三菱電機シーケンサ MELSEC Q, iQ-F 等対応 Ethernet, RS-232C 通信 Visual Basic .NET DLL

SimpleMC 1.0

取り扱い説明書

Mail: 下記サイトのメールフォームから
URL: https://www.kumasys.jp

更新履歴

2023.05.02. SimpleMC ver1.00 公開 （ただし元となったソフトがあり、そのリネーム版）

目次

1. 開発環境

本ライブラリは Visual Basic .NET、 .NET Framework 4.5 以上の SDK におけるユーザアプリの作成を想定しています。C# で利用される場合の機能保証は致しません。

2. 対応通信プロトコル

2.1. Ethernet プロトコル

	文字コード	
ﾌﾚｰﾑ	バイナリ	ASCII
3E	○	○
4E	○	○

2.2. RS-232C プロトコルとデータビット

	ﾌﾚｰﾑ				文字ｺｰﾄﾞ		
形式	1C	2C	3C	4C	ﾊﾞｲﾅﾘ	ASCII	ﾋﾞｯﾄ
形式 2	○	○	○	○	-	○	7bit
形式 3	○	○	○	○	-	○	7bit
形式 4	○	○	○	○	-	○	7bit
形式 5	-	-	-	○	○	-	8bit

2.3. RS-232C 伝送速度(bps)

300/600/1200/2400/4800/9600/14400/19200/38400/57600/115200
の範囲で、RS-232C の端末にあわせて設定可能

2.4. RS-232C パリティ

奇数、偶数を RS-232C 端末に合わせて設定可能

3. 対応 PLC

MC プロトコルに対応した PLC 通信ユニット

三菱電機
MELSEC Q シリーズ
＜Ethernet＞ QJ71E71-100 や Ethernet 機能付き PLC **＜RS-232C>** QJ71C24N-R2 や RS-232C 機能(計算機リンク)付き PLC
MELSEC QnA シリーズ
（動作未保証）
MELSEC iQ-F シリーズ
<Ethernet> FX5U の内蔵 Ethernet 通信機能
MELSEC iQ-R シリーズ
（動作未保証）

4. 開発環境

項目	詳細
メモリ、ディスク	16MB 以上の空メモリ容量 16MB 以上の空きディスク容量
OS	Windows7 64bit
フレームワーク	.NET Framework 4.5 SDK
言語	Visual Basic .NET （C# 未保証）
統合開発環境	Visual Studio 2013, 2019

統合開発環境、フレームワークの動作条件や利用方法につきましては Microsoft のホームページ等をご参照ください。

5. インテリセンス xml

本ライブラリはインテリセンス(コーディング補助情報)も xml ファイルとして付属しています。Visual Studio を使って作成される場合、この xml ファイルをプロジェクトから読込むことでプログラムコードを入力しているさいに DLL のクラスや関数などの使い方が表示されます。これによりユーザがアプリケーションを作成する時間が短縮します。

ダウンロードファイルに付属しているサンプルプロジェクトには DLL やインテリセンス（XML）をすでにインポート済みですので設定等のご参考に利用ください。

6. 端末(PC)の設定

一般の Ethernet や RS-232C 通信と同じ要領で設定してください。
ただし、RS-232C の場合、推奨される設定がありますので付録をご確認ください。

7. アセンブリ

ユーザプロジェクトで最初に以下のコードによりアセンブリを読み込むように設定します。

```
Imports SimpleMC.SimpleMC
```

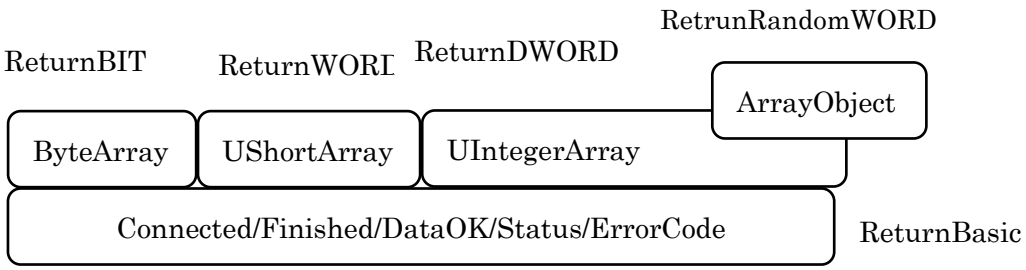
アセンブリを利用するには以下のようにインスタンスを作成します。

```
Dim ins As SimpleMC.SimpleMC = New SimpleMC.SimpleMC
```

8. クラス（ステータス、データ）

通信の戻り値となるクラスは、必ず ステータス(ReturnBasic)クラスを含んでいます。
Write(書き込み)系の関数であれば ReturnBasic クラスが戻り値になります。
関数により戻り値クラスが異なりますので各関数の項目をご参照ください。

戻り値クラスの関係は以下になります。



戻り値の使い方に関しましては、

```
ins.WriteDeviceRandom2(3, objword).Finished
```

このように直接クラス内のデータを取得することもできますが、

```
Dim rtn As New RetrunRandomWORD
```

いったんこのようにデータ型を定義してから、

```
rtn = ins.WriteDeviceRandom2(3, objword)
```

このように戻り値クラスを退避してから戻り値をお使いください。

8.1. ステータスクラス ReturnBasic

この戻り値クラスは非同期通信におけるステータスを含みます。

ステータス	データ型	意味
Connected	Boolean	接続成功
Finished	Boolean	通信終了または中断
DataOK	Boolean	通信成功
Status	Byte	ステータス（使用非推奨）
ErrorCode	UShort	（ダミー）

これらのうち**実際に利用するのは Finished と DataOK になります。**

＜同期実行をおこなう＞

本ライブラリにおける通信は非同期通信になります。そのため同期実行を行う場合は Finished ステータスを用います。

```
While Not rtn.Finished : End While
```

このように Finished ステータスの値をみる While 文で待機することで同期実行になります。While を抜けるタイミングは、通信成功、またはタイムアウト時間が経過したときです。これにより待機させる場合は通信関数実行前に必ず適切なタイムアウト時間を設定しておいてください。

＜通信結果を利用する＞

通信成功の可否を次の動作に利用したい場合は、 DataOK ステータスを用います。

```
While Not rtn.Finished : End While
        If rtn.DataOK Then
'   通信成功時の処理
        Else
'   通信失敗時の処理
        End If
```

8.2. データクラス (ReturnBasic を包含)

Read(読み込み)系関数の戻り値である各データクラスは必ずステータスクラス ReturnBasic を含みます(継承関係は「クラス(ステータス、データ)」参照)。

```
Dim rtn As New ReturnBIT
rtn = ins.ReadDeviceBlock2ArrayBit("M0", 10)
While Not rtn.Finished : End While
        If rtn.DataOK Then ' 通信成功時
MsgBox(rtn.ByteArray(1))   '  通信で取得したリレーM2 の値をボックスに表示
        Else
'  通信失敗時の処理
        End If
```

このように**関数に対応した戻り値クラス**のインスタンス変数を定義し、戻りインスタンスをこのようにいったん退避すれば、定義した変数を用いてステータスもデータも利用できるというわけです。

8.3. データクラス ReturnWORD

このクラスは ReadDeviceBlock2DWORD などワードデータ読み込みの関数の戻り値クラスであり、ReturnBasic クラスを包含し、PLC から取得した UShort 配列データ UShortArray を含みます。

※関数実行後にこのクラスの UShortArray を用いる場合は通信が成功しているかを条件で見てから用いてください

8.4. データクラス ReturnDWORD

このクラスは ReadDeviceBlock2 などダブルワードデータ読み込みの関数の戻り値クラスであり、ReturnBasic クラスを包含し、PLC から取得した UInteger 配列データ UIntegerArray を含みます。

※関数実行後にこのクラスの UIntegerArray を用いる場合は通信が成功しているかを条件で見てから用いてください

8.5. データクラス ReturnBIT

このクラスは ReadDeviceBlock2ArrayBIT などビットデータ読み込みの関数の戻り値クラスであり、ReturnBasic クラスを包含し、PLC から取得した Byte 配列データ ByteArray を含みます。

※関数実行後にこのクラスの ByteArray を用いる場合は通信が成功しているかを条件で見てから用いてください

8.6. データクラス RetrunRandomWORD※3

このクラスは ReadDeviceBlock2ArrayBIT などビットデータ読み込みの関数の戻り値クラスであり、ReturnBasic クラスを包含し、PLC から取得した Ushort 配列データ UshortArray および 可変引数のオブジェクト ArrayObject ※2 を返します。

※ 関数実行後にこのクラスの ByteArray や ArrayObject を用いる場合は通信が成功しているかを条件で見てから用いてください

※2. ReadDeviceRandom2 関数は引数にデバイス名(“D0”など)を格納した配列を指定することもできますが、
直接、可変引数として “D0” などのデバイス名を指定することもできます。
戻り値の一つである ArrayObject は、可変引数としてデバイスを指定した場合に、ArrayObject と Ushort 配列が、
デバイスとデータという対応関係にあるということで用途によっては使い道があるかもしれません。

※3. 「Retrun」は本マニュアルの誤植ではありません。

9. 通信関数

```
Dim ins As New SimpleMC.SimpleMC
'Dim dat(10) As UShort
'dat にデータ格納処理（省略）
Dim ins As New ReturnBasic
ins.WriteDeviceBlock2("D0", dat, 3)
```

通信関数はこのようにアセンブリのインスタンス変数を定義してから実行します。通信関数の戻り値の使い方に関しては「7.クラス（ステータス、データ）」の各節を参照ください。

9.1. WriteDeviceBlock2(ワード単位一括書込)

この関数では任意点数の番号が連続したデバイスにワード単位の書き込みを行います。

＜第 1 引数：str As String＞
たとえば, "D700" や “M10” といった文字列で始まりのデバイスを指定してください。

＜第 2 引数：inputData As UShort() ＞
書き込みたいワードデータ

＜第 3 引数：k As UShort＞
書込ワード点数,

＜戻り値＞
戻り値は ReturnBasic クラスインスタンス

＜例＞
例： デバイス D0～D2 の 3 点に UShort 配列要素 dat(0)～dat(2)を書き込む

```
ins.WriteDeviceBlock2(“D0”, dat, 3)
```

この例では実行後（通信成功時）、
D0= dat(0)、 D1= dat(1)、 D2= dat(2)
となります。

9.2. WriteDeviceBlock2BIT(ビット単位一括書込)

この関数ではビットデバイスのみを対象とします。ビット単位にビットデバイスに書き込みます。

＜第 1 引数：str As String＞
たとえば, “M10” といった文字列で始まりのビットデバイスを指定してください。

＜第 2 引数：inputData As UShort() ＞
ビットデバイス列に書き込みたいワードデータ。つまり HEX でビット書き込みをします。

＜第 3 引数：k As UShort＞
inputData のデータのうち、実際に書き込むビット数

＜戻り値＞
戻り値は ReturnBasic クラスインスタンス

例： デバイス M0～M16 の 17 点に UShort 配列要素 dat の番号 0 の LSB から順に 17 ビット分書き込む。

```
dat(0)= &HABCD '1010_1011_1100_1101
dat(1)= &HABCD '1010_1011_1100_1101
ins.WriteDeviceBlock2BIT("M0", dat, 17)
```

この例では実行後（通信成功時）の PLC のデバイスは、
M0= dat(0) And (&B0000_0000_0000_0001)　'1
M1= dat(0) And (&B0000_0000_0000_0010)　'0
M2= dat(0) And (&B0000_0000_0000_0100)　'1
M3= dat(0) And (&B0000_0000_0000_1000)　'1
M4= dat(0) And (&B0000_0000_0001_0000)　'0
M5= dat(0) And (&B0000_0000_0010_0000)　'0
M6= dat(0) And (&B0000_0000_0100_0000)　'1
M7= dat(0) And (&B0000_0000_1000_0000)　'1
M8= dat(0) And (&B0000_0001_0000_0000)　'1
M9= dat(0) And (&B0000_0010_0000_0000)　'1
M10= dat(0) And (&B0000_0100_0000_0000)　'0

```
M11= dat(0) And (&B0000_1000_0000_0000)     '1
M12= dat(0) And (&B0001_0000_0000_0000)     '0
M13= dat(0) And (&B0010_0000_0000_0000)     '1
M14= dat(0) And (&B0100_0000_0000_0000)     '0
M15= dat(0) And (&B1000_0000_0000_0000)     '1
M16= dat(1) And (&B0000_0000_0000_0001)     '1
M17 以降 = そのまま(書き込みせず)
```

の値になっております。指定されていないビットデバイス(例では M17 以降)には**一瞬たりとも書き込みません。**
なので立てられては困るフラグがある場合に有用な関数です。

9.3. WriteDeviceBlock2ArrayBit (ビット単位一括書込)

この関数は任意点数の番号が連続したビットデバイスへ書き込む関数です。WriteDeviceBlock2 でも任意点数のビットデバイスに書き込むことができますが、書き込みデータの要素とデバイスを 1 対 1 の対応にしたい場合は 書き込みデータが Byte 配列であるこの関数を用います。

＜第 1 引数：str As String＞
たとえば, “M10” といった文字列で始まりのビットデバイスを指定してください。

＜第 2 引数：inputData As Byte() ＞
書き込む Byte 配列の各要素とビットデバイスが 1 対 1 の関係になります。
書き込む Byte 配列は 0 であれば OFF、 それ以外は ON のデータと解釈します。

＜第 3 引数：k As UShort＞
書き込むビット点数を指定します。

＜戻り値＞
戻り値は ReturnBasic クラスインスタンス

＜例＞
例： デバイス M0～M1 の 2 点に Byte 配列要素 dat(0)～dat(1)を書き込む

```
dat(0)= &00 '0 は OFF と解釈
dat(1)= &FF '0 以外は ON と解釈
ins.WriteDeviceBlock2ArrayBit(“D0”, dat, 1)
```

この例の場合、関数実行後（通信成功時)、PLC デバイスは

```
M0 = dat(0) And (&B0000_0000)   ' 0
M1=dat(1) And (&B0000_0001)     ' 1
```

になります。

9.4. WriteDeviceBlock2DWord (ダブルワード単位一括書込)

この関数では、任意点数の番号が連続したダブルワードデバイスに、UInteger 型(ダブルワード)配列データを書き込みます。

＜第 1 引数：str As String＞
たとえば, “D0” といった文字列で始まりのワードデバイスを指定してください。

＜第 2 引数：inputData As UInteger() ＞
配列番号 0 から順にダブルワード単位で各要素にデータを格納してください。
各要素とダブルワードデバイス(“D0” & “D1”、”D2” & “D3”など)は対応関係です。

＜第 3 引数：k As UShort＞
書込むダブルワードデバイスの点数を指定してください。

＜戻値＞
戻り値は ReturnBasic クラスインスタンス

＜例＞
例： デバイス D0～D5 のダブルワード 3 点に UInteger 配列 dat(0)～dat(2)を書き込む。

```
dat(0)= &H12345678
dat(1)= &H87654321
dat(2)= &HABCD6543
ins.WriteDeviceBlock2DWord("D0", dat, 3)
```

この例の場合、関数実行後（通信成功時)、PLC デバイスは
D0=0x5678, D1=0x1234
D2=0x4321, D3=0x8765
D4=0x6543, D5=0xABCD

になります。

9.5. WriteDeviceRandom2(ワード単位ランダム書込)

この関数では、任意点数の番号が連続しないワードデバイスに、UShort 型(ワード)データを書き込みます。

この関数ではデバイス、書き込みデータの指定を、2 次元オブジェクト変数を引数する方法と、可変引数で指定する方法があります（オーバーライド関数になっています)。

＜第 1 引数：k As UShort＞
書込むワードデバイスの点数を指定してください。

＜第 2 引数：inp(,) As Object＞
書き込み先のワードデバイスと対応するデータを 2 次元オブジェクトとして指定します。
オブジェクト内は デバイス名(String 型) とそれに対応する データ(UShort 型)の組み合わせにしてください。

例： D0=33, D10=55, D20=77 をオブジェクト変数という形で書き込む

```
Dim obj(2, 1) As Object
 obj(0, 0) = "D0" : obj(0, 1) = 33
 obj(1, 0) = "D0" : obj(1, 1) = 55
 obj(2, 0) = "D0" : obj(2, 1) = 77
       ins.WriteDeviceRandom2(3, obj)
```

＜第 2 引数(オーバーライド関数)：ParamArray inp() As Object＞

例： D0=33, D10=55, D20=77 を可変引数という形で書き込む

```
       ins.WriteDeviceRandom2(3, “D0”, 33, “D10”, 55, “D20”, 77)
```

可変引数で指定する場合は第 2 引数以降は、デバイス名(String)とそれに対応するデータ(Ushort 型) をセットで順番に記述します。

＜戻値＞
戻り値は ReturnBasic クラスインスタンス

9.6. WriteDeviceRandom2DWord(ダブルワード単位ランダム書込)

この関数では、任意点数の番号が連続しないダブルワードデバイスに、UInteger 型(ダブルワード)データを書き込みます。

この関数ではデバイス、書き込みデータの指定を、2 次元オブジェクト変数を引数する方法と、可変引数で指定する方法があります（オーバーライド関数になっています)。

＜第 1 引数：k As UShort＞
書込むダブルワードデバイスの点数を指定してください。

＜第 2 引数：inp(,) As Object＞
書き込み先のダブルワードデバイスと対応するデータを 2 次元オブジェクトとして指定します。オブジェクト内は ダブルワードの最初のデバイス名(String 型) とそれに対応するデータ(UInteger 型)の組み合わせにしてください。

例： D0-D1=0x12345678, D10-D11=87654321 をオブジェクト変数という形で書き込む

```
Dim obj(2, 1) As Object
 obj(0, 0) = "D0" : obj(0, 1) = &H12345678
 obj(1, 0) = "D10" : obj(1, 1) = &H87654321
       ins.WriteDeviceRandom2DWord(2, obj)
```

＜第 2 引数(オーバーライド関数)：ParamArray inp() As Object＞

例： D0-D1=0x12345678, D10-D11=87654321 を可変引数という形で書き込む

```
    ins.WriteDeviceRandom2DWord(2, “D0”, &H12345678, “D10”, &H87654321)
```

可変引数で指定する場合は第 2 引数以降は、デバイス名(String)とそれに対応するデータ(UInteger 型) をセットで順番に記述します。

＜戻値＞
戻り値は ReturnBasic クラスインスタンス

9.7. WriteDeviceRandom2ArrayBIT(ビット単位ランダム書込)

この関数では、任意点数の番号が連続しないビットデバイスに、Byte 型(ダブルワード)データを書き込みます。

この関数ではデバイス、書き込みデータの指定を、2 次元オブジェクト変数を引数する方法と、可変引数で指定する方法があります（オーバーライド関数になっています)。

＜第 1 引数：k As UShort＞
書込むビットデバイスの点数(ビット点数)指定してください。

＜第 2 引数：inp(,) As Object＞
書き込み先のビットデバイスと対応するデータを 2 次元オブジェクトとして指定します。オブジェクト内は ビットデバイス名(String 型) とそれに対応する データ(Byte 型)の組み合わせにしてください。

例： M0=0, M1=1 をオブジェクト変数という形で書き込む

```
Dim obj(2, 1) As Object
 obj(0, 0) = "M0" : obj(0, 1) = &H0
 obj(1, 0) = "M10" : obj(1, 1) = &H1
       ins.WriteDeviceRandom2(2, obj)
```

＜第 2 引数(オーバーライド関数)：ParamArray inp() As Object＞

例： M0=0, M1=1 を可変引数という形で書き込む

```
 ins.WriteDeviceRandom2ArrayBIT(2, “M0”, &H0, “M10”, &H1)
```

可変引数で指定する場合は第 2 引数以降は、デバイス名(String)とそれに対応するデータ(Byte 型) をセットで順番に記述します。

＜戻値＞
戻り値は ReturnBasic クラスインスタンス

9.8. WriteRandomBlock2(ワード単位ランダムブロック書込)

削除

9.9. ReadDeviceBlock2(ワード単位一括読込)

この関数では任意点数の番号が連続するデバイスをワード単位に読込を行います。ビット単位で UShort 配列に読み込みたい場合は ReadDeviceBlock2BIT を、ビット単位に Byte 配列に読み込みたい場合は ReadDeviceBlock2ArrayBIT を用いてください。

＜第 1 引数：str As String＞
たとえば, "D700" や “M10” といった文字列で始まりのデバイスを指定してください。

＜第 2 引数：k As UShort＞
ワードデバイスを指定した場合は読込ワード点数、ビットデバイスを指定した場合は、16×k が読込ビット数になります。

＜戻り値＞
戻り値は ReturnWORD クラスインスタンス

＜例＞

例： デバイス D0～D2 の 3 点の値を UShort 配列インスタンスに読込み
D2 の値を表示

```
Dim ins As New ReturnWORD
ins=ReadDeviceBlock2(“D0”, 3)
Msgbox(ins.UShortArray(2))
```

例： デバイス **M0-M15 の 1 ワード(16 点)**の値を UShort 配列インスタンスに読込み、M0-M15 の値(16 点)を 2 進数表示。

```
Dim ins As New ReturnWORD
ins=ReadDeviceBlock2(“M0”, 1)
Msgbox(Convert.ToString(ins.UShortArray(0),2))
```

デバイス M0-M16 の 16 点が **M15 より順に** 1,0,0,0,0,1,0,0,1,1,0,0,0,0,1,0 だった場合、これを実行した場合、
「1000010011000001」と表示される。

9.10. ReadDeviceBlock2DWord(ダブルワード単位一括読込)

この関数では任意点数の番号が連続するダブルワードデバイスをワード単位に読込を行います。

＜第 1 引数：str As String＞
たとえば, "D700"といった文字列で始まりのデバイスを指定してください。

＜第 2 引数：k As UShort＞
読込むダブルワードの点数を指定してください。

＜戻り値＞
戻り値は ReturnDWORD クラスインスタンス

＜例＞
例： デバイス D0～D5 の 3 点のダブルワード値を UInteger 配列インスタンスに読込み、D2-D4(1 点のダブルワード)の値を表示

```
Dim rtn As New ReturnDWORD
rtn = ReadDeviceBlock2DWORD(“D0”, 3)
While Not rtn.Finished : End While
        If rtn.DataOK Then ' 通信成功時
MsgBox(rtn. UIntegerArray(1))
        Else
'  通信失敗時の処理
        End If
```

PLC デバイス の値が
D0=0x1234
D1=0x5678
D2=0xABCD
D3=0x12EF

としたとき、これを実行しボックスに表示される値は 「317696973」(0x1234ABCD)。

9.11. ReadDeviceBlock2BIT(ビット単位一括読込)

この関数では任意点数の番号が連続するビットデバイスをビット単位に UShort 型に読込を行います。

＜第 1 引数：str As String＞
たとえば, “M10” といった文字列で始まりのデバイスを指定してください。

＜第 2 引数：k As UShort＞
読み込むビットデバイスのビット数を指定してください。

＜戻り値＞
戻り値は ReturnWORD クラスインスタンス

＜例＞

例： デバイス **M0-M16 の 17 点**の値を UShort 配列インスタンスに読込み、M0-M15 の値(16 点)を 2 進数表示したあと、M16 の値を表示

```
Dim ins As New ReturnWORD
ins=ReadDeviceBlock2BIT(“M0”, 17)
Msgbox(Convert.ToString(ins.UShortArray(0),2))
Msgbox(Convert.ToString(ins.UShortArray(1),2))
```

デバイス M0-M16 の 17 点が **M16 より順に** 1,1,0,0,0,1,0,0,1,1,0,0,0,0,1,0,1 だった場合、これを実行した場合、
「1000100110000101」と表示された(LSB が M0 の値)あと
「0000000000000001」と表示される(LSB が M16 の値)。

9.12. ReadDeviceBlock2ArrayBIT(ビット単位一括読込)

この関数では任意点数の番号が連続するビットデバイスをビット単位に Byte 型に読込を行います。

＜第 1 引数：str As String＞
たとえば, “M10” といった文字列で始まりのデバイスを指定してください。

＜第 2 引数：k As UShort＞
読み込むビットデバイスのビット数を指定してください。

＜戻り値＞
戻り値は ReturnBIT クラスインスタンス

＜例＞

例： デバイス M0-M2 の 3 点の値を Byte 配列インスタンスに読込み、M2 の値を表示する。

```
Dim ins As New ReturnBIT
ins=ReadDeviceBlock2ArrayBIT(“M0”, 3)
Msgbox(ins.ByteArray(2))
```

9.13. ReadDeviceRandom2(ワード単位ランダム読込)

この関数では任意点数の番号が連続しないワードデバイスやビットデバイスを、ワード単位に UShort 配列インスタンスに読み込みます。

引数は読み込みたい デバイス名(String 型)からなる 1 次元オブジェクト配列にするか、またはデバイス名(String 型)を可変引数で指定します。

1 次元オブジェクト配列で指定する場合はオブジェクトの大きさ(要素数)がそのまま読込み点数になります。

＜戻り値＞
戻り値は ReturnWORD クラスインスタンス

＜例＞
例： D0, D100 を読込み、D100 の値を表示(オブジェクト配列で指定)

```
Dim obj(1) As Object
 obj(0) = "D0" : obj(1) = "D100"
 Msgbox(ins.ReadDeviceRandom2(obj).UShortArray(1))
```

例： D0, D100 を読込み、D100 の値を表示（可変引数で指定）

```
Msgbox(ins.ReadDeviceRandom2(“D0”,”D100”).UShortArray(1))
```

例： M0-M15, M25-M40 を読込み、M25-M40 の値を表示(オブジェクト配列で指定)

```
Dim obj(1) As Object
 obj(0) = "M0" : obj(1) = "M25"
 Msgbox(ins.ReadDeviceRandom2(obj).UShortArray(1))
```

例： M0-M15, M25-M40 を読込み、M25-M40 の値を表示(可変引数で配列で指定)

```
Msgbox(ins.ReadDeviceRandom2(“M0”,”M25”).UShortArray(1))
```

10. 符号付データを変換する関数(おまけ)

データ変換関数は DatagridView のセルと PLC デバイス間のデータのやりとりのさいに通信関数とあわせて用います。関数名が日本語です。おまけ程度の関数とお考えください。

10.1. 符号無 16 ビットデータへ変換

DatagridView のセル値(符号付 16bit 整数)を PLC デバイスに書込むときに使います。

＜引数：input As Integer＞DatagridView のセル値を指定してください。

＜戻値：UShort 型＞
WriteDeviceBlock2 の第 2 引数 UShort 型配列の要素に格納することを想定した符号なし 16bit 整数

＜例＞

```
Dim dat(1) As UShort:
 dat(0) = 符号無 16 ビットデータへ変換(DatagridView1(1,1).Value)
 ins.WriteDeviceBlock2DWord("D0", dat, 1)　' D0=(DatagridView1(1,1)の数値
```

10.2. 符号無 32 ビットデータへ変換

DatagridView のセル値(ダブルワード分の符号付 32bit 整数)を PLC デバイスに書込むときに使います。

＜引数：input As Long＞DatagridView のセル値を指定してください。

＜戻値：UInteger 型＞
WriteDeviceBlock2DWord の第 2 引数 UInteger 型配列の要素に格納することを想定した符号なし 32bit 整数

＜例＞

```
Dim ins As New SimpleMC.SimpleMC
Dim dat(1) As UInteger:
 dat(0) = 符号無 32 ビットデータへ変換 (DatagridView1(1,1).Value)
ins.WriteDeviceBlock2DWord("D0", dat, 1)　' D0, D1=(DatagridView1(1,1)の数値
```

10.3. 符号無 16 ビットデータから変換

PLC から取得したデータ値を DatagridView のセルに書込むときに使います。

＜引数：input As UShort＞
ReadDeviceBlock2 の戻値にある UShort 型配列の要素から DatagridView のセルに格納することを想定した符号付 16bit 整数

＜戻値：Short 型＞ 戻値は DatagridView セルに格納してください。

＜例＞
```
Dim ins As New SimpleMC.SimpleMC
Dim dat(1) As Short
dat= 符号無 16 ビットデータから変換(ins.ReadDeviceBlock2("D0", dat, 1))
DatagridView1(1,1).Value = dat(0)
```

10.4. 符号無 32 ビットデータから変換

PLC から取得したデータ値を DatagridView のセルに書込むときに使います。

＜引数：input As UShort＞
ReadDeviceBlock2DWord の戻値にある UInteger 型配列の要素から DatagridView のセルに格納することを想定した符号付 32bit 整数

＜戻値：Integer 型＞ 戻値は DatagridView セルに格納してください。

＜例＞
```
Dim ins As New SimpleMC.SimpleMC
Dim dat(1) As Integer
dat= 符号無 32 ビットデータから変換(ins.ReadDeviceBlock2DWord("D0", dat, 1))
DatagridView1(1,1).Value = dat(0)
```

11. 設定プロパティ

設定プロパティには「基本プロパティ」「Ethernet プロパティ」「RS-232C ネットワークプロパティ」「RS-232C 通信プロパティ」「RS-232C 伝送プロパティ」「PLC ネットワーク構成プロパティ」「オプションプロパティ」があります。

また、プロパティの一部には便宜のため列挙体を用意しておりますので、列挙体があるプロパティに関しては値にはなるべくこれをお使いください。

【必ず設定するプロパティ】
「基本プロパティ」

【PC と通信ユニット間が Ethernet 通信の場合】
「Ethernet プロパティ」を必ず設定

【PC と通信ユニット間が Ethernet 通信の場合）
「RS-232C ネットワークプロパティ」「RS-232C 通信プロパティ」を必ず設定

【PLC ネットワークにおける通信ユニットと通信する場合】
「PLC ネットワーク構成プロパティ」を必ず設定

＜プログラムを作成するときの流れ＞
アセンブリを読み込む、インスタンス生成
↓
プロパティを設定
↓
設定した通信先に対して通信関数を実行

また、プログラム途中で通信先を変える場合は、通信関数を実行する前に該当する部分のプロパティだけを上書きすることで最初と異なる通信先に対してデータのやりとりができます。

11.1. 基本プロパティ

プロパティ名	設定内容	設定例	列挙体名
MCProtocol	プロトコルの選択	2/3/4/5 (形式 2-5) 62/78 (ﾌﾚｰﾑ 3E-4E)	EnumMCProtocol
TIMEOUT	タイムアウト時間	300 (3000msec)	-
QiQ	PLC シリーズ選択	1/2 (Q, QnA, iQ-F/ iQ-R)	EnumQiQ

QiQ は初期値は 1 (Q, QnA, iQ-F 対応) になっております。

11.2. Ethernet プロパティ

プロパティ名	設定内容	設定例	列挙体名
BinaryASCII	通信文字コードの選択	-1/0 (ﾊﾞｲﾅﾘ/ASCII)	EnumBinaryASCII
ActHostAddress	PLC の IP アドレス	“192.168.11.1”	-
ActPortNumber	PLC の IP ポート番号	2000	-

BinaryASCII プロパティは PLC の Ethernet 設定に合わせてください。

11.3. RS-232C ネットワークプロパティ

プロパティ名	設定内容	設定例	列挙体名
CFrame	フレームの選択	1/2/3/4 (1C-4C)	EnumCFrame
ActSourceUnitNumber	RS-232C 局番	10	-
SumCheckRS232C	ｻﾑﾁｪｯｸ設定	False/True	-

○CFrame は初期値では 4 (4C) に設定しており、特に設定しなくても通信可能です。
もし 4 以外に設定した場合、形式 5 においては動作しません。プロトコル形式 5 で通信する場合は必ず 4 (4C)に設定してください。
○ActSourceUnitNumer は PLC の RS-232C 設定に合わせてください。

○SumCheckRS232C は PLC の RS-232C 設定に合わせてください。

11.4. RS-232C 通信プロパティ

以下のプロパティは PLC や端末機器(PC)の RS232C 設定にあわせてください。

プロパティ名	設定内容	設定例	列挙体名
ActCOMPort	端末 COM ポート	“COM3”	-
ActBaudRate	端末ボーレート	9600/19200/115200 など	EnumBaudRate
ActDatabits	端末データビット	7/8 (7bit/8bit)	EnumDatabits
ActStopbits	端末ストップビット	0/1/2/3 (無/1bit/2bit/1.5bit)	EnumStopbits
ActParity	パリティ	0/1/2/3/4 (無/奇/偶/常 1/常 0)	EnumParity

11.5. RS-232C 伝送プロパティ

以下のプロパティは必須ではありませんが RS-232C 端末の設定によっては設定が必要な場合があります。

プロパティ名	設定内容	設定例	列挙体名
ActDtrEnable	端末 Dtr 有効	False/True	-
ActRtsEnable	端末 Rts 有効	False/True	-
ActHandShake	端末ﾊﾝﾄﾞｼｪｲｸ方式	0/1/2/3 (無/XonXoff/RTS /RTS&XonXoff)	EnumHandShake

11.6. PLC ネットワーク構成プロパティ

以下のプロパティは PLC ネットワークが CPU と通信ユニットのみで、通信ユニットと端末（PC）が直結の場合は設定不要ですが、PLC が RS-232C のディジチェーン接続（マルチドロップ接続）や MELSECNET でネットワーク構成されている場合は設定が必要です。

プロパティ名	設定内容
ActSourceStation	自局. RS232C 用
ActSourceUnitNumber	局番. RS232C 用
ActNetworkNumber	ネットワーク番号
ActUnitNumber	PC 番号
ActIONumber	IO 番号
ActConnectUnitNumber	要求先局番

初期値は CPU 直結構成で値が振られています。
設定値については以下の表、付録、 MX Component など参考にしてください。
また設定の便宜として「EnumPLC 構成」「Enum 二重化・マルチ CPU 種別」という列挙体も用意しています。

通信設定の PLC 構成別の設定値一覧(16 進数)

		ネットワーク内			他ネットワーク			
パラメータ名	PLC での用語	1.ｼﾝｸﾞﾙ CPU（初期値）	2.二重化 ﾏﾙﾁ CPU	※1 3.IO 通信	4.設定 管理局	5.現在 管理局	6.-	※1 7.IO 通信
ActNetworkNumber	1.ﾈｯﾄﾜｰｸ番号	00	00	00	対象局	対象局	対象局	中継局
ActUnitNumber	2.PC 番号	FF	FF	FF	7D	7E	対象局	中継局
ActIONumber	3.IO 番号	03FF	※3	中継局 ※2	03FF	03FF	03FF	中継局 ※2
ActConnectUnitNumber	4.局番	00	※3	中継局	00	00	00	中継局

※1.マルチドロップ接続のリモートにアクセスするときは中継局であるマスターに関するパラメータを設定

※2. マルチドロップ中継局の IO 先頭番号を H10 で割った値（たとえば 1F0 なら 1F）

※3. 以下の表

二重化システムまたはマルチ CPU にアクセスする場合の設定値(16 進数)

			ActIONumber	ActUnitNumber
マルチ CPU	管理 CPU		03FF	00
	非管理	1 号機	03E0	
		2 号機	03E1	
		3 号機	03E2	
		4 号機	03E3	
二重化 ｼｽﾃﾑ	CPU	制御系	03D0	
		待機系	03D1	
		A 系	03D2	
		B 系	03D3	
	CC-Link IE	1 号機	03E0	
		2 号機	03E1	
		制御系	03D0	
		待機系	03D1	

11.7. オプションプロパティ

プロパティ名	設定内容	設定例
ActCPUTimeOut	CPU タイムアウト時間	(初期値 0x1000)
TimeOutMessageDisplay	ﾀｲﾑｱｳﾄ時ﾒｯｾｰｼﾞ表示可否	(初期値 False)
TimeOutMessage	ﾀｲﾑｱｳﾄ時表示ﾒｯｾｰｼﾞ内容	“タイムアウトしました”

特段の設定は不要ですが、ActCPUTimeOUT に関しては、通信データが大きいときだけ通信がうまくいかないなどの症状が発生するときに変更するとよいかもしれません。

12. サポートする PLC デバイス

本ライブラリでサポートする PLC デバイスは以下になります。

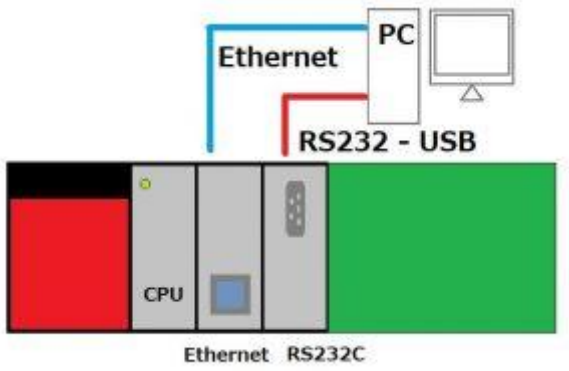
本 DLL でサポートする PLC デバイス種類

表記	名称	デバイスサイズ	アドレス表記
D	データレジスタ	ワード	10 進数
M	内部リレー	ビット	10 進数
X	入力リレー	ビット	16 進数
Y	出力リレー	ビット	16 進数
L	ラッチリレー	ビット	10 進数
B	リンクリレー	ビット	16 進数
W	リンクレジスタ	ワード	16 進数
TS	タイマ接点	ビット	10 進数
TC	タイマコイル	ビット	10 進数
TN	タイマ現在値	ワード	10 進数
CS	カウンタ接点	ビット	10 進数
CC	カウンタコイル	ビット	10 進数
CN	カウンタ現在値	ワード	10 進数
Z	インデックス	ワード	10 進数
V	エッジリレー	ビット	10 進数
ZR	拡張ﾌｧｲﾙﾚｼﾞｽﾀ	ワード	10 進数
R	ファイルレジスタ	ワード	10 進数
F	アナンシェータ	ビット	10 進数

付録

A. Ethernet, RS-232C ユニットと交互に通信するプログラム

以下の図に示す PLC 構成において、PC と Ethernet, RS-232C 通信ユニットにそれぞれ交互に通信する Visual Basic .NET のサンプルプログラムを示します。

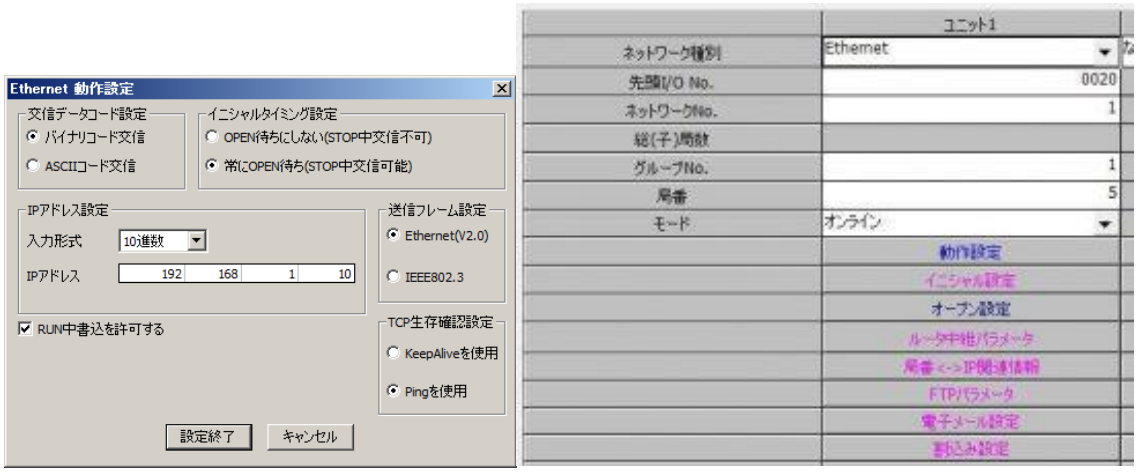


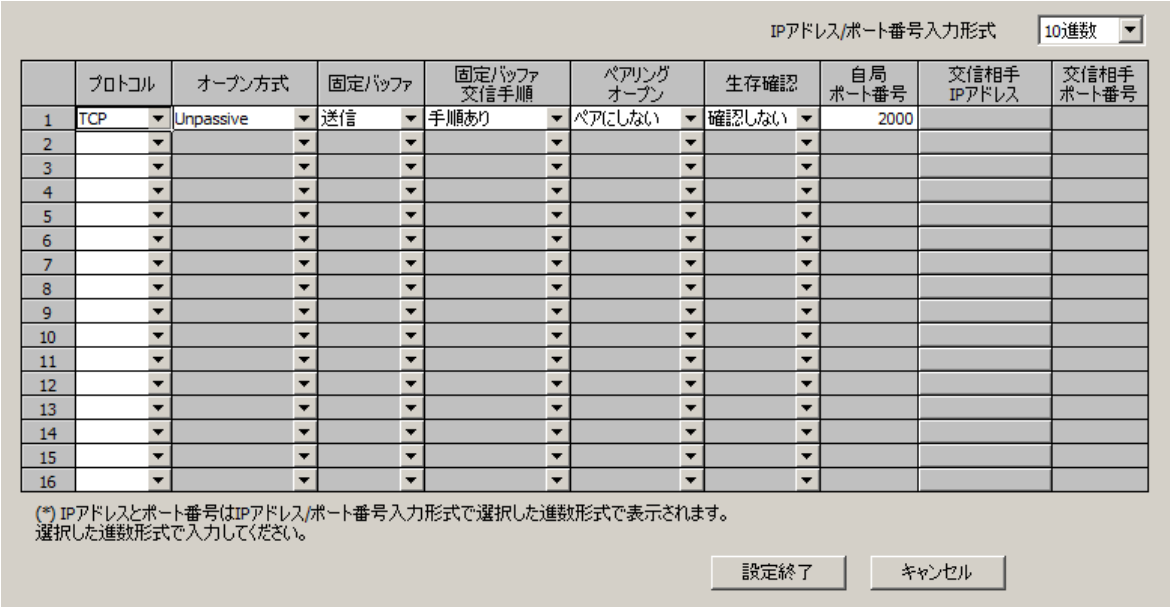

```
Imports SimpleMC.SimpleMC
Public Class SimpleMCDLLSample
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click

        Dim ins As New SimpleMC.SimpleMC
        '####### Ether設定  ####################
ins.ActHostAddress = 192.168.1.10
        ins.ActPortNumber = 2000
        ins.BinaryASCII = EnumBinaryASCII.Binary 'PLCに設定したもの
        '#######  RS232設定  ####################
        ins.ActBaudRate = 19200
        ins.ActDtrEnable = True
        ins.ActParity = EnumParity.Even偶数
        ins.ActCOMPort = COM5
        ins.ActStopbits = EnumStopbits.One
        'ins.SumCheckRS232C = False
        'ins.ActSourceUnitNumber = H10  'RS232C局番
        'ins.ActDatabits = 7  '形式2-4は7bit，形式5は8bit固定なので設定しな
```

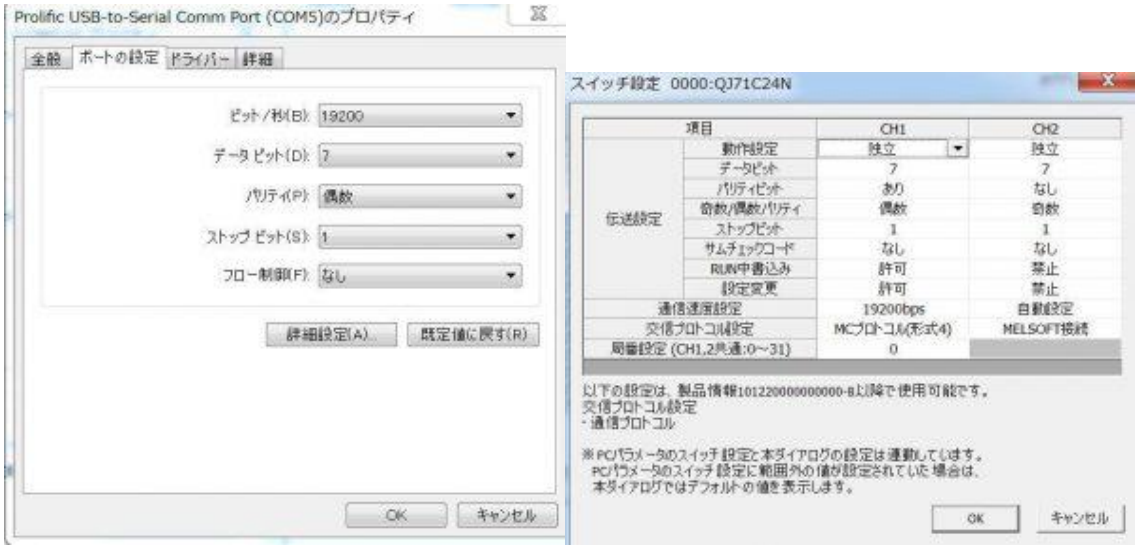
```
いでください
        'ins.ActHandShake = False '必要があれば設定
        'ins.ActRtsEnable= False '必要があれば設定
        '############################

        '##  まずはRS232Cで通信（形式4)
        ins.TIMEOUT = 100
        ins.MCProtocol = EnumMCProtocol.形式4
        ' ins.CFrame = EnumCFrame.フレーム4C '通常は設定不要．なお形式5では
4C固定なので設定しないでください

        MsgBox(ワード単位ブロック書込み)
        ins.WriteRandomBlock2(2, D0, 2, 123, 234, M0, 3, HAAAA, HBBBB,
HFFFF)

        Dim s As String
        Dim randomread As New RetrunRandomWORD
        MsgBox(ランダムワード読込)
        randomread = ins.ReadDeviceRandom2(D0, D1, D2)
        For i = 0 To randomread.UShortArray.Length - 1
            s = s & randomread.UShortArray(i).ToString("D5")
        Next
        MsgBox(s)

        s = ""

        '##  次はEthernetで通信
        ins.TIMEOUT = 100
        ins.MCProtocol = EnumMCProtocol.フレーム3E
        MsgBox(ワード単位ブロック書込み)
        ins.WriteRandomBlock2(2, D0, 2, 123, 234, M0, 3, HAAAA, HBBBB,
HFFFF)
        MsgBox(ランダムワード読込)
        randomread = ins.ReadDeviceRandom2("D0", "D1", "D2")

        For i = 0 To randomread.UShortArray.Length - 1
```

```
            s = s & randomread.UShortArray(i).ToString("D5")
        Next
        MsgBox(s)
    End Sub
End Class
```

【このときの設定例】

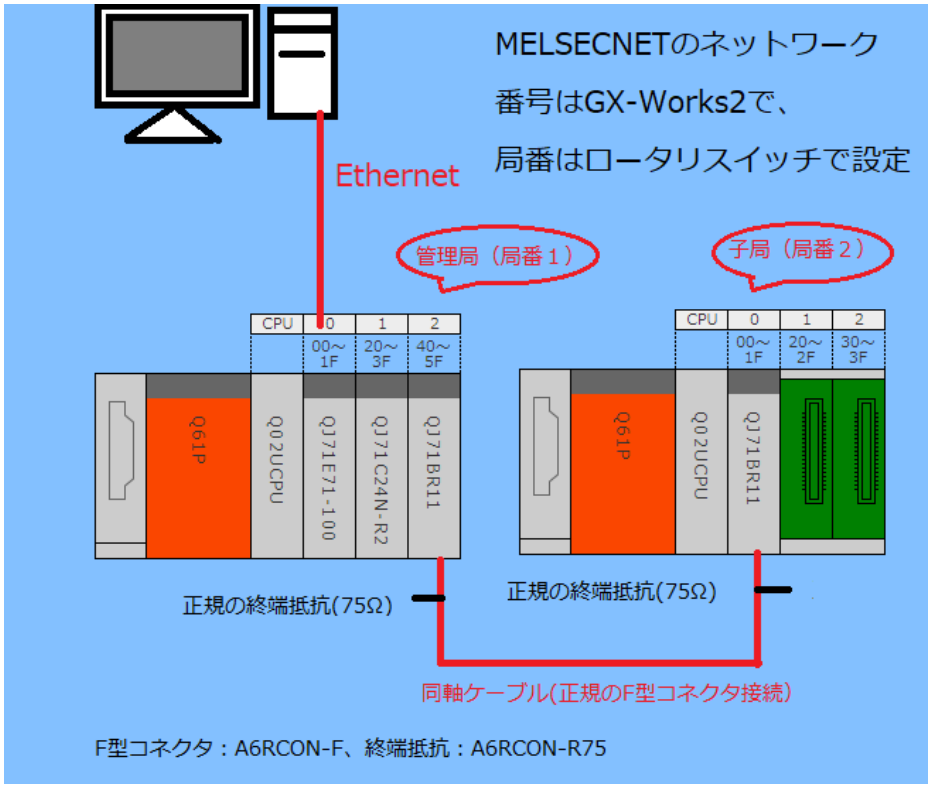






B. MELSECNET 構成の子局 PLC と通信するプログラム(一部)

以下の図のように MELSECNET ネットワーク構成にある子局 PLC のデバイスにアクセスする Visual Basic .NET プログラムの要点を示します。



```
        Dim ins As New SimpleMC.SimpleMC 'インスタンスの生成
        ' ins.TimeOutMessage = "タイムアウトしました"
        ' ins.TimeOutMessageDisplay = True
        ins.TIMEOUT = 10000                    '通信の最大待ち時間
        ins.MCProtocol = EnumMCProtocol.フレーム3E  'Ethernet(3E,4E)のﾊﾞｲﾅﾘ，形式5(4Cﾌﾚｰﾑﾊﾞｲﾅﾘ)に対応しています。
        ins.BinaryASCII = EnumBinaryASCII.Binary 'GX-Works2で設定したMCプロトコルの交信データコード
        ins.ActHostAddress = "192.168.1.10"  'EthernetユニットのIPアドレス
        ins.ActPortNumber = 2000 'Ethernetユニットのポート番号
        ins.ActNetworkNumber = 2  'MELSECNET管理局、子局のネットワークの番号を2と設定した。

        '初期設定を終えたら、以下で管理局、子局それぞれにアクセスする

        ins.ActUnitNumber = 1  'ロータリスイッチで設定した管理局の局番
        MsgBox(ins.ReadDeviceBlock2("D4", 1).UShortArray(0))          '局番1のCPUのレジスタD4の値を読みだして表示

        ins.ActUnitNumber = 2 'ロータリスイッチで設定した子局の局番
        MsgBox(ins.ReadDeviceBlock2("D4", 1).UShortArray(0))          '局番2のCPUのレジスタD4の値を読みだして表示
```

【このときの設定例】