|  |
| --- |
| **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  クマのぬいぐるみ  自動的に生成された説明 |

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 4](#_Toc55445824)

[2. 対応通信プロトコル 4](#_Toc55445825)

[2.1. Ethernetプロトコル 4](#_Toc55445826)

[2.2. RS-232Cプロトコルとデータビット 4](#_Toc55445827)

[2.3. RS-232C 伝送速度( bps) 4](#_Toc55445828)

[2.4. RS-232C パリティ 4](#_Toc55445829)

[3. 対応PLC 5](#_Toc55445830)

[4. 開発環境 5](#_Toc55445831)

[5. インテリセンスxml 6](#_Toc55445832)

[6. 端末(PC)の設定 6](#_Toc55445833)

[7. アセンブリ 7](#_Toc55445834)

[8. クラス（ステータス、データ） 7](#_Toc55445835)

[8.1. ステータスクラス ReturnBasic 8](#_Toc55445836)

[8.2. データクラス (ReturnBasicを包含) 9](#_Toc55445837)

[8.3. データクラス ReturnWORD 9](#_Toc55445838)

[8.4. データクラス ReturnDWORD 9](#_Toc55445839)

[8.5. データクラス ReturnBIT 10](#_Toc55445840)

[8.6. データクラス RetrunRandomWORD※3 10](#_Toc55445841)

[9. 通信関数 11](#_Toc55445842)

[9.1. WriteDeviceBlock2(ワード単位一括書込) 11](#_Toc55445843)

[9.2. WriteDeviceBlock2BIT(ビット単位一括書込) 12](#_Toc55445844)

[9.3. WriteDeviceBlock2ArrayBit (ビット単位一括書込) 13](#_Toc55445845)

[9.4. WriteDeviceBlock2DWord (ダブルワード単位一括書込) 15](#_Toc55445846)

[9.5. WriteDeviceRandom2(ワード単位ランダム書込) 16](#_Toc55445847)

[9.6. WriteDeviceRandom2DWord(ダブルワード単位ランダム書込) 17](#_Toc55445848)

[9.7. WriteDeviceRandom2ArrayBIT(ビット単位ランダム書込) 18](#_Toc55445849)

[9.8. WriteRandomBlock2(ワード単位ランダムブロック書込) 19](#_Toc55445850)

[9.9. ReadDeviceBlock2(ワード単位一括読込) 19](#_Toc55445851)

[9.10. ReadDeviceBlock2DWord(ダブルワード単位一括読込) 20](#_Toc55445852)

[9.11. ReadDeviceBlock2BIT(ビット単位一括読込) 22](#_Toc55445853)

[9.12. ReadDeviceBlock2ArrayBIT(ビット単位一括読込) 23](#_Toc55445854)

[9.13. ReadDeviceRandom2(ワード単位ランダム読込) 24](#_Toc55445855)

[10. 符号付データを変換する関数(おまけ) 25](#_Toc55445856)

[10.1. 符号無16ビットデータへ変換 25](#_Toc55445857)

[10.2. 符号無32ビットデータへ変換 25](#_Toc55445858)

[10.3. 符号無16ビットデータから変換 26](#_Toc55445859)

[10.4. 符号無32ビットデータから変換 26](#_Toc55445860)

[11. 設定プロパティ 27](#_Toc55445861)

[11.1. 基本プロパティ 28](#_Toc55445862)

[11.2. Ethernetプロパティ 28](#_Toc55445863)

[11.3. RS-232Cネットワークプロパティ 28](#_Toc55445864)

[11.4. RS-232C通信プロパティ 29](#_Toc55445865)

[11.5. RS-232C伝送プロパティ 29](#_Toc55445866)

[11.6. PLCネットワーク構成プロパティ 30](#_Toc55445867)

[11.7. オプションプロパティ 31](#_Toc55445868)

[12. サポートするPLCデバイス 32](#_Toc55445869)

[付録 33](#_Toc55445870)

[A. Ethernet, RS-232Cユニットと交互に通信するプログラム 33](#_Toc55445871)

[B. MELSECNET構成の子局PLCと通信するプログラム(一部) 36](#_Toc55445872)

# 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。

# 対応通信プロトコル

## Ethernetプロトコル

|  |  |  |
| --- | --- | --- |
|  | 文字コード | |
| ﾌﾚｰﾑ | バイナリ | ASCII |
| 3E | 〇 | 〇 |
| 4E | 〇 | 〇 |

## 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 | 〇 | 〇 | 〇 | 〇 | - | 〇 | 7bit |
| 形式3 | 〇 | 〇 | 〇 | 〇 | - | 〇 | 7bit |
| 形式4 | 〇 | 〇 | 〇 | 〇 | - | 〇 | 7bit |
| 形式5 | - | - | - | 〇 | 〇 | - | 8bit |

## 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

## RS-232C パリティ

奇数、偶数を RS-232C端末に合わせて設定可能

# 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） |

# 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。

# 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。

# 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。

# 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 |

# 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。

RetrunRandomWORD

ReturnDWORD

ReturnWORD

ReturnBIT

ArrayObject

UIntegerArray

UShortArray

ByteArray

ReturnBasic

Connected/Finished/DataOK/Status/ErrorCode

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。

## 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 |

## 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。

## 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

## 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

## 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

## 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。

# 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。

## 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。

## 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。

## 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。

## 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。

## 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

## 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

## 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

## WriteRandomBlock2(ワード単位ランダムブロック書込)

削除

## 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、

「100001001100001」と表示される。

## 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)。

## 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

「000000000000001」と表示される(LSBがM16の値)。

## 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)) |

## 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)) |

# 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。

## 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

## 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

## 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)

## 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)

# 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。

## 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。

## 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。

## 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 | - |

〇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。

〇ActSourceUnitNumer は PLC のRS-232C設定に合わせてください。

〇SumCheckRS232C は PLC のRS-232C設定に合わせてください。

## 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 |

## 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 |

## 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 |

## 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。

# 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。

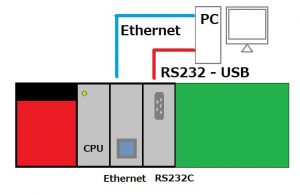
**本DLLでサポートするPLCデバイス種類**

|  |  |  |  |
| --- | --- | --- | --- |
| 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

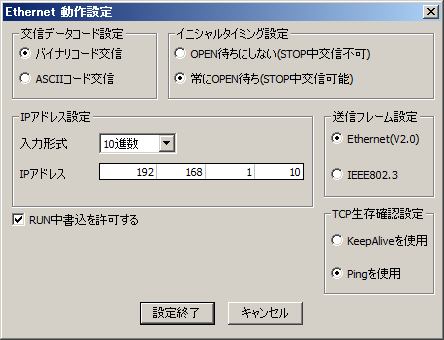
## Ethernet, RS-232Cユニットと交互に通信するプログラム

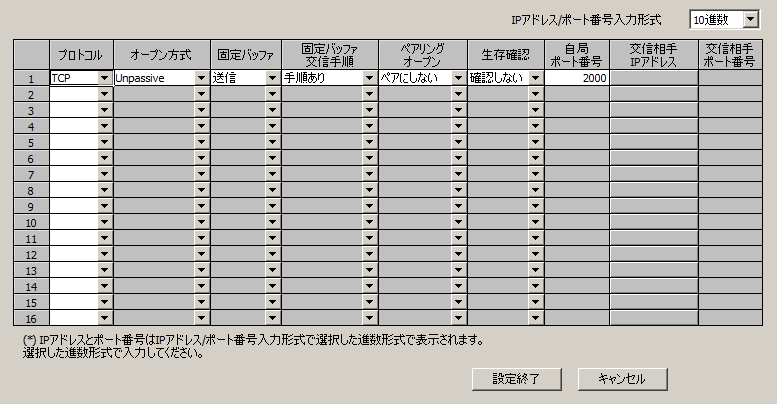
以下の図に示すPLC構成において、PCとEthernet, RS-232C通信ユニットにそれぞれ交互に通信する Visual Basic .NET のサンプルプログラムを示します。

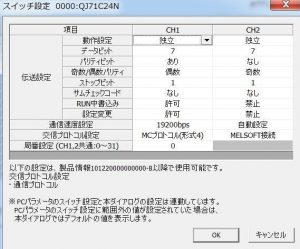


|  |
| --- |
| Imports SimpleMC.SimpleMC  Public Class SimpleMCDLLSample  Private Sub Button2\_Click(sender As Object, e As EventArgs) Handles Button2.Click  Dim ins As New SimpleMC.SimpleMC  '####### Ether設定 #####################  ins.ActHostAddress = 192.168.1.10  ins.ActPortNumber = 2000  ins.BinaryASCII = 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】

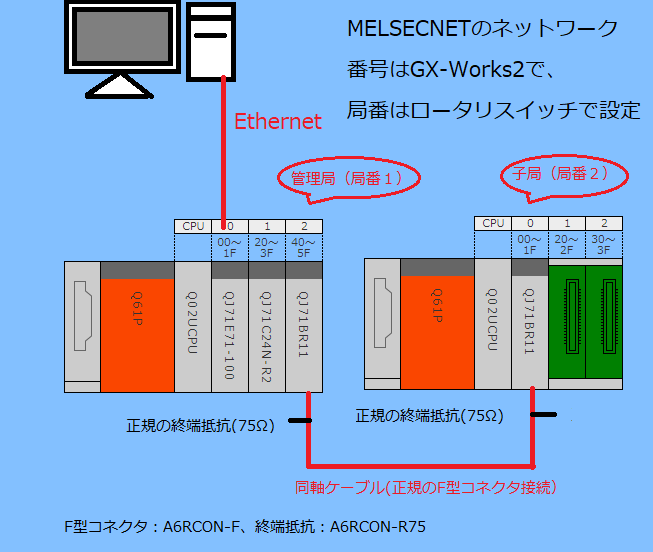






## MELSECNET構成の子局PLCと通信するプログラム(一部)

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】

