SHIBAURA MACHINE
NCBOY providers
User's Guide

Version 1.0.0

April 16, 2021

NOTE:

[Revision History]

Version	Date	Description
1.0.0	2021-04-16	First edition

［Compatible models］

Model	Version	Notes
NCBOY-200	-	-
NCBOY-3200	-	-

［Operation confirmed models］

Model	Version	Notes
NCBOY-3200	VNAM=07.09 VTA=07.09 TERA=01.01	-

Contents

1. Introduction

This user's guide registers register names in the register table for NCBOY of SHIBAURA MACHINE CO., LTD. and obtains the corresponding register data. Fig. 1-1 shows the overall configuration of this provider and the device. The providers are referred to as NCBOY providers.

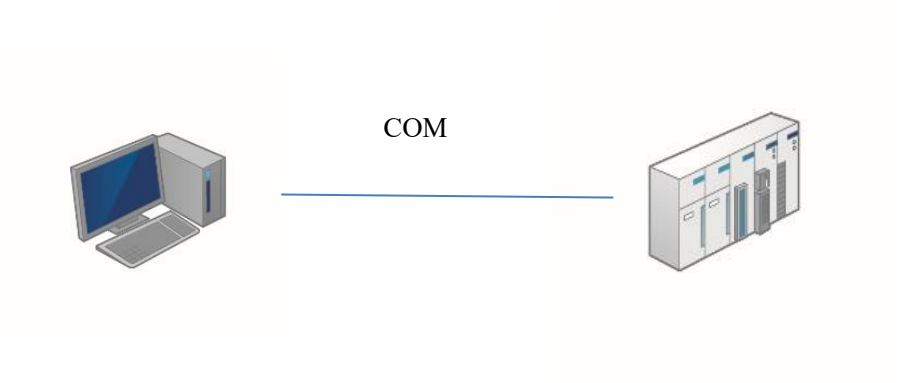

Fig. 1-1 Configuration Diagram

Fig. 1-2 shows the correspondence between this provider and each device.
(* This is an example. It does not represent everything.)

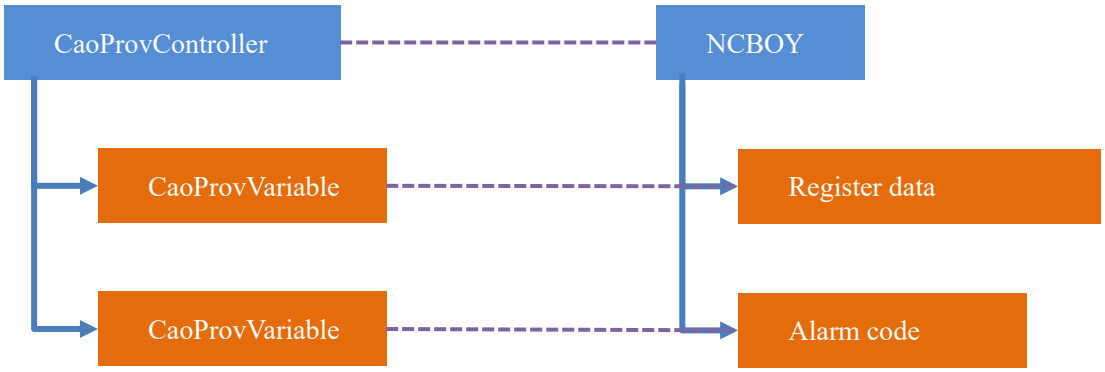

Fig. 1-2 Provider configuration and device information

2. Setting Up Your Environment for Application Development

2.1. Connecting NCBOY to a ClientPC

This provider communicates with NCBOY unit using a RS-232C compliant COM connection. Connect the communication cable to port number 2 of NCBOY.

3. Command Reference

3.1. Method/Property List

Table 3-1 List of methods and properties

Category	Methods/Properties[1]		Function	See Also
CaoWorkspace				
	AddController	M	Connected to controller	P.8
CaoController				
	Variables	P	Retrieving Variable Collections Holded by the Controller	P.9
	AddVariable	M	Adding Variable Objects	P.10
CaoVariable				
	Value	P	Get/set value	P.10

3.2. Method properties

3.2.1. CaoWorkspace classes

3.2.1.1. AddController method

In CaoWorkspace, add a controller object. NCBOY providers connect to the appropriate NCBOY by referring to the parameters passed when AddController method is executed. The following are the specifics of AddController method:

SYNOPSIS

```
AddController
(
    "<Controller name>",                    // Controller name (optional)
    "CaoProv.SHIBAURAMACHINE.NCBOY",        // Provider name (fixed)
    "<Machine name>",                       // Provider execution machine name (unused)
    "<Option>"                              // Option character string (optional)
)
```

Option

The following is an optional specification for Option character string: Option character string is a comma (,) string consisting of the options listed below.

[1] M:Indicates methods, P: properties, and E: events, respectively.

Option	Required	Description	Value Range	Default Value
Conn	✓	Specifies the communication parameters for COM-connection with NCBOY. Refer to 3.2.1.1.1 for details.		
Timeout	-	Specifies the wait time (ms) between the connection destination and the timeout for one communication.	0 ~	2000

3.2.1.1.1. CONN Optional

The following is a Conn optional connection parameter string: Here, braces ("[]") are optional, and the underlined part in the description of each parameter indicates the default value when no options are specified.

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

<COM Port>	:	COM port number . '1' -COM1, '2' - COM2, …
<BaudRate>	:	Baud rate. 4800, 9600, 19200, 38400, 57600, 115200
<Parity>	:	Parity . 'N'-NONE, 'E'-EVEN, 'O'-ODD
<DataBits>	:	Number of data bits. '7'-7bit, '8'-8bit
<StopBits>	:	No. of Stop Bits . '1'-1bit, '2'-2bit
<Flow>	:	Flow control. '0'-None, '1'-Xon/Xoff, '2'-Hardware control It can be specified by taking OR.

Example (C#)

```
// Engine
ORiN2.ManagedCAO.CCaoEngine engine = new ORiN2.ManagedCAO.CCaoEngine();
// Workspace
ORiN2.ManagedCAO.CCaoWorkspace workspace = engine.AddWorkspace("NewWrks", "");
// Controller
ORiN2.ManagedCAO.CCaoController controller= workspace.AddController("NCBOYSample",
                                            "CaoProv.SHIBAURAMACHINE.NCBOY",
                                            "",
                                            "Conn = COM:1, Timeout = 5000");
```

3.2.2. CaoController classes

3.2.2.1. Variables Properties

Gets a collection of variables that the controller holds.

Example (C#)

```
// Variable Collection Retrieval
```

```
ORiN2.ManagedCAO.CCaoVariables variables = controller.Variables;
// Variable acquisition
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

3.2.2.2. AddVariable method

Adds a variable object to CaoController. Only the variable names shown in 3.3.2 can be used.

AddVariable is specifi ed as follows.

SYNOPSIS

```
AddVariable
(
    "<Variable name>",      // Variable name
    "<Option>"              // Option character string (optional)
)
```

3.2.3. CaoVariable classes

3.2.3.1. Value Properties

Retrieves data from the connected NCBOY. The behavior depends on the variable name. For details, refer to section 3.3, Variable list.

3.3. Variable list

Defines a list of variables that can be used in each class. Variables refer to objects of CaoVariable classes.

3.3.1. System and User Variables

There are two types of variables in the provider: system variables and user variables.

System Variables

A variable that accesses only the information in the object that holds the variable. System variables are often static data. System variables are characterized by a leading "@" in their names. For more information, see 3.3.2.

User variable

A variable that you can use to specify what information you want to access when you create a variable by using Option character string. Due to its nature, user variables can be given arbitrary strings to register multiple variables (useful if you only want to change options, etc.). The content of user-variable of this provider varies depending on Type option. See 3.3.3 for user variables.

3.3.2. CaoController class system variable

Variable name	Description	Value	See Also

		Get	Put	
@MAKER_NAME	Obtain the manufacturer's name.	✓	-	P.11
@VERSION	Get the DLL version.	✓	-	P.11
@ALARM	Gets the alarm code currently detected in NOBOY.	✓	-	P.11
@DEVICE_VERSION	Gets the software version of the CPUs in NCBOY.	✓	-	P.12
@SISTEMINFO	Acquires the information of each unit and the information of the amplifier.	✓	-	P.13

3.3.2.1. @MAKER_NAME

Obtain the manufacturer's name.

Data Type

Type Description	
VT_BSTR	Holds the manufacturer name.

Example (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MAKER_NAME","");
// Acquisition of Values
string value = var.Value as string;
```

3.3.2.2. @VERSION

Gets the DLL version.

Data Type

Type Description	
VT_BSTR	Holds the version of the DLL. *.*.*

Example (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@VERSION","");
// Acquisition of Values
string value = var.Value as string;
```

3.3.2.3. @ALARM

Gets a list of 32 alarm codes currently detected in NCBOY.

Data Type

Type Description	
VT_UI2 | VT_ARRAY	
i[2]	Holds the detected alarm number. ※If the number of alarms is less than 32, 0 is stored. When the number of alarms is 2, the content of the array is returned as "(Alarm 1), (Alarm 2), 0,0,0,0... 0,0".

Example (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@ALARM","");
// Acquisition of Values
ushort[] alarms = var.Value as ushort[];
```

3.3.2.4. @DEVICE_VERSION

Gets the software version of the CPUs in NCBOY.

Data Type

Type Description	
VT_BSTR | VT_ARRAY	
VT_BSTR	Keeps VNAM revision. **.**
VT_BSTR	Keeps the VTA version. **.**
VT_BSTR	Keeps TARA revision. **.**

Example (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@DEVICE_VERSION","");
// Acquisition of Values
string[] versions = var.Value as string[];
// Decompose into versions
string vnamVer = versions [0];
string vtaVer = versions[1];
string taraVer = versions[2];
```

[2] $0 \leq i \leq 31$

3.3.2.5. @SYSTEMINFO

Acquires the information of each unit and the information of the amplifier.

Data Type

Type Description				
VT_VARIANT | VT_ARRAY				
0	VT_UI1			Holds the system decision flag. Value: 0 = NCBOY-200, 1 = NCBOY-3200
1	VT_UI4			The unit distribution cycle [μs] is retained.
2	VT_UI2			Holds the total number of units.
3	VT_UI2			Holds the total number of axes.
4	VT_ARRAY | VT_VARIANT			Holds unit information for the total number of units.
	n[3]	VT_ARRAY | VT_UI2		
		0	VT_UI2	Holds the decimal point position of each unit.
		1	VT_UI2	Holds the number of axes of each unit.
5	VT_ARRAY | VT_VARIANT			Get axis information. In this array, the axis information of all units is stored in ascending order of unit number.
	m[4]	VT_ARRAY | VT_BSTR		
		0	VT_BSTR	The axis name is retained.
		1	VT_BSTR	Keeps the amplifier version. **.**

Example (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@SYSTEMINFO", "");
// Acquisition of Values
object[] systemInfo = var.Value as object[];
// Break down into each item
byte? systemFlag = systemInfo[0] as byte?;
uint? unitDstributionCycle = systemInfo[1] as uint?;
ushort? unitCount = systemInfo[2] as ushort?;
```

[3] $0 \leq n \leq$ the total number of units
[4] $0 \leq m \leq$ the total number of axes

```
object[] unitInfos = systemInfo[4] as object[];
// Example of acquiring unit information
foreach (var unitInfoObject in unitInfos)
{
    UInt16[] unitInfo = unitInfoObject as UInt16[];
    UInt16 decimalPointPos = unitInfo[0];
    UInt16 axisCountPerUnit1 = unitInfo[1];
}

// Example of acquiring axis names for each unit
List<string>[] axisNames = new List<string>[unitCount.Value];
ushort? axisCount = systemInfo[3] as ushort?;     // Total number of axes
int unitNo = 0;                                   // Unit number
int axisNoForUnit = 0;                            // Unit axis No.
ushort? axisCountPerUnit = 0;                     // Number of axes in unit
foreach (var axisInfoObject in (systemInfo[5] as object[]))
{
    string[] axisInfo = axisInfoObject as string[];
    // When switching units
    if (axisNoForUnit == 0)
    {
        axisNames[unitNo] = new List<string>();
        axisCountPerUnit = (unitInfos[unitNo] as UInt16[])[1] as ushort?;
    }

    string axisName = axisInfo[0] as string;
    axisNames[unitNo].Add(axisName);

    // Increment the unit count and switch to the next unit if necessary.
    axisNoForUnit++;
    if (axisNoForUnit == axisCountPerUnit)
    {
        axisNoForUnit = 0;
        axisCountPerUnit = 0;
        unitNo++;
    }
}
```

3.3.3. CaoController Class User Variables

Variable name	Description	Value		See Also
		Get	Put	
<Register table variable>	Gets the value of the register. Up to 16 register values can be obtained for each variable. Up to 32 register variables can be created. (Fixed to Type option-0)	✓	-	P.15
<Alarm history>	Retrieves any alarm history. (Fixed to Type Option-1)	✓	-	P.17

3.3.3.1. <register table> variable

This variable acquires the values of the registers specified by Register option in batch mode.

When creating this variable, the provider sets a table (called a register table) on NCBOY to handle multiple registers in a batch, and then accesses the register values through that table.

If the table number specified at variable creation matches the table number of an existing variable, an error is returned to prevent changes to the register table accessed by the existing variable.

Note that the register table is used only to communicate between providers and NCBOY, and does not affect the operation of NCBOY.

Option	Required	Description	Value Range
Type	✓	Fixed to 0	--
Table	✓	Specifies the register table number to be used. The table number must be unique.	1 ~ 32
Register	✓	Specify up to 16 registers to be acquired. ※Separate each register name with a colon (":"). e.g., Register=H123:HB456:HW789	--

Data Type

Type Description
VT_ARRAY | VT_VARIANT

<table>
<tr>
<td>i[5]</td>
<td>VT_VARIANT</td>
<td>The data corresponding to the registered register name is retained.
For the data type of VT_VARIANT corresponding to the data type of each register, refer to the following correspondence tables.
<table>
<tr><th>Register Data Type</th><th>Supported Types</th></tr>
<tr><td>Register name error</td><td>VT_EMPTY</td></tr>
<tr><td>Bit register</td><td>VT_BOOL</td></tr>
<tr><td>Byte register (signed)</td><td>VT_I1</td></tr>
<tr><td>Byte register (unsigned)</td><td>VT_UI1</td></tr>
<tr><td>Byte register (Hex)</td><td>VT_UI1</td></tr>
<tr><td>Word register (signed)</td><td>VT_I2</td></tr>
<tr><td>Word register (unsigned)</td><td>VT_UI2</td></tr>
<tr><td>Word register (Hex)</td><td>VT_UI2</td></tr>
<tr><td>Long register (signed)</td><td>VT_I4</td></tr>
<tr><td>Long register (unsigned)</td><td>VT_UI4</td></tr>
<tr><td>Long register (Hex)</td><td>VT_UI4</td></tr>
<tr><td>Timer register</td><td>VT_UI4
1st byte: ON/OFF info
This bit is set to 1 when the timer is up or counted up, and to 0 when it is not.
2nd byte: Status information
0 = Not used
1 = Counter preset
2 = Counting in progress
3 = Count up
3rd and 4th bytes: Count data
Remaining timer value or remaining count value</td></tr>
<tr><td>Floating-point register</td><td>VT_R4</td></tr>
</table>
</td>
</tr>
</table>

Example (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("TABLE01","Type=0, Table=1, Register=Reg1:Reg2:Reg3");
// Acquisition of Values
object[] value = var.Value as object[];      //as object[]
```

[5] $0 \leq i \leq 15$

```
byte? reg1 = value[0] as byte?;          //Reg1 assumes signed byte register
short? reg2 = value[1] as short?;        //Reg2 assumes signed word register
int? reg3 = value[2] as int?;            //Reg3 assumes signed long register
```

3.3.3.2. <alarm history> variable

Acquires the alarm history of the number of data items you want to acquire from the specified alarm history number.

If the alarm history number to be acquired exceeds 255, it returns to the first alarm history number to acquire the data.

Example) If 24 alarm history numbers are specified from 250, the alarm history of 250 to 255 and 0 to 17 will be acquired.

Option	Required	Description	Value Range
Type	✓	1 fixed	--
No	✓	Specifies the starting position of the alarm history to be acquired.	0 ~ 255
Elem	✓	Specify the number of alarm histories to be acquired.	1 ~ 24

Data Type

Type Description			
VT_ARRAY | VT_VARIANT			
n[6]	VT_ARRAY | VT_VARIANT		Alarm history Holds one piece of information. VT_EMPTY if the alarm history does not exist.
	0	VT_UI2	Holds the alarm code.
	1	VT_DATE	Keeps the alarm history.

Example (C#)

```
// Add Variable
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("ALARMHISTORY","Type=1, No=0, Elem=24");
// Acquisition of Values
object[] alarmHIstory = var.Value as object[];
foreach (var alarmObject in alarmHIstory)
{
```

[6] $0 \leq i \leq the$ elem option

```
        object[] alarmInfo = alarmObject as object[];
        if (alarmInfo != null)
        {
            ushort? alarmCode = alarmInfo[0] as ushort?;
            DateTime? dateTime = alarmInfo[1] as DateTime?;
        }
    }
```

4. NCBOY Programming by provider

With NCBOY providers, you can connect NCBOY to the client computer as follows:

- Creating a CaoEngine
- Creating a CaoWorkspace
- Creating a CaoController

After connecting to NCBOY, CaoVariable can be accessed by creating a new.

4.1. Sample programming to acquire data in registers

This section describes a sample program for acquiring data in some registers in NCBOY as an example.

Table 4-1 Sample program requirements

Requirements	Description
Host	Connect with a COM
	The host COM port number is 1.
Process Description	Set the registers "H000", "HB000" and "HW000" to the table.
	Acquires data from the register list of the set table.

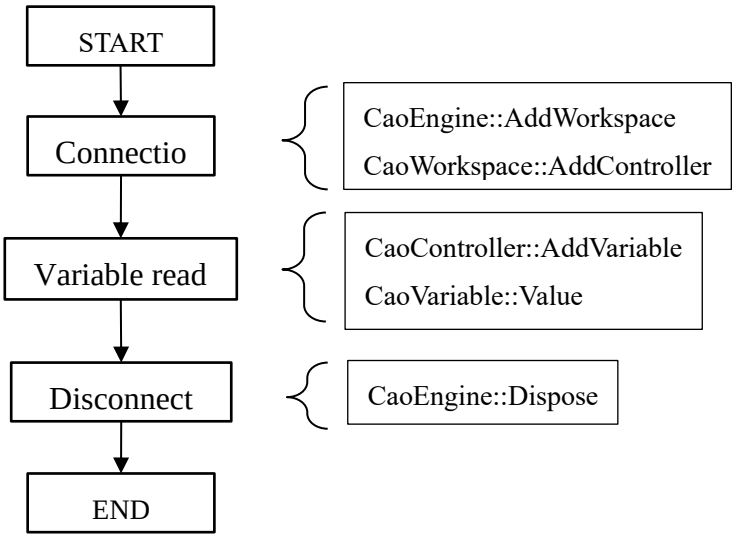

Fig. 4-1 Flow of data acquisition

Specific codes are given in the following sections.

4.1.1. Sample program

The following is an overview of the sample program.

Sample **Sample.cs**

```
// Object
```

```
    private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
    private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
    private ORiN2.ManagedCAO.CCaoController m_caoController = null;
    private ORiN2.ManagedCAO.CCaoVariable m_varRegTable = null;

    public void Main()
    {
        // Connection
        this.Connect();
        //Add register table variable
        this.m_varRegTable = this.m_caoController.AddVariable("RegTable1","Type=0,Table=1,Reg
ister=H000:HB000:HW000");

        object[] tableData = this.m_varRegTable.Value as object[];
        // *** register data acquisition (bit type)
        bool? bData = tableData[0] as bool?;
        // *** register data acquisition (byte type)
        byte? bytData = tableData[1] as byte?;
        // *** register data acquisition (short type)
        short? iData = tableData[2] as short?;
        // Disconnect
        this.Disconnect();
    }

    // Connection method
    private void Connect()
    {
        // Generate CaoEngine object
        this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
        // Generate CaoWorkspace object
        this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
        // Generate CaoController object
        this.m_caoController = this.m_caoWorkspace.AddController("NCBOYSample",
                                                    "CaoProv.SHIBAURAMACHINE.NCBOY",
                                                "",
                                                "Conn=COM:1");
    }
```

```
// Disconnect method
private void Disconnect()
{
    this.m_caoEngine.Dispose();
    this.m_caoEngine = null;
}
```

4.1.1.1. Connection

Follow the steps below to connect to NCBOY.

(1) Prepare a variable to hold the object.

The objects required to connect are CaoEngine object, CaoWorkspace object, and CaoController object. CaoWorkpace object does not need to have variables in order to retrieve CaoController object from CaoWorkspaces. You will also need a CaoVariable for accessing the variable. The following is a code example for C#.

Example (C#)

```
// Variables for CaoEngine
private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
// Variables for CaoWorkspace
private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
// Variables for CaoController
private ORiN2.ManagedCAO.CCaoController m_caoController = null;
// Variables for CaoVariable
private ORiN2.ManagedCAO.CCaoVariable m_varRegTable = null;
```

(2) Creates a CaoEngine.

CaoEngine is generated using the New keyword.

Example (C#)

```
// Generate CaoEngine object
this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
```

(3) Gets or generates a CaoWorkspace container.

When you create a CaoEngine object, it defaults to generating one CaoWorkspaces object and one CaoWorkspace object. The following is a sample code/default CaoWorkspace for creating a new CaoWorkspace.

Example (C#)

```
// Generate CaoWorkspace object
this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
```

(4) Creates a CaoController.

To generate a CaoController object, set the provider name to use and the parameters to use. For NCBOY providers, specify the destination with Conn options. The following is a code example:

Example (C#)

```
// Generate CaoController object
this.m_caoController = this.m_caoWorkspace.AddController("NCBOYSample",
                                                          "CaoProv.SHIBAURAMACHINE.NCBOY",
                                                          "",
                                                          "Conn=COM:1");
```

4.1.1.2. Register Data Acquisition

(1) Creates a CaoVariable that accesses the register table.

Create the <register table> variable by specifying the register names "H000", "HB000", and "HW000" for register table 1.

※For the register name, specify the register name actually set in NCBOY.

Example (C#)

```
this.m_varRegTable = this.m_caoController.AddVariable("RegTable1","Type=0,Table=1,Register=H000:HB000:HW000");
```

(2) Acquires data from the added <register table> variable.

When retrieving, use Value property of CaoVariable object. Stores the retrieved data into a variable of each data type. "H000" is a variable of bool type because it is assumed to be a bit register, "HB000" is a variable of byte type because it is assumed to be a byte register, and "HW000" is stored in short type because it is assumed to be a word register.

Example (C#)

```
object[] tableData = this.m_varRegTable.value as object[];
//register data acquisition (bit type)
bool? bData = tableData[0] as bool?;
//register-data-acquisition (byte type)
byte? bytData = tableData[1] as byte?;
//register-data-acquisition (short type)
short? iData = tableData[2] as short?;
```

4.1.1.3. Disconnection

To disconnect from the controller, you erase the generated objects and delete the objects that you want to erase from the collection class that manages the objects. However, you do not need to explicitly remove it if you use ORiN.ManagedCAO. The following is a code example:

Example (C#)

```
// Remove all objects from CaoEngine
this.m_caoEngine.Dispose();
// Clear CaoEngine
this.m_caoEngine = null;
```

5. NCBOY Provider Error Codes

This provider has the following unique error codes masked with the 0x8011****. (Refer to Table 5-1 Unique Error Codes Table.)

For information about common ORiN2 errors, see the Error Codes section in ORiN2 Programming Guide.

Table 5-1 Unique Error Codes

Error Number	Description
0x80110001	AddController::Conn optional specification failure Conn option is specified incorrectly.
0x80110002	AddController::Timeout optional specification failure Timeout option is specified incorrectly.
0x80110101	AddVariable::Type optional error Type optional specification when adding a user variable is incorrect.
0x80110102	<register table> variable option error The required option for the <register table> variable is not specified or the specification method is incorrect. Check the setting range and method again.
0x80110103	<register table> variable table option error The table number that has already been used or does not exist. Delete CaoVariable that you are using or specify a different table number.
0x80110104	<Alarm history> Variable option error The required option for the <alarm history> variable is not specified or the specification method is incorrect. Check the setting range and method again.

If the received data returned from NCBOY is unexpected, this provider masks the received data with "0x801000**" and returns it. If the received data is a command error, "0x801001**" is masked and returned.

Error Number	Description
0x80100001	Checksum check of received data failed. Review the cable disconnection and communication environment.
0x80100002	Receive data packet error. An unexpected data was returned. Review the communication environment.
0x80100101	Warning information defined for each command. Applicable when the memory is being cleared.
0x801001FF	A communication error occurred.
0x801001FE	A checksum error occurred.
0x801001FD	A sampling error has occurred.
0x801001FC	A command number error occurred.

Error Number	Description
0x801001FB	The number of data bytes has been exceeded.
0x801001FA	An RS communication read error occurred.
0x801001F9	An RS communication write error occurred.
0x801001F8	An RS communication time-over occurred.
0x801001F7	A data error occurred.
0x801001F6	Setting mode error occurred.
0x801001F2	A memory error occurred.
0x801001F1	Register list undefined occurred.
0x801001F0	A register name error occurred.

6. Trouble shooting

This section provides troubleshooting information for problems that you encounter while working with your provider.

6.1. When error code "0x80110002" occurs frequently during communication

Receive data packet error when data is acquired with CaoVariable::value property

(0x80110002) If you receive frequent returns, try the following:

Description
Decrease the baud rate of NCBOY.

Appendix A. Communication protocol command correspondence table

Communication command number	Supported Variables
01	@ALARM Variable Value Properties
02	<Alarm history> Variable Value property
03	@DEVICE_VERSION Variable Value Properties
04	@SYSTEMINFO Variable Value Properties
11	<register table> When executing variable AddVariable
12	<register table> Variable Value property