not establish TCP/IP connectivity. This error usually occurs if the host is on a network or IP address, or if the host is named incorrectly. The remote host must Listen the appropriate Port number.
E_CAOP_CONNECTION_WAS_CLOSED	0x80100F45	The remote peer has closed TCP/IP connect. Indicates that TCP/IP connectivity was closed or corrupted by a remote peer.
E_CAOP_SOCKET_LIB	0x80100F46	Socket library error. TCP/IP socket library (e.g. WINSOCK) could not be loaded. The DLL may not be found or installed.
E_CAOP_PORT_ALREADY_BOUND	0x80100F47	The TCP port is already bound. Indicates that the specified TCP port cannot be bound. Port may have been taken by another application or not yet released by TCP/IP stackfor reuse.
E_CAOP_LISTEN_FAILED	0x80100F48	Lisen failed. The specified TCP-port Listen failed.
E_CAOP_FILEDES_EXCEEDED	0x80100F49	File descriptor exceeded. The maximum number of available file descriptors has been exceeded.
E_CAOP_PORT_NO_ACCESS	0x80100F4A	You do not have permission to access the serial port or TCP port. For a serial port, change the access privilege. For TCP/ IP, the TCP-port number Out of

		IPPORT_RESERVED.
E_CAOP_PORT_NOT_AVAIL	0x80100F4B	The TCP port cannot be used. The specified TCP port is not available in this operating environment.
E_CAOP_LINE_BUSY	0x80100F4C	Serial line is busy. The serial line is receiving noise, etc., despite the fact that there must not be traffic.
E_CAOP_CHECKSUM	0x80100F81	Checksum error The checksum of the received frame is invalid.
E_CAOP_INVALID_FRAME	0x80100F82	Invalid frame error. Indicates that the received frame is not supported by any structure or content of the communication protocol or does not match the frame of the previously transmitted query.
E_CAOP_INVALID_REPLY	0x80100F83	Invalid response error. Notifies that the received response frame does not support the communication protocol.
E_CAOP_REPLY_TIMEOUT	0x80100F84	Timeout error This may occur when the server device does not respond in time or does not respond at all. The wrong server device address triggers this error.
E_CAOP_SEND_TIMEOUT	0x80100F85	Transmit timeout error. Indicates that data transmission has timed out. This can occur if the handshake line is not set correctly.
E_CAOP_INVALID_MBAP_ID	0x80100F86	Invalid identifier. Protocol or transaction identifier, but incorrect. The TCP server device must return an identifier received from the TCP client.
E_CAOP_MBUS_EXCEPTION_RESPONSE	0x80100FA0	Notifies that a Modbus exception response message has been received.

E_CAOP_MBUS_ILLEGAL_FUNCTION_RESPONSE	0x80100FA1	Modbus Indicates that an invalid function-exception response (code 01) has been received.
E_CAOP_MBUS_MBUS_ILLEGAL_ADDRESS_RESPONSE	0x80100FA2	Indicates that a Modbus illegal data address exception response (code 02) has been received.
E_CAOP_MBUS_ILLEGAL_VALUE_RESPONSE	0x80100FA3	Modbus Indicates that an invalid-value exception response (code 03) was received.
E_CAOP__MBUS_SLAVE_FAILURE_RESPONSE	0x80100FA4	This bit notifies that a Modbus slave-failed exception response (code04) was received.

- If a Windows system error occurs, the error number is masked with "0x8010000".

e.g.: System error of Windows: 2 (0x0002) Error of → CAO API: 0x80100002

For information about ORiN2 common errors, see the Error Codes section of ORiN2 Programming Guide (Link).

3. Sample program

3.1. Client mode

RS232C/RS485 Devices: After connecting with COM1,

- For a device with server device address = 1, the user variable "DO1" is defined and ON or OFF is output (set) to address 1 of DO (DiscreateOutput).
- For a device with server device address = 2, the user variable "DI1" is defined as 8-bit wide,
  The status of addresses 1 to 8 of the DI (DiscreateInput) is acquired as byte-data in the MSB (Most Significant Bit).

Shows the sample to be used.

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=COM:1")
    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. Synchronous Mode Sample

After connecting in server mode, synchronous mode, TCP communication mode,

The following shows an example of using Timer event (Timer1) to receive a request (query) message from a client device at 100ms intervals using CaoController::Execute" ReceiveQuery "command" and reply with the "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)
As Integer ' Holding Register for Private m_iHR (MEM_ARRAY_SIZE-1) (16-bit)
As Integer ' Holding Register for Private m_iIR (MEM_ARRAY_SIZE-1) (16-bit)
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, Conn=eth:0.0.0.0:502")
  Timer1.Interval = 100
  Timer1.Enabled = True

End Sub

Private Sub Timer1_Timer()

  Dim vntArg As Variant
  Dim vntQueryData As Variant

  ' Execute "ReceiveQuery" command
  VntQueryData = m_caoCtrl.Execute("ReceiveQuery", vntArg)

  If IsEmpty(vntQueryData) Then
      Exit Sub
  End If

  ' Query Data Analysis & Reply Data Generation
  VntArg = AnalyzeQueryDataToCreateReplyData(vntQueryData)

  If VarType(vntArg) <> vbEmpty Then
      ' Execute "SendReply" command
      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
'Read more than one DO(Discrete Output)(1)
'Read more than one DI(Discrete Input)(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 error
                bResult = False
            End If
        Else
            m_byExpSts = 2 ' exception status 2: score error
            bResult = False
        End If
    Else
        Exit Function
    End If

' Holding register (16 bits) Multiple reads (3)
' Input register (16 bits) Multiple reads (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 error
                bResult = False
            End If
```

```
                Else
                    m_byExpSts = 2 ' exception status 2: score error
                    bResult = False
                End If
            Else
                Exit Function
            End If

        ' Read exception status 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-write (15)
        ' Holding register (16 bits) Multiple write (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: Write point unmatch error
                            bResult = False
                        End If
                    Else
                        m_byExpSts = 4 'Exception status 4: Write data type unmatch error
                        bResult = False
                    End If
                Else
                    m_byExpSts = 3 'Exception status 3: Address range error
                    bResult = False
                End If
            Else
                m_byExpSts = 2 ' exception status 2: score error
                bResult = False
            End If
        Case Else
                m_byExpSts = 1 'Exception status 1: Notification function code error
                bResult = False
        End Select

        AnalyzeQueryDataToCreateReplyData = Array(lFuncCode, bResult, vntData)

    End Function
```

3.2.2. Asynchronous Mode Sample

The following example shows how to receive a request (query) message from a client device with CaoController::OnMessage "QUERY_MSG message" and return it with Message::Reply() method after connecting with the server mode, asynchronous mode, TCP communication mode.

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)
As Integer ' Holding Register for Private m_iHR (MEM_ARRAY_SIZE-1) (16-bit)
As Integer ' Holding Register for Private m_iIR (MEM_ARRAY_SIZE-1) (16-bit)
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, Conn=eth:0.0.0.0")

End Sub

Private Sub m_caoCtrl_OnMessage(ByVal pICaoMess As CAOLib.ICaoMessage)

    Select Case pICaoMess.Number
    Case MSG_ID_QUERY_MSG

        ' Query Data Analysis & Reply Data Generation
        Dim vntReply As Variant
        VntReply = AnalyzeQueryDataToCreateReplyData(pICaoMess.Value)
        PutLogQueryMsg pICaoMess

        If VarType(vntReply) <> vbEmpty Then
            ' Reply (reply) process
            pICaoMess.Reply vntReply
            PutLogReplyData vntReply
        End If

    End Select

    Exit Sub

End Sub
```

4. Appendix

4.1. Comparison with previous Modbus command-name

The following table shows the comparison between the old Modbus command name and the new command name.

Table 4-1 Comparison table with previous Modbus command name

Old command name	New command name	Function Code (HEX)	Page
ReadCoilStatus	ReadMultipleDiscreteOutputs	1 (0x01)	P.30
ReadInputStatus	ReadMultipleDiscreteInputs	2 (0x02)	P.30
ReadHoldingRegister	ReadMultipleHoldingRegisters	3 (0x03)	P.30
ReadInputRegister	ReadMultipleInputRegisters	4 (0x04)	P.32
ForceSingleCoil	WriteSingleDiscreteOutput	5 (0x05)	P.33
PresetSingleRegister	WriteSingleHoldingRegister	6 (0x06)	P.33
ReadExceptionStatus	Same as the old command name	7 (0x07)	P.33
DiagnosticsReturnQueryData	↑	8 (0x08) - 0	P.34
DiagnosticsRestartCommunicationsOption	↑	8 (0x08) - 1	P.34
ForceMultipleCoils	WriteMultipleDiscreteOutputs	15 (0x0F)	P.35
PresetMultipleRegisters	WriteMultipleHoldingRegisters	16 (0x10)	P.35
MaskWrite4XRegister	MaskWriteHoldingRegister	22 (0x16)	P.36
ReadWrite4XRegisters)	ReadWriteMultipleHoldingRegisters	23 (0x17)	P.37
FetchCommEventCounter	AnotherFunctionCode 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