芝浦機械
NCBOY プロバイダ
ユーザーズ ガイド

Version 1.0.0

April 16, 2021

備考:

【改版履歴】

バージョン	日付	内容
1.0.0	2021-04-16	初版.

【対応機種】

機種	バージョン	注意事項
NCBOY-200	-	-
NCBOY-3200	-	-

【動作確認機種】

機種	バージョン	注意事項
NCBOY-3200	VNAM=07.09 VTA=07.09 TERA=01.01	-

目次

1. はじめに

本書は，株式会社芝浦機械の NCBOY に対してレジスタテーブルにレジスタ名を登録し，対応したレジスタデータを取得するプロバイダのユーザーズガイドです．図 1-1 が本プロバイダとデバイスの全体構成図になります．以降本プロバイダを NCBOY プロバイダと呼称します．

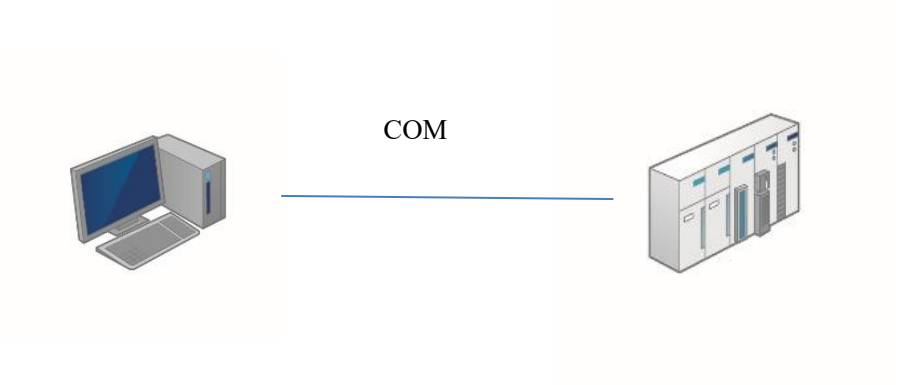

図 1-1 構成図

また，本プロバイダ及びデバイスそれぞれの対応を図 1-2 に表します．
(※一例です．全てを表しているわけではありません．)

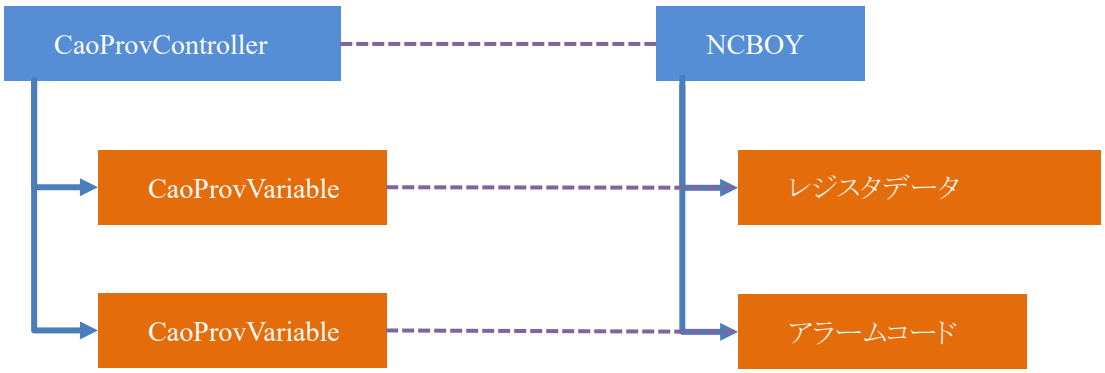

図 1-2 プロバイダの構成とデバイス情報との対応図

2. アプリケーション開発のための環境セットアップ

2.1. NCBOY とクライアント PC との接続

本プロバイダは, RS-232C 準拠の COM 接続により NCBOY 本体との通信を行います. NCBOY のポート番号 2 に通信ケーブルを接続してください.

3. コマンドリファレンス

3.1. メソッド/プロパティー覧

表 3-1 メソッド/プロパティー覧

カテゴリ	メソッド/プロパティ[1]		機能	参照
CaoWorkspace				
	AddController	M	コントローラに接続	P.8
CaoController				
	Variables	P	コントローラが保持する変数コレクションの取得	P.9
	AddVariable	M	変数オブジェクトの追加	P.10
CaoVariable				
	Value	P	値の取得/設定	P.10

3.2. メソッド・プロパティ

3.2.1. CaoWorkspace クラス

3.2.1.1. AddController メソッド

CaoWorkspace に，コントローラオブジェクトを追加します．NCBOY プロバイダでは，AddController メソッド実行時に渡されたパラメータを参照し，該当する NCBOY と接続を行います．以下に，AddController メソッドの仕様を示します．

書式

```
AddController
(
    "<コントローラ名>",                          // コントローラ名(任意)
    " CaoProv.SHIBAURAMACHINE.NCBOY ",// プロバイダ名（固定）
    "<マシン名>",                                // プロバイダ実行マシン名(未使用)
    "<オプション>"                               // オプション文字列(省略可能)
)
```

オプション

以下にオプション文字列に指定するオプションを示します．オプション文字列は下記に示す各オプションをカンマ(,)でつなげた文字列となります．

[1] M:メソッド, P:プロパティ, E:イベントをそれぞれ示します．

オプション	必須	説明	値範囲	デフォルト値
Conn	○	NCBOY との COM 接続における通信パラメータを指定します. 詳細は, 3.2.1.1.1 を参照してください.		
Timeout	-	接続先と1回の通信におけるタイムアウトまでにおける待機時間(ms)を指定します.	0-	2000

3.2.1.1.1. CONN オプション

以下に Conn オプションの接続パラメータ文字列を示します. ここで中括弧("[]")内は省略可能なことを, 各パラメータの解説中の下線部はオプションを指定しなかった時のデフォルト値をそれぞれ示します.

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

<COM Port>	:	COM ポート番号. '1' -COM1, '2' - COM2, ...
<BaudRate>	:	通信速度. 4800, 9600, 19200, 38400, 57600, 115200
<Parity>	:	パリティ. 'N'-NONE, 'E'-EVEN, 'O'-ODD
<DataBits>	:	データビット数. '7'-7bit, '8'-8bit
<StopBits>	:	ストップビット数. '1'-1bit, '2'-2bit
<Flow>	:	フロー制御. '0'-None, '1'-Xon/Xoff, '2'-ハードウェア制御 OR をとって指定できます.

使用例(C#)

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

3.2.2. CaoController クラス

3.2.2.1. Variables プロパティ

コントローラが保持する, 変数コレクションを取得します.

使用例(C#)

```
// 変数コレクション取得
ORiN2.ManagedCAO.CCaoVariables variables = controller.Variables;
```

```
// 変数取得
ORiN2.ManagedCAO.CCaoVariable variable = variables[0];
```

3.2.2.2. AddVariable メソッド

CaoController に変数オブジェクトを追加します. 変数名には 3.3.2 に示すもののみ使用できます.
以下に, AddVariable の仕様を示します.

書式

```
AddVariable
(
    "<変数名>",              // 変数名
    "<オプション>"           // オプション文字列(省略可能)
)
```

3.2.3. CaoVariable クラス

3.2.3.1. Value プロパティ

接続した NCBOY からデータを取得します. 変数名によって動作が異なります. 詳細は, 3.3.変数一覧を参照してください.

3.3. 変数一覧

各クラスで使用可能な変数一覧を定義します. なお変数は, CaoVariable クラスのオブジェクトを指します.

3.3.1. システム変数とユーザー変数

プロバイダにはシステム変数とユーザー変数の 2 種類の変数が存在します.

システム変数

その変数を保持するオブジェクト内で唯一の情報にアクセスするための変数です. システム変数はしばしば静的データである場合があります. システム変数は名前の先頭に"@"がついているのが特徴です. 詳細については, 3.3.2 を参照してください.

ユーザー変数

変数を作成する際にどのような情報にアクセスするのかを, オプション文字列を使用することでユーザーが指定できる変数です. ユーザー変数はその性質上, 複数変数を登録(オプションのみ変更したい場合等に有用)するために任意の文字列を付与することが可能です. 本プロバイダのユーザー変数は, Type オプションにより取得する内容が異なります. ユーザー変数については 3.3.3 を参照してください.

3.3.2. CaoController クラス システム変数

変数名	説明	Value	参照

		get	put	
@MAKER_NAME	メーカー名を取得します.	○	-	P.11
@VERSION	DLL バージョンを取得します.	○	-	P.11
@ALARM	現在 NOBOY で検出されているアラームコードを取得します.	○	-	P.11
@DEVICE_VERSION	NCBOY 内の各 CPU のソフトウェアバージョンを取得します.	○	-	P.12
@SISTEMINFO	各ユニットの情報およびアンプの情報を取得します.	○	-	P.13

3.3.2.1. @MAKER_NAME

メーカー名の取得をします.

データ型

型説明	
VT_BSTR	メーカー名を保持します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@MAKER_NAME","");
// 値取得
string value = var.Value as string;
```

3.3.2.2. @VERSION

DLL のバージョンの取得をします.

データ型

型説明	
VT_BSTR	DLL のバージョンを保持します. *.*.*

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@VERSION","");
// 値取得
string value = var.Value as string;
```

3.3.2.3. @ALARM

NCBOY 内で現在検出されているアラームコードの一覧を 32 個取得します.

データ型

型説明	
VT_UI2 | VT_ARRAY	
i^2	検出されたアラーム番号を保持します. ※アラーム数が 32 個に満たない場合は, 0 が格納されます. アラーム数が 2 個の場合,配列の内容が「(アラーム 1),(アラーム 2),0,0,0,0・・・・0,0」というように返却されます.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@ALARM","");
// 値取得
ushort[] alarms = var.Value as ushort[];
```

3.3.2.4. @DEVICE_VERSION

NCBOY 内の各 CPU のソフトウェアバージョンを取得します.

データ型

型説明	
VT_BSTR | VT_ARRAY	
VT_BSTR	VNAM バージョンを保持します. **.**
VT_BSTR	VTA バージョンを保持します. **.**
VT_BSTR	TARA バージョンを保持します. **.**

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@DEVICE_VERSION","");
// 値取得
string[] versions = var.Value as string[];
// 各バージョンに分解
string vnamVer = versions [0];
string vtaVer = versions[1];
```

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

```
string taraVer = versions[2];
```

3.3.2.5. @SYSTEMINFO

各ユニットの情報およびアンプの情報を取得します.

データ型

型説明				
VT_VARIANT | VT_ARRAY				
0	VT_UI1			システム判断フラグを保持します. 値: 0 = NCBOY-200, 1 = NCBOY-3200
1	VT_UI4			ユニット分配周期[μs]を保持します.
2	VT_UI2			ユニット総数を保持します.
3	VT_UI2			総軸数を保持します.
4	VT_ARRAY | VT_VARIANT			ユニット情報をユニット総数分保持します.
	n[3]	VT_ARRAY | VT_UI2		
		0	VT_UI2	各ユニットの小数点位置を保持します.
		1	VT_UI2	各ユニットの軸数を保持します.
5	VT_ARRAY | VT_VARIANT			軸情報を取得します. この配列には, すべてのユニットの軸情報がユニット番号の昇順にまとめて格納されています.
	m[4]	VT_ARRAY | VT_BSTR		
		0	VT_BSTR	軸名称を保持します.
		1	VT_BSTR	アンプバージョンを保持します. **.**

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("@SYSTEMINFO", "");
// 値取得
object[] systemInfo = var.Value as object[];
// 各項目に分解
byte? systemFlag = systemInfo[0] as byte?;
uint? unitDstributionCycle = systemInfo[1] as uint?;
```

[3] $0 \leq n \leq$ ユニット総数
[4] $0 \leq m \leq$ 総軸数

```
ushort? unitCount = systemInfo[2] as ushort?;
object[] unitInfos = systemInfo[4] as object[];
// ユニット情報の取得例
foreach (var unitInfoObject in unitInfos)
{
    UInt16[] unitInfo = unitInfoObject as UInt16[];
    UInt16 decimalPointPos = unitInfo[0];
    UInt16 axisCountPerUnit1 = unitInfo[1];
}

// ユニット別の軸名称の取得例
List<string>[] axisNames = new List<string>[unitCount.Value];
ushort? axisCount = systemInfo[3] as ushort?;     // 総軸数
int unitNo = 0;                                   // ユニット番号
int axisNoForUnit = 0;                            // ユニット内軸番号
ushort? axisCountPerUnit = 0;                     // ユニット内軸数
foreach (var axisInfoObject in (systemInfo[5] as object[]))
{
    string[] axisInfo = axisInfoObject as string[];
    // ユニット切り替え時
    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);

  // ユニット別カウントをインクリメントして必要であれば次ユニットに切り替える
  axisNoForUnit++;
  if (axisNoForUnit == axisCountPerUnit)
  {
      axisNoForUnit = 0;
      axisCountPerUnit = 0;
      unitNo++;
  }
```

```
}
```

3.3.3. CaoController クラス ユーザー変数

変数名	説明	Value		参照
		get	put	
<レジスタテーブル変数>	レジスタの値を取得します. 変数 1 つにつき, 最大 16 個のレジスタの値を取得可能です. レジスタ変数は最大 32 個作成することが可能です. (Type オプション 0 固定)	○	-	P.15
<アラーム履歴>	任意のアラーム履歴を取得します. (Type オプション 1 固定)	○	-	P.17

3.3.3.1. <レジスタテーブル>変数

本変数は Register オプションで指定したレジスタの値を一括で取得します.

プロバイダは, 本変数作成時に, 複数レジスタを一括で扱うためのテーブル(レジスタテーブルと呼ぶ)を NCBOY 上に設定し, 以後はそのテーブルを介してレジスタの値にアクセスします.

変数作成時に指定したテーブル番号が, 既存変数のテーブル番号と一致する場合, 既存変数がアクセスするレジスタテーブルの変更を防ぐためにエラーを返します.

なお, レジスタテーブルはプロバイダと NCBOY との通信に使用するだけであり, NCBOY の動作には影響しません.

オプション	必須	説明	値範囲
Type	○	0 固定	--
Table	○	使用するレジスタテーブル番号を指定します. テーブル番号は一意である必要があります.	1 ~ 32
Register	○	値を取得したいレジスタを最大 16 個まで指定します. ※指定する際は各レジスタ名の間に":"(コロン)を入れて区切ります. 例) Register=H123:HB456:HW789	--

データ型

型説明
VT_ARRAY | VT_VARIANT

<table>
<tr>
<td>i[5]</td>
<td>VT_VARIANT</td>
<td>登録したレジスタ名に対応するデータを保持します.
各レジスタのデータ型に対応する VT_VARIANT のデータ型は, 以下の対応表を参照してください.
<table>
<tr><th>レジスタデータ型</th><th>対応型</th></tr>
<tr><td>レジスタ名異常</td><td>VT_EMPTY</td></tr>
<tr><td>ビットレジスタ</td><td>VT_BOOL</td></tr>
<tr><td>バイトレジスタ(符号あり)</td><td>VT_I1</td></tr>
<tr><td>バイトレジスタ(符号なし)</td><td>VT_UI1</td></tr>
<tr><td>バイトレジスタ(16 進)</td><td>VT_UI1</td></tr>
<tr><td>ワードレジスタ(符号あり)</td><td>VT_I2</td></tr>
<tr><td>ワードレジスタ(符号なし)</td><td>VT_UI2</td></tr>
<tr><td>ワードレジスタ(16 進)</td><td>VT_UI2</td></tr>
<tr><td>ロングレジスタ(符号あり)</td><td>VT_I4</td></tr>
<tr><td>ロングレジスタ(符号なし)</td><td>VT_UI4</td></tr>
<tr><td>ロングレジスタ(16 進)</td><td>VT_UI4</td></tr>
<tr><td>タイマレジスタ</td><td>VT_UI4
1 バイト目 : ON/OFF 情報
タイムアップ, カウントアップで 1, それ以外は 0 になります.
2 バイト目 : ステータス情報
0 = 未使用
1 = カウンタプリセット
2 = カウント中
3 = カウントアップ
3, 4 バイト目 : カウントデータ
残りタイマ値または残りカウント値</td></tr>
<tr><td>浮動小数点レジスタ</td><td>VT_R4</td></tr>
</table>
</td>
</tr>
</table>

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("TABLE01","Type=0, Table=1, Register=Reg1:Reg2:Reg3");
// 値取得
object[] value = var.Value as object[];      //as object[]で object 配列にキャストする
```

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

```
byte? reg1 = value[0] as byte?;          //Reg1 は符号ありバイトレジスタ想定
short? reg2 = value[1] as short?;        //Reg2 は符号ありワードレジスタ想定
int? reg3 = value[2] as int?;            //Reg3 は符号ありロングレジスタ想定
```

3.3.3.2. <アラーム履歴>変数

指定したアラーム履歴番号から取得したいデータ数分のアラーム履歴を取得します.
取得するアラーム履歴番号が 255 を超える場合は, 先頭のアラーム履歴番号に戻りデータを取得します.
例)アラーム履歴番号が 250 から 24 個指定した場合は, 250~255,0~17 のアラーム履歴が取得されます.

オプション	必須	説明	値範囲
Type	○	1 固定	--
No	○	取得するアラーム履歴の開始位置を指定します.	0-255
Elem	○	取得するアラーム履歴の個数を指定します.	1-24

データ型

型説明			
VT_ARRAY | VT_VARIANT			
n[6]	VT_ARRAY | VT_VARIANT		アラーム履歴 1 つ分の情報を保持します. アラーム履歴が存在しない場合は VT_EMPTY になります.
	0	VT_UI2	アラームコードを保持します.
	1	VT_DATE	アラーム履歴を保持します.

使用例(C#)

```
// 変数追加
ORiN2.ManagedCAO.CCaoVariable var = controller.AddVariable("ALARMHISTORY","Type=1, No=0, Elem=24");
// 値取得
object[] alarmHIstory = var.Value as object[];
foreach (var alarmObject in alarmHIstory)
{
    object[] alarmInfo = alarmObject as object[];
    if (alarmInfo != null)
    {
        ushort? alarmCode = alarmInfo[0] as ushort?;
```

[6] $0 \leq i \leq$ Elem オプション

```
            DateTime? dateTime = alarmInfo[1] as DateTime?;
        }
    }
```

4. NCBOY プロバイダによるプログラミング

NCBOY プロバイダでは，以下の手順でクライアント PC と NCBOY を接続することができます．

- CaoEngine の作成
- CaoWorkspace の作成
- CaoController の作成

NCBOY に接続した後は，CaoVariable オブジェクトを生成することで，NCBOY の情報にアクセスすることができます．

4.1. レジスタ内のデータを取得するサンプルプログラミング

ここでは例として NCBOY 内の一部レジスタ内のデータを取得するためのサンプルプログラムについて解説します．

表 4-1 にサンプルプログラムの要件を，図 4-1 にサンプルプログラムの流れをそれぞれ記述しています．

表 4-1 サンプルプログラムの要件

要件	説明
接続先	COM で接続する
	接続先 COM ポート番号は1
処理内容	レジスタ"H000","HB000","HW000"をテーブルに設定する
	設定したテーブルのレジスタリストからデータの取得を行う

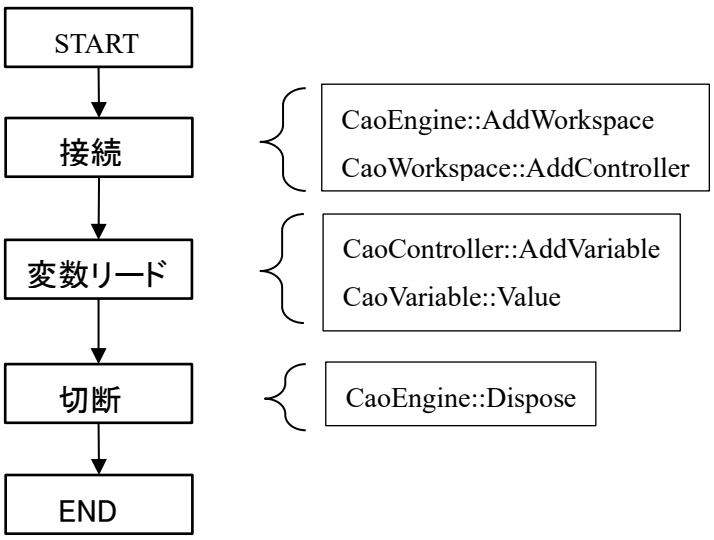

図 4-1 データ取得までの流れ

以降の節から具体的なコードを示します．

4.1.1. サンプルプログラム

以下にサンプルプログラムの全体像を示します.

Sample | **Sample.cs**

```
// オブジェクト
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()
{
    // 接続
    this.Connect();
    //レジスタテーブル変数の追加
    this.m_varRegTable = this.m_caoController.AddVariable("RegTable1","Type=0,Table=1,Register=H000:HB000:HW000");

    object[] tableData = this.m_varRegTable.Value as object[];
    //***レジスタデータ取得(bit 型)
    bool? bData = tableData[0] as bool?;
    //***レジスタデータ取得(byte 型)
    byte? bytData = tableData[1] as byte?;
    //***レジスタデータ取得(short 型)
    short? iData = tableData[2] as short?;
    // 切断
    this.Disconnect();
}

// 接続メソッド
private void Connect()
{
    // CaoEngine オブジェクトの生成
    this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
    // CaoWorkspace オブジェクトの生成
    this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
    // CaoController オブジェクトの生成
    this.m_caoController = this.m_caoWorkspace.AddController("NCBOYSample",
```

```
                                        "CaoProv.SHIBAURAMACHINE.NCBOY",
                                        "",
                                        "Conn=COM:1");
    }

    // 切断メソッド
    private void Disconnect()
    {
        this.m_caoEngine.Dispose();
        this.m_caoEngine = null;
    }
```

4.1.1.1. 接続

以下の手順で NCBOY と接続します.

(1) オブジェクトを保持するための変数を用意します.

接続に必要なオブジェクトは, CaoEngine オブジェクトと CaoWorkspace オブジェクトと CaoController オブジェクトです. CaoWorkpace オブジェクトは, CaoController オブジェクトを CaoWorkspaces から取得する場合には変数を用意する必要はありません. また変数にアクセスするための CaoVariable オブジェクトも必要になります. 以下に C#でのコード例を示します.

使用例(C#)

```
    // CaoEngine オブジェクト用の変数
    private ORiN2.ManagedCAO.CCaoEngine m_caoEngine = null;
    // CaoWorkspace オブジェクト用の変数
    private ORiN2.ManagedCAO.CCaoWorkspace m_caoWorkspace = null;
    // CaoController オブジェクト用の変数
    private ORiN2.ManagedCAO.CCaoController m_caoController = null;
    // CaoVariable オブジェクト用の変数
    private ORiN2.ManagedCAO.CCaoVariable m_varRegTable = null;
```

(2) CaoEngine オブジェクトを生成します.

CaoEngine オブジェクトは New キーワードを使って生成します.

使用例(C#)

```
    // CaoEngine オブジェクトの生成
    this.m_caoEngine = new ORiN2.ManagedCAO.CCaoEngine();
```

(3) CaoWorkspace オブジェクトを取得もしくは生成します.

CaoEngine オブジェクトを生成すると, デフォルトで CaoWorkspaces オブジェクトと CaoWorkspace オブジェクトを 1 つずつ生成します. 以下に CaoWorkspace オブジェクトを新しく生成するコード例とデフォルトの CaoWorkspace を示します.

使用例(C#)

```
// CaoWorkspace オブジェクトの生成
this.m_caoWorkspace = this.m_caoEngine.AddWorkspace("NewWrks", "");
```

(4) CaoController オブジェクトを生成します.

CaoController オブジェクトを生成するには, 使用するプロバイダ名と使用するためのパラメータを設定します. NCBOY プロバイダでは, 接続先を Conn オプションで指定します. 以下にコード例を示します.

使用例(C#)

```
// CaoController オブジェクトの生成
this.m_caoController = this.m_caoWorkspace.AddController("NCBOYSample",
                                                          "CaoProv.SHIBAURAMACHINE.NCBOY",
                                                          "",
                                                          "Conn=COM:1");
```

4.1.1.2. レジスタデータ取得

(1) レジスタテーブルにアクセスする CaoVariable オブジェクトを生成します.

レジスタテーブル 1 に対してレジスタ名「H000」,「HB000」,「HW000」を指定して<レジスタテーブル>変数を作成します.

※レジスタ名は NCBOY に実際に設定されているレジスタ名を指定してください.

使用例(C#)

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

(2) 追加した<レジスタテーブル>変数からデータの取得を行います.

取得する際は, CaoVariable オブジェクトの Value プロパティで取得します. 取得したデータを各データ型の変数へと格納します. "H000"はビットレジスタ想定のため bool 型の変数, "HB000"はバイトレジスタ想定のため byte 型の変数, "HW000"はワードレジスタ想定のため short 型にそれぞれ格納します.

使用例(C#)

```
object[] tableData = this.m_varRegTable.value as object[];
//レジスタデータ取得(bit 型)
bool? bData = tableData[0] as bool?;
//レジスタデータ取得(byte 型)
```

```
byte? bytData = tableData[1] as byte?;
//レジスタデータ取得(short 型)
short? iData = tableData[2] as short?;
```

4.1.1.3. 切断

コントローラと切断する場合は，生成したオブジェクトを消去すると共に，オブジェクトを管理するコレクションクラスから消去するオブジェクトを削除します．ただし，ORiN.ManagedCAO を使用した場合は明示的に削除する必要はありません．以下にコード例を示します．

使用例(C#)

```
// CaoEngine からすべてのオブジェクトを削除
this.m_caoEngine.Dispose();
// CaoEngine の消去
this.m_caoEngine = null;
```

5. NCBOY プロバイダエラーコード

本プロバイダには, 0x8011****でマスクした以下の独自エラーコードが存在します. (表 5-1 独自エラーコード表参照)

ORiN2 の共通エラーについては,「ORiN2 プログラミングガイド」のエラーコードの章を参照してください.

表 5-1 独自エラーコード表

エラー番号	説明
0x80110001	AddController::Conn オプション指定エラー Conn オプションの指定方法が間違っています.
0x80110002	AddController::Timeout オプション指定エラー Timeout オプションの指定方法が間違っています.
0x80110101	AddVariable::Type オプションエラー ユーザー変数追加時の Type オプション指定が間違っています.
0x80110102	<レジスタテーブル>変数オプションエラー <レジスタテーブル>変数の必須オプションが指定されていないか, 指定方法が間違っています. 設定範囲や設定方法を再度ご確認ください.
0x80110103	<レジスタテーブル>変数テーブルオプションエラー すでに使われているテーブル番号か, 存在しないテーブル番号です. 使用している CaoVariable 変数を削除するか別のテーブル番号を指定してください.
0x80110104	<アラーム履歴>変数オプションエラー <アラーム履歴>変数の必須オプションが指定されていないか, 指定方法が間違っています. 設定範囲や設定方法を再度ご確認ください.

また, 本プロバイダは, NCBOY から返却される受信データが想定外だった場合「0x801000**」でマスクして返します. 受信データがコマンドエラーだった場合,「0x801001**」でマスクして返します.

エラー番号	説明
0x80100001	受信データのチェックサム確認が失敗しました. ケーブルの断線や通信環境を見直してください.
0x80100002	受信データパケット異常エラーです. 想定外のデータが返却されました. 通信環境を見直してください.
0x80100101	各コマンドで定義されている警告情報です. メモリクリア中などが該当します.
0x801001FF	通信異常が発生しました.
0x801001FE	チェックサム異常が発生しました.
0x801001FD	サンプリング異常が発生しました.

エラー番号	説明
0x801001FC	コマンド番号異常が発生しました.
0x801001FB	データバイト数オーバーが発生しました.
0x801001FA	RS 通信リードエラーが発生しました.
0x801001F9	RS 通信ライトエラーが発生しました.
0x801001F8	RS 通信タイムオーバーが発生しました.
0x801001F7	データ異常が発生しました.
0x801001F6	設定モード異常が発生しました.
0x801001F2	メモリ異常が発生しました.
0x801001F1	レジスタリスト未定義が発生しました.
0x801001F0	レジスタ名エラーが発生しました.

6. トラブルシューティング

本項では，プロバイダを運用中に発生した問題へのトラブルシューティングを記載します．

6.1. 通信中にエラーコード「0x80110002」が頻繁に発生する場合

CaoVariable::value プロパティでデータの取得をした際に「受信データパケット異常エラー（0x80110002）」が頻繁に返却される場合，以下のことを試してください．

内容
NCBOY 本体のボーレートを下げてください．

付録A. 通信プロトコルコマンド対応表

通信コマンド番号	対応変数
01	@ALARM 変数 Value プロパティ
02	<アラーム履歴>変数 Value プロパティ
03	@DEVICE_VERSION 変数 Value プロパティ
04	@SYSTEMINFO 変数 Value プロパティ
11	<レジスタテーブル>変数 AddVariable 実行時
12	<レジスタテーブル>変数 Value プロパティ