1 はじめに

MelsecCommunication ライブラリは、三菱電機製 PLC (Q シリーズ, iQ-R シリーズ) と PC との間でデータ授受を必要とする際、PC 側ソフトウェアの製作簡便化を図るためのライブラリです。
ご承知の通り、PLC と PC の間でデータ授受を行うユーザープログラムを作成する場合、MX Component の導入を検討するのが最も妥当です。
一方、装置物件を取り巻く様々な要因で MX Component の採用が難しいケースもあります。
そこで、MX Component には及ばないながらも、とは言え、現実的に使用頻度の高い交信に限ってミドルウェアっぽく仕上げたものが、本 MelsecCommunication ライブラリです。
元々は私自身の業務物件で必要に迫られ作ったものですが、同業諸氏の皆様になんらかお役に立てればと思い公開することにしました。

2 前提条件

MelsecCommunication ライブラリはあくまでも MC プロトコル /3E フレームを話すだけしか出来ません。そのため...

- PC 側処理用プログラム / ユーザーインターフェースを製作できる環境が必須です
- PLC 側では通信のためのプログラム的アプローチは必要ありませんが、IP アドレス、ポート設定など、最低限の CPU 設定は必要です
- 例外の発生箇所 / 原因を突き止めやすくするため (と称して)、例外処理は最小限にとどめています
- 非同期処理は利用側コードで記述する方が使い勝手が良いだろうと思っています
- 処理内容によっては、PLC 側ソフトウェアの工夫も必要かもしれません
- Windows デスクトップアプリケーション、PLC システム双方において、構築経験をお持ちの方を対象としています
- FA 特有の困難を突破できる体力を持ち、頑張れば何とかできるはずなのに、なんか面倒くさくて触れないようにしている方も対象としています

本ライブラリで達成できない通信が必要になる場合や、高い確実性が求められる場合には、MX Component に頼った方がよいと思います。ぜひそうしてください。

3. 対象環境

- PC 側: .NET Framework 4 以上 (プラットフォーム (32bit / 64bit) は不問)
- PLC 側: MC プロトコル /3E フレームを解釈できるユニット (Ethernet ポート付き CPU、または E71)
- ネットワーク: スイッチングハブを介して、PC と PLC が接続されていること

4. 仕様

TCP 接続、バイナリコード交信専用です。

本ライブラリでは、下記コマンドをサポートします。

- ワード単位の一括読み出し (Q シリーズ: 0401 0000, iQ-R シリーズ: 0401 0002)
- ワード単位の一括書き込み (Q シリーズ: 1401 0000, iQ-R シリーズ: 1401 0002)
- ビット単位の一括読み出し (Q シリーズ: 0401 0001, iQ-R シリーズ: 0401 0003)
- ビット単位の一括書き込み (Q シリーズ: 1401 0001, iQ-R シリーズ: 1401 0003)
- ワード単位のランダム読み出し (Q シリーズ: 0403 0000, iQ-R シリーズ: 0403 0002) ※モニタ条件指定は非対応
- ワード単位のランダム書き込み (Q シリーズ: 1402 0000, iQ-R シリーズ: 1402 0002) ※モニタ条件指定は非対応
- ビット単位のランダム書き込み (Q シリーズ: 1402 0001, iQ-R シリーズ: 1402 0003)
- ビット単位のランダム読み込み (該当コマンド無し/ ビット一括書き込みコマンドの応用クラス)

5. 製作環境と動作確認

下記環境で製作、動作確認を行いました。

- Windows 10 Professional 64bit 21H1 19043.1110 (～1.1.0)
  Windows 11 Professional 64bit 21H2 22000.652 (1.2.1～)
- Intel EXPI9402PT
- Visual Studio Community 2019 Ver.16.10.3
  Visual Studio Community 2022 Ver.17.1.6
- Q06UDVCPU (Q35B へ単体装着) ※構成図をドキュメント末尾に掲載します。
  R01CPU (R33B, RH42C4NT2P, R60TD8)
  R04CPU (R38B へ単体装着)

また、下記ドキュメントを元に製作しました。

- MELSEC コミュニケーションプロトコル リファレンスマニュアル
  SH(名)-080003-AG(2104)KWIX/ MC-PROTOCOL-R/ 13JQ34
- QnUCPU ユーザーズマニュアル (内蔵 Ethernet ポート 通信編)
  SH(名)-080806-Y(2007)MEE/ QNUDEHCPU-U-ET-J/ 13JY96
- MELSEC iQ-R Ethernet ユーザーズマニュアル (応用編)
  SH(名)-081253-S(2007)MEE/ R-ETHER-U-OU-J/ 13J2B5

6. 免責事項

MelsecCommunication ライブラリは GPL として公開します。

MelsecCommunication ライブラリの使用に起因する、あらゆる損害、障害、不都合な事象について責任を負いかねます。

利用者の責任において使用してください。

7. 導入

- ご自身のプロジェクトに "MelsecCommunication.vb" をインポートするか、この中身すべてをコピペします。
  それだけです。
- フォームベースのコード (frmMain.vb 他) は単なるサンプルです。
- ソースコードはご覧の通り VB です。
  VB 以外の言語で進められている場合は、ソリューションに VB プロジェクトを追加し、DLL として振る舞うように持ち込むのが手っ取り早いかもしれません。
- 名前空間やクラス名が気に入らなければ、この段階でお好みに変更しておくとあとあと気分が良いでしょう。

8. 使い方

サンプルを追いかけてもらうのが百聞は一見にしかず、ですが...
おおまかな流れは下記のようになります。
(アプリ起動) → (1) → (2) → (3 ～ 4 を適宜繰り返し) → (5) → (アプリ終了)

1) 先ず、MelsecCPU オブジェクトを作成します。

コンストラクターでは CPU シリーズ (Q または R) を引数として与えます。
このオブジェクトはたいていの場合、アプリケーション存命中は生かしたままだと思います。

2) (1) で作成したオブジェクトの Connect/ConnectAsync メソッドにて、PLC へ接続します。

同期処理の "Connect" メソッドと、非同期処理の "ConnectAsync" メソッドを用意しました。
どちらのメソッドも、PLC の IP アドレスとポート番号を引数として与えます。
非同期メソッドを使用する場合、接続完了/失敗時にイベントが発生しますので、イベントハンドラーを事前に設定してください。

3) データ読み出し

(サンプルアプリケーション :: frmMain.cmdLoad_Click, frmDeviceOperation.BitDeviceWatchWorker_DoWork 参照)

ワードデバイスの場合、WordReader オブジェクトを作成して読み出します。
コンストラクターには (1) で作成したオブジェクトを与えます。
WordReader クラスは Disposable なので、Using で運用すると良いと思います。
Read 関数は、先頭アドレス、デバイス種別、データ数を指定すると Byte 配列を返してきます。
Read 関数の戻り値は主に次の方法で取り扱ってください。

- **A:** DataBuilder のコンストラクターに与えます。
  データの取り出しが容易になりますが、取り出し順序・型は PLC プロジェクトと整合させてください。
- **B:** DeviceConverter クラスの共有メソッドで適切なデータ型へ変換します。
  手間ですが自由度は高いです。

4) データ書き込み

(サンプルアプリケーション :: frmMain.cmdUpdate_Click, DeviceOperation_Click 参照)

ワードデバイスの場合、WordWriter オブジェクトを作成して書き込みます。
コンストラクターには (1) で作成したオブジェクトを与えます。
WordWriter クラスは Disposable なので、Using で運用すると良いと思います。
Write メソッドは、先頭アドレス、デバイス種別、書き込みデータ (DataBuilder 型 または Byte 配列) を指定すると PLC へデータを書き込みます。
書き込みデータは Write メソッド実行前に用意しておく必要があり、主に次の方法で用意します。

- **A:** DataBuilder の Set*** メソッドでデータを構築する。
- **B:** Byte 配列を用意し、DeviceConverter クラスの共有メソッドで元データを積み重ねていく。

5) アプリケーションの終了に先だって...

(1) の MelsecCPU オブジェクトの Disconnect メソッドで接続を解除してください。

9. サンプルアプリケーション

謎の加熱装置を想定したサンプルです。
レシピデータをタッチパネル上だけでなく、PC でも編集管理できる点がこの装置のセールスポイントです。

1) 装置概要
   - 容器を所定温度に加熱し、容器内のワークを乾燥させる装置です。
   - 運転開始後、所定時間経過すると自動的に運転を停止します。
   - 運転期間中、容器内に N2 を所定の流量で導入し続けます。
   - 運転期間中、ブロワーにて容器内を撹拌します。
   - 運転レシピを PLC のローカルメモリに 10 種保持します。
   - 遠隔地から付帯設備（ヒーター・N2 バルブ・ブロワー）を個別に操作できます。

2) レシピ内容
   - レシピ名（10 ワード分/ ターミネーター含む）
   - 運転時間（1 ワード）
   - 目標温度（1 ワード）
   - N2 流量（1 ワード）

3) レシピマップ
   - D1000〜D1019: レシピ 1
     - +0 ... +9: レシピ名（文字列）
     - +10: 運転時間（整数, 分）
     - +11: 目標温度（整数, ℃, 10 倍値）
     - +12: N2 流量（整数, LCCM, 10 倍値）
     - +13〜+19: 予約
   - D1020〜D1039: レシピ 2
   - D1040〜D1059: レシピ 3
   - D1060〜D1079: レシピ 4
   - D1080〜D1099: レシピ 5
   - D1100〜D1119: レシピ 6
   - D1120〜D1139: レシピ 7
   - D1140〜D1159: レシピ 8
   - D1160〜D1179: レシピ 9
   - D1180〜D1199: レシピ 10

4) 機器遠隔操作
   M20〜、M30〜 をリモートデバイス（RX, RY）に見立てて、
   - M10: ヒーター動力 MC
   - M11: N2 バルブ
   - M12: ブロワー動力 MC

   を手動操作できます。

検証用の実機環境を用意できる場合、下記設定のプロジェクトを転送しますと動作確認を行えます。
下記に挙げる項目は必要最小限の設定 (必須設定) です。
<PC パラメーター>

- IP アドレス: 192.168.100.39 (*)
- 更新データコード: バイナリコード交信
- RUN 中書込を許可する (FTP と MC プロトコル) = ON
- オープン設定 (下記設定を含ませる)

    プロトコル: TCP
    オープン方式: MC プロトコル
    自局ポート番号: 8192 (*)

*印: PC 側プログラムで整合させる限り任意の値で可
なお、同梱の GX-Works プロジェクトは上記の設定に加え、付帯機器の手動操作用ラダーを記述したものです。

10. リファレンス

MelsecCPU クラス

MelsecCPU クラスは、接続先 PLC との通信機能、PLC の仕様条件を保持します。

コンストラクター

MelsecCPU(CPUTypeFlags)

書式

```
Public Sub New(Platform As CPUTypeFlags)
```

パラメーター

Platform	対象 PLC を特定する、CPUTypeFlags 列挙体を指定します。

解説

指定された CPUTypeFlags に基づき、クラスの新しいインスタンスを初期化します。

プロパティ

CommClient As TcpClient

内部で保持している TcpClient を取得します。

Platform As CPUTypeFlags

接続先 CPU の種類を取得します。

メソッド

Connect(String, Integer) As TcpClient

書式

```
Public Function Connect(Address As String, Port As Integer) As TcpClient
```

パラメーター

Address	対象 PLC の IP アドレスを指定します。
Port	対象 PLC へ接続する際のポート番号を指定します。

戻り値

正常終了	PLC への接続情報を保持する TcpClient オブジェクト。
返信異常	Nothing

解説

接続先 CPU の IP アドレスとポート番号を指定して、同期処理にて接続を試みます。

ConnectAsync(String, Integer)

書式

```
Public Sub ConnectAsync(Address As String, Port As Integer)
```

パラメーター

Address	対象 PLC の IP アドレスを指定します。
Port	対象 PLC へ接続する際のポート番号を指定します。

解説

接続先 CPU の IP アドレスとポート番号を指定して、非同期処理にて接続を試みます。

接続に成功した場合は ConnectionCreated イベントが発生します。

接続に失敗した場合は ConnectionFailed イベントが発生します。

Disconnect()

書式

```
Public Sub Disconnect()
```

解説

内部で保持している TcpClient を Close したあと、オブジェクトを Dispose します。

イベント

ConnectionCreated(Object, ConnectionCreatedEventArgs)

PLC との非同期接続が成功すると発生します。

ConnectionCreatedEventArgs の Client プロパティにより、接続情報を保持する TcpClient を取得できます。

ConnectionFailed(Object, EventArgs)

PLC との非同期接続が失敗すると発生します。

例

次の例では、PLC オブジェクトを作成し、同期処理で PLC へ接続します。

```
Private _QCPU As MelsecCPU

'同期接続実行
Private Sub Connect()
   _QCPU = New MelsecCPU(CPUTypeFlags.QSeries)
   _QCPU.Connect("192.168.100.39", 8192)
End Sub
```

まず、PLC への接続情報を保持する MelsecCPU オブジェクトを作成します。この操作は同期接続、非同期接続どちらの方式でも共通です。引数として接続先 PLC のタイプ (Q または iQ-R) を指定します。

次に、MelsecCPU オブジェクトの Connect メソッドに、接続先 PLC の IP アドレスと接続ポートを与えて実行します。

接続に成功した場合、接続の基底となる TcpClient を戻り値として取得できます。

次の例では、PLC オブジェクトを作成し、非同期処理で PLC へ接続します。

まず、PLC への接続情報を保持する MelsecCPU オブジェクトを作成します。これは前述の同期接続と同じ作業です。

次に、非同期処理の結果を取得するイベントハンドラーを設定します。接続完了時には ConnectionCreated イベント、失敗時には ConnectionFailed イベントが発生します。

最後に、MelsecCPU オブジェクトの ConnectAsync メソッドに、同期接続と同様のパラメータ (IP アドレス、接続ポート) を与えて ConnectAsync メソッドを実行します。

```
Private _QCPU As MelsecCPU

'非同期接続実行
Private Sub ConnectAsync()
   _QCPU = New MelsecCPU(CPUTypeFlags.QSeries)
   AddHandler _QCPU.ConnectionCreated, AddressOf ConnectionCreated
   AddHandler _QCPU.ConnectionFailed, AddressOf ConnectionFailed
   _QCPU.ConnectAsync("192.168.100.39", 8192)
End Sub

'非同期接続成功時
Private Sub ConnectionCreated(sender As Object, e As ConnectionCreatedEventArgs)
   Dim Connection As TcpClient
   RemoveHandler _QCPU.ConnectionCreated, AddressOf ConnectionCreated
   RemoveHandler _QCPU.ConnectionFailed, AddressOf ConnectionFailed
   TcpClient = e.Client
```

```
End Sub

'非同期接続失敗時
Private Sub ConnectionFailed(sender As Object, e As EventArgs)
   RemoveHandler _QCPU.ConnectionCreated, AddressOf ConnectionCreated
   RemoveHandler _QCPU.ConnectionFailed, AddressOf ConnectionFailed
   _QCPU.Dispose()
   _QCPU = Nothing
End Sub
```

接続完了イベント (ConnectionCreated) 発生時には、引数 e As ConnectionCreatedEventArgs の Client プロパティを参照することで、接続の基底となる TcpClient オブジェクトにアクセスできます。
この TcpClient オブジェクトは、同期接続 (Connect メソッド) の戻り値と同一内容です。

次に、接続済みの MelsecCPU オブジェクトを PLC から切断します。
PLC への接続を同期、非同期のどちらで行った場合でも共通です。

```
Private Sub Disconnect()
   _QCPU.Disconnect()
   _QCPU.Dispose()
   _QCPU = Nothing
End Sub
```

Disconnect メソッドの実行により、内部の TcpClient オブジェクトの接続は閉じられ、Dispose されます。
次に MelsecCPU オブジェクト自体を Dispose します。
なお、MelsecCPU の Dispose メソッドを実行すると、内部の TcpClient に対して Disconnect、および Dispose が実施されるため、ユーザーコードで明示的に Disconnect メソッドを実行しなくても構いません。

DataBuilder クラス

DataBuilder クラスは PLC への連続書き込み / 連続読み出しのデータ処理を簡略化する機能を提供します。

コンストラクター

DataBuilder()

書式

```
Public Sub New()
```

解説

クラスの新しいインスタンスを初期化します。

DefaultEncoding プロパティは ShiftJIS で初期化されます。

DataBuilder(Byte())

書式

```
Public Sub New(Source As Byte())
```

パラメーター

Source　　PLC データを表す Byte 配列を指定します。

解説

クラスの新しいインスタンスを初期化し、指定した Byte 配列を内部バッファへコピーします。

DefaultEncoding プロパティは ShiftJIS で初期化されます。

プロパティ

Bytes As Integer

内部バッファのデータサイズ (Byte 単位) を取得します。

DefaultEncoding As Encoding

SetString メソッド、および GetString メソッドでテキスト変換に使用するエンコード形式を取得または設定します。

メソッド

GetUInt16() As UInt16

内部バッファの先頭から 2 バイトのデータを消費して(*)、符号無し 16 ビット整数に変換して取得します。

内部バッファが 2 バイト未満の場合、ConvertDataShortageException が発生します。

GetInt16() As Int16

内部バッファの先頭から 2 バイトのデータを消費して(*)、符号付き 16 ビット整数に変換して取得します。

内部バッファが 2 バイト未満の場合、ConvertDataShortageException が発生します。

GetUInt32() As UInt32

内部バッファの先頭から 4 バイトのデータを消費して(*)、符号無し 32 ビット整数に変換して取得します。

内部バッファが 4 バイト未満の場合、ConvertDataShortageException が発生します。

GetInt32() As Int32

内部バッファの先頭から 4 バイトのデータを消費して(*)、符号付き 32 ビット整数に変換して取得します。

内部バッファが 4 バイト未満の場合、ConvertDataShortageException が発生します。

GetSingle() As Single

内部バッファの先頭から 4 バイトのデータを消費して(*)、単精度浮動小数点型値に変換して取得します。

内部バッファが 4 バイト未満の場合、ConvertDataShortageException が発生します。

GetDouble() As Double

内部バッファの先頭から 8 バイトのデータを消費して(*)、倍精度浮動小数点型値に変換して取得します。

内部バッファが 8 バイト未満の場合、ConvertDataShortageException が発生します。

GetString(UShort) As String

内部バッファの先頭から指定したワード数分のデータを消費して(*)、文字列に変換して取得します。

バイト列から文字列への変換は DefaultEncoding により行われます。

このメソッドを呼び出す前に、必要に応じて DefaultEncoding を設定してください。

内部バッファの要素数が変換に必要となるワード相当数未満の場合、ConvertDataShortageException が発生します。

GetAnyBytes(UShort) As Byte()

内部バッファの先頭から指定したワード数分のデータを取得(*) します。

内部バッファの要素数が要求されたワード相当数未満の場合、ConvertDataShortageException が発生します。

> 注記 (*)
> `Get***` メソッドが正常に終了した場合、変換 (`GetAnyBytes` の場合は取得) に使用したデータは内部バッファから消去されます。
> すなわち、`Get***` メソッドを実行するたびに内部バッファのデータは先頭から順次減少し、最終的に要素数は `0` になります。
> `Get***` メソッドを利用して値を取得する場合、`PLC` のレジスタに整合する順序・型に注意して実行してください。

SetUInt16(Object)

指定されたデータを符号なし 16 ビット整数へ変換し、内部バッファの末尾に追加します。

指定されたデータの変換に失敗した場合、InvalidCastException が発生します。

SetInt16(Object)

指定されたデータを符号付き 16 ビット整数へ変換し、内部バッファの末尾に追加します。

指定されたデータの変換に失敗した場合、InvalidCastException が発生します。

SetUInt32(Object)

指定されたデータを符号なし 32 ビット整数へ変換し、内部バッファの末尾に追加します。

指定されたデータの変換に失敗した場合、InvalidCastException が発生します。

SetInt32(Object)

指定されたデータを符号付き 32 ビット整数へ変換し、内部バッファの末尾に追加します。

指定されたデータの変換に失敗した場合、InvalidCastException が発生します。

SetSingle(Object)

指定されたデータを単精度浮動小数点型値に変換し、内部バッファの末尾に追加します。

指定されたデータの変換に失敗した場合、InvalidCastException が発生します。

SetDouble(Object)

指定されたデータを倍精度浮動小数点型値に変換し、内部バッファの末尾に追加します。

指定されたデータの変換に失敗した場合、InvalidCastException が発生します。

SetString(Object, UShort)

指定されたデータを文字列に変換し、DefaultEncoding により得られるバイト列を内部バッファの末尾に追加します。

このメソッドを呼び出す前に、必要に応じて DefaultEncoding を設定してください。

変換された文字列長が指定されたワード長を超える場合、切り詰められます。

変換された文字列長が指定されたワード長に満たない場合、不足分は Null 値で埋められます。

ワード長にはターミネーター (1 ワードの Null 値) を含みます。たいていの場合、変換する文字列長に +1 した値を指定します。

SetNull(UShort)

指定されたワード長分の Null 値を内部バッファの末尾に追加します。

SetAnyBytes(Byte())

指定されたバイト列を内部バッファの末尾に追加します。

ToArray As Byte()

現在の内部バッファを取得します。

WordAddress クラス

WordAddress クラスはワードデバイスのアドレス情報を保持します。

コンストラクター

WordAddress(Integer, WordDeviceFlags)

書式

```
Public Sub New(Offset As Integer, DeviceCode As WordDeviceFlags)
```

パラメーター

Offset	ワードデバイスのオフセット値を指定します。
DeviceCode	ワードデバイスの種別を指定します。

解説

指定されたパラメーターを基にクラスの新しいインスタンスを初期化します。

WordAddress(String)

書式

```
Public Sub New(Address As String)
```

パラメーター

Address	ワードデバイスをアドレス表記で指定します。

解説

指定されたパラメーターを基にクラスの新しいインスタンスを初期化します。

プロパティ

Offset As Integer

ワードデバイスのオフセット値を取得します。

DeviceCode As WordDeviceFlags

ワードデバイスの種別を取得します。

例

次の例では、どちらも D1000 を表します。

```
Dim D1000A As WordAddress = New WordAddress(1000, WordDeviceFlags.D)
Dim D1000B As WordAddress = New WordAddress("D1000")
```

BitAddress クラス

BitAddress クラスはビットデバイスのアドレス情報を保持します。

コンストラクター

BitAddress(Integer, BitDeviceFlags)

書式

```
Public Sub New(Offset As Integer, DeviceCode As BitDeviceFlags)
```

パラメーター

Offset	ビットデバイスのオフセット値を指定します。
DeviceCode	ビットデバイスの種別を指定します。

解説

指定されたパラメーターを基にクラスの新しいインスタンスを初期化します。

BitAddress(Integer, Integer, BitDeviceFlags)

書式

```
Public Sub New(Channel As Integer, Offset As Integer, DeviceCode As BitDeviceFlags)
```

パラメーター

Channel	ビットデバイスのチャンネル値を指定します。
Offset	ビットデバイスのオフセット値を指定します。
DeviceCode	ビットデバイスの種別を指定します。

解説

指定されたパラメーターを基にクラスの新しいインスタンスを初期化します。

BitAddress(String)

書式

```
Public Sub New(Address As String)
```

パラメーター

Address	ビットデバイスをアドレス表記で指定します。

解説

指定されたパラメーターを基にクラスの新しいインスタンスを初期化します。

プロパティ

Offset As Integer

ビットデバイスのオフセット値を取得します。

DeviceCode As BitDeviceFlags

ビットデバイスの種別を取得します。

例

次の例では、どれもビット Y2C を表します。

```
Dim Y2C_A As BitAddress = New BitAddress(44, BitDeviceFlags.Y)
Dim Y2C_B As BitAddress = New BitAddress(&H2C, BitDeviceFlags.Y)
Dim Y2C_C As BitAddress = New BitAddress(2, 12, BitDeviceFlags.Y)
Dim Y2C_D As BitAddress = New BitAddress(&H2, &HC, BitDeviceFlags.Y)
Dim Y2C_E As BitAddress = New BitAddress("Y2C")
```

WordDeviceCollection クラス

WordDeviceCollection クラスは、ワードデバイスにランダムアクセスする際の対象デバイス情報を保持します。

コンストラクター

WordDeviceCollection()

書式

```
Public Sub New()
```

解説

クラスの新しいインスタンスを初期化します。

プロパティ

Items As List(Of WordDeviceDefinition)

内部で保持するデバイス情報を取得します。

SingleWordDevices As List(Of WordDeviceDefinition)

内部で保持するデバイス情報の内、シングルワードデバイス情報のみ取得します。

DoubleWordDevices As List(Of WordDeviceDefinition)

内部で保持するデバイス情報の内、ダブルワードデバイス情報のみ取得します。

メソッド

DataLength(AccessModeFlags) As Integer

内部で保持するデータを伝文形式に変換した時に占有するバイト数を取得します。

値を取得するには、読み取り伝文 (AccessModeFlags.Read)、書き込み伝文 (AccessModeFlags.Write) のどちらかを指定します。

Add(UInt16Device)

アクセス情報の末尾に符号無し 16 ビット整数型デバイスを追加します。

UInt16Device については後述を参照してください。

Add(Int16Device)

アクセス情報の末尾に符号付き 16 ビット整数型デバイスを追加します。

Int16Device については後述を参照してください。

Add(UInt32Device)

アクセス情報の末尾に符号無し 32 ビット整数型デバイスを追加します。

UInt32Device については後述を参照してください。

Add(Int32Device)

アクセス情報の末尾に符号付き 32 ビット整数型デバイスを追加します。

Int32Device については後述を参照してください。

Add(SingleFloatDevice)

アクセス情報の末尾に単精度浮動小数点型デバイスを追加します。

SingleFloatDevice については後述を参照してください。

例

次の例では、ランダムアクセス対象として、下記デバイスを登録します。

・D1000 = 1234 (符号無し 16 ビット整数)

・D1002 = 2345 (符号付き 16 ビット整数)

・D1004 = 3456 (符号無し 32 ビット整数)

・D1007 = 4567 (符号付き 32 ビット整数)

・D1012 = 56.78 (単精度浮動小数点)

```
Dim RandomWords As New WordDeviceCollection

With RandomWords
    .Add(New UInt16Device(New WordAddress(1000, WordDeviceFlags.D), 1234US))
    .Add(New Int16Device(New WordAddress(1002, WordDeviceFlags.D), 2345S))
    .Add(New UInt32Device(New WordAddress(1004, WordDeviceFlags.D), 3456UI))
    .Add(New Int32Device(New WordAddress(1007, WordDeviceFlags.D), 4567I))
    .Add(New SingleFloatDevice(New WordAddress(1012, WordDeviceFlags.D), 56.78F)
End With
```

RandomWordReader.Read メソッド、および RandomWordWriter.Write メソッドは、WordDeviceCollection が保持するデータ順序、すなわち、Add メソッドの実行順序でアクセスします。

BitDeviceCollection クラス

BitDeviceCollection クラスは、ビットデバイスにランダム書き込みをする際の対象デバイス情報を保持します。

コンストラクター

BitDeviceCollection()

書式

```
Public Sub New()
```

解説

クラスの新しいインスタンスを初期化します。

プロパティ

Items As List(Of BitDevice)

内部で保持するデバイス情報を取得します。

メソッド

DataLength() As Integer

内部で保持するデータを伝文形式に変換した時に占有するバイト数を取得します。

Add(BitDevice As BitDevice)

アクセス情報の末尾にビット情報を追加します。

BitDevice については後述を参照してください。

例

次の例では、ランダムアクセス対象として、下記デバイスを登録します。

- M1000 = True
- M1003 = Flase
- Y3 = True
- Y1A = False
- Y1C = True
- Y1D = False

```
1  Dim RandomBits As New BitDeviceCollection
2
3  With RandomBits
4     .Add(New BitDevice(New BitAddress("M1000"), True))
5     .Add(New BitDevice(New BitAddress(1003, BitDeviceFlags.M), False))
6     .Add(New BitDevice(New BitAddress("Y3"), True))
7     .Add(New BitDevice(New BitAddress(26, BitDeviceFlags.Y), False))
8     .Add(New BitDevice(New BitAddress(&H1C, BitDeviceFlags.Y), True))
9     .Add(New BitDevice(New BitAddress(1, &HD, BitDeviceFlags.Y), False))
10 End With
```

UInt16Device クラス
Int16Device クラス
UInt32Device クラス
Int32Device クラス
SingleFloatDevice クラス

WordDeviceCollection クラスへアクセス対象を指定するために使用されます。

> 注記 (*)
> ここでは UInt16Device クラスについてのみ説明します。
> コンストラクター、メソッドの機能は Int16Device、UInt32Device、Int32Device、SingleFloatDevice においても同様です。

継承

WordDeviceDefinition → UInt16Device

WordDeviceDefinition → Int16Device

WordDeviceDefinition → UInt32Device

WordDeviceDefinition → Int32Device

WordDeviceDefinition → SingleFloatDevice

コンストラクター

UInt16Device(WordAddress, UInt16)

書式

```
Public Sub New(Address As WordAddress, InitialValue As UInt16)
```

パラメーター

Address	対象デバイスを指示する WordAddress オブジェクトを指定します。
InitialValue	書き込み用途で使用する場合、書き込む値を指定します。読み込み用途ではこの値は無視されます。

解説

クラスの新しいインスタンスを初期化します。

プロパティ

Address As WordAddress

対象デバイスのアドレスを指示する WordAddress オブジェクトを取得します。

DataSize As DataSizeFlags

対象デバイスのサイズを指示する DataSizeFlags 列挙体を取得します。

DataLength As Integer

対象デバイスのバイト数を取得します。

Value As Object

対象デバイスの値を取得または設定します。

取得する場合、適切な型へ変換する必要があります。

メソッド

DecodeBytes(Source As List(Of Byte))

PLC レジスタを表すバイト配列をクラスに整合した値に変換し、Value プロパティにセットします。

変換に成功した場合、変換に使用した項目は引数に与えたリストから消去されます。

GetBytes(AccessMode As AccessModeFlags) As Byte()

Value プロパティの値を伝文形式に変換したバイト配列を取得します。

値を取得するには、読み取り伝文 (AccessModeFlags.Read)、書き込み伝文 (AccessModeFlags.Write) のどちらかを指定します。

例

次の例では、D1000 に初期値 1234 を指定してオブジェクトを初期化します。

```
Dim D1000 As UInt16Device = New UInt16Device(New WordAddress("D1000"), 1234)
```

BitDevice クラス

BitDeviceCollection クラスへアクセス対象を指定するために使用されます。

継承

BitDeviceDefinition → BitDevice

コンストラクター

BitDevice(BitAddress, Boolean)

書式

```
Public Sub New(Address As BitAddress, InitialValue As Boolean)
```

パラメーター

Address	対象デバイスを指示する BitAddress オブジェクトを指定します。
InitialValue	書き込み用途で使用する場合、書き込む値を指定します。読み込み用途ではこの値は無視されます。

解説

クラスの新しいインスタンスを初期化します。

プロパティ

Address As WordAddress

対象デバイスのアドレスを指示する BitAddress オブジェクトを取得します。

Value As Boolean

対象デバイスの値を取得または設定します。

メソッド

GetBytes() As Byte()

Value プロパティの値を伝文形式に変換したバイト配列を取得します。

例

次の例では、M100 に初期値 True を指定してオブジェクトを初期化します。

```
Dim M100 As BitDevice = New BitDevice(New BitAddress("M100"), True)
```

DeviceConverter クラス

DeviceConverter クラスは PLC データと PC データの相互変換機能を提供します。

(変換機能 (XToYArray, XArrayToX メソッド) については解説を省略します)

共有メソッド

SingleWordToBits(UInt16) As Boolean()

指定された符号なし 16 ビット整数をビット (Boolean) 配列に変換して取得します。

下記の例では次の Boolean 配列を得ます。

Bits(0) = True
Bits(1) = False
Bits(2) = True
Bits(3) = True
Bits(4) = False
Bits(5) = False
Bits(6) = True
Bits(7) = True

```
Dim Bits As Boolean() = SingleWordToBits(205)
```

DoubleWordToBits(UInt32) As Boolean()

指定された符号なし 32 ビット整数をビット (Boolean) 配列に変換して取得します。

下記の例では次の Boolean 配列を得ます。

Bits(0) = True
Bits(1) = False
Bits(2) = True
Bits(3) = True
Bits(4) = False
Bits(5) = False
Bits(6) = True
Bits(7) = True
Bits(8) = False
Bits(9) = True
Bits(10) = False
Bits(11) = True
Bits(12) = True
Bits(13) = True
Bits(14) = False
Bits(15) = True

```
Dim Bits As Boolean() = SingleWordToBits(47821)
```

BitsToSingleWord(Boolean()) As UInt16

指定されたビット (Boolean) 配列を符号無し 16 ビット整数に変換して取得します。

配列数が指定された配列サイズを超える場合、余剰分は無視されます。

配列数が指定された配列サイズに満たない場合、不足分は False として埋められ変換されます。

下記の例で、Value は 181 を得ます。

```
Dim Value As UInt16 = BitsToSingleWord({Ture, False, True, False, True, True, False, True})
```

BitsToDoubleWord(Boolean()) As UInt32

指定されたビット (Boolean) 配列を符号無し 32 ビット整数に変換して取得します。

配列数が指定された配列サイズを超える場合、余剰分は無視されます。

配列数が指定された配列サイズに満たない場合、不足分は False として埋められ変換されます。

下記の例で、Value は 39861 を得ます。

```
Dim Value As UInt32 = BitsToDoubleWord({
       True, False, True, False, True, True, False, True,
       True, True, False, True, True, False, False, True})
```

GetWordDeviceCode(String) As WordDeviceFlags

指定されたデバイスアドレス表記（例 "D1000"）から、ワードデバイス種別を取得します。

認識されるデバイス種別は、SD, D, W, TN, RTN, CN です。

引数がワードデバイスとして認識できない場合、戻り値は 0 となります。

下記の例ではそれぞれ次の結果を得ます。

・DeviceCode1 = WordDeviceFlags.D

・DeviceCode2 = 0

```
Dim DeviceCode1 As WordDeviceFlags = GetWordDeviceCode("D1000")
Dim DeviceCode2 As WordDeviceFlags = GetWordDeviceCode("M100)
```

GetBitDeviceCode(String) As BitDeviceFlags

指定されたデバイスアドレス表記（例 "M100"）から、ビットデバイス種別を取得します。

認識されるデバイス種別は、SM, X, Y, M, L, F, V, B, TS, TC, RTS, RTC, CS, CC です。

引数がビットデバイスとして認識できない場合、戻り値は 0 となります。

下記の例ではそれぞれ次の結果を得ます。

・DeviceCode1 = 0

・DeviceCode2 = BitDeviceFlags.M

```
Dim DeviceCode1 As BitDeviceFlags = GetBitDeviceCode("D1000")
Dim DeviceCode2 As BitDeviceFlags = GetBitDeviceCode("M100)
```

GetDeviceAddress(String) As Integer

指定されたデバイスアドレス表記（例 "D1000", "M100", "X2C"）から、オフセット値を取得します。

このメソッドは下記デバイスを認識します。

ワードデバイス: SD, D, W, TN, RTN, CN

ビットデバイス: SM, X, Y, M, L, F, V, B, TS, TC, RTS, RTC, CS, CC

16 進数指定のデバイス（W, X, Y, B）は 16 進数表記の文字列を指定してください。

下記の例ではそれぞれ次の結果を得ます

・D_Offset = 1000

・M_Offset = 100

・X_Offset = 44 (&H2C)

```
Dim D_Offset As Integer = GetDeviceAddress("D1000")
Dim M_Offset As Integer = GetDeviceAddress("M100")
Dim X_Offset As Integer = GetDeviceAddress("X2C")
```

WordOperation クラス

WordOperation クラスはワードデータのビット操作機能を提供します。

共有メソッド

BitTest(UInt16, UShort) As Boolean

BitTest(UInt32, UShort) As Boolean

元データをビット配列に変換し、指定されたビット位置の状態を取得します。

PLC ラダー命令の TESTP、DTESTP に類似します。

BitSet(UInt16, UShort) As UInt16

BitSet(UInt32, UShort) As UInt32

元データをビット配列に変換し、指定された位置のビットを ON とします。またその結果を取得します。

PLC ラダー命令の BSETP、および同命令をダブルワード型に発展させたものに類似します。

BitReset(UInt16, UShort) As UInt16

BitReset(UInt32, UShort) As UInt32

元データをビット配列に変換し、指定された位置のビットを OFF とします。またその結果を取得します。

PLC ラダー命令の BRSTP、および同命令をダブルワード型に発展させたものに類似します。

BitInvert(UInt16, UShort) As UInt16

BitInvert(UInt32, UShort) As UInt32

元データをビット配列に変換し、指定された位置のビットを反転させます。またその結果を取得します。

例

下記の例ではそれぞれ次の値を得ます。

BitTest16 = False	BitTest32 = True
BitSet16 = 189	BitSet32 = 40885
BitReset16 = 165	BitReset32 = 37813
BitInvert16 = 149	BitInvert32 = 35765

```
Dim Value16 As UInt16 = 181US '(bit8< 1011 0101 >bit0)
Dim Value32 As UInt32 = 39861UI '(bit15< 1001 1011 1011 0101 >bit0)

Dim BitTest16 As Boolean = WordOperation.BitTest(Value16, 3)
Dim BitSet16 As UInt16 = WordOperation.BitSet(Value16, 3)
Dim BitReset16 As UInt16 = WordOperation.BitReset(Value16, 4)
Dim BitInvert16 As UInt16 = WordOperation.BitInvert(Value16, 5)

Dim BitTest32 As Boolean = WordOperation.BitTest(Value32, 7)
Dim BitSet32 As UInt32 = WordOperation.BitSet(Value32, 10)
Dim BitReset32 As UInt32 = WordOperation.BitReset(Value32, 11)
Dim BitInvert32 As UInt32 = WordOperation.BitInvert(Value32, 12)
```

WordReader クラス

WordReader クラスは PLC の連続するワードデバイスを、デバイス点数を指定して一括で読み出す機能を提供します。

対応する MC プロトコル コマンド

MELSEC-Q/L シリーズ　0401 0000

MELSEC-iQ-R シリーズ　0401 0002

継承

PLCProtocol → WordReader

コンストラクター

WordReader(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

共有メソッド

QuickRead(MelsecCPU, WordAddress, NumericDataTypeFlags) As Object

書式

```
Public Shared Function QuickRead(
     MelsecCPU As MelsecCPU, Address As WordAddress, NumericDataType As NumericDataTypeFlags
     ) As Object
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。
Address	WordAddress 型オブジェクトにより、読み出すデバイス種別、アドレスを指定します。
NumericDataType	読み出すデバイスの数値型を指定します。

戻り値

正常終了	NumericDataType で指定されたデータ型を内包する Object 型。
返信異常	Nothing

解説

指定されたアドレスのワードデータを、指定された数値型データに変換して取得します。

なお、読み出すデータ長は NumericDataType に基づき自動的に決定されます。

正常終了時、戻り値は Object 型になるため、ユーザーコードで適切な型に変換する必要があります。

見かけ上、WordReader のインスタンスを作成する必要が無いので、ユーザーコードの記述が簡潔になります。

ただし、QuickRead メソッド内部で WordReader の作成、解放を行っているため、多用するとパフォーマンスに影響が出る可能性があります。

QuickRead(MelsecCPU, WordAddress, UShort, System.Text.Encoding)

書式

```
Public Shared Function QuickRead(
     MelsecCPU As MelsecCPU, Address As WordAddress, FormatWords As UShort, Encoding As Encoding
     ) As String
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。
WordAddress	WordAddress 型オブジェクトにより、読み出すデバイス種別、先頭アドレスを指定します。
FormatWords	読み出すデバイス点数をワード単位で指定します。
Encoding	テキスト変換に使用するエンコード形式を指定します。

戻り値

正常終了	String 型
返信異常	Nothing

解説

指定された先頭アドレスから、指定された数のワードデータを文字列型データに変換して取得します。
FormatWords の値は読み出すワードデバイスの点数であり、文字数ではないので注意してください。
見かけ上、WordReader のインスタンスを作成する必要が無いので、ユーザーコードの記述が簡潔になります。
ただし、QuickRead メソッド内部で WordReader の作成、解放を行っているため、多用するとパフォーマンスに影響が出る可能性があります。

メソッド

Read(WordAddress, Integer) As Byte()

書式

```
Public Function Read(StartAddress As WordAddress, RequestWordCount As Integer) as Byte()
```

パラメーター

StartAddress	WordAddress 型オブジェクトにより、読み出すデバイス種別、先頭アドレスを指定します。
RequestWordCount	読み出すデバイス点数をワード単位で指定します。

戻り値

正常終了	要求されたデータのバイト配列。
返信異常	Nothing

解説

指定された先頭アドレスから、指定された数のワードデータを同期処理で PLC から取得します。
Read メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

例

次の例では、ワードレジスタ D1000 を先頭に連続した 13 Word のデータを読み出し、データの型変換を行うまでを紹介します。
ここでは該当レジスタに下記データが含まれるものと仮定します。

・D1000 = 1234 (符号無し 16 ビット整数)
・D1001 = 2345 (符号付き 16 ビット整数)
・D1002 = 3456 (符号無し 32 ビット整数)
・D1004 = 4567 (符号付き 32 ビット整数)
・D1006 - D1009 = "MELSEC" (文字列 4 Word 占有 ※ターミネーター (Null) 含む)
・D1010 - D1013 = "SYSMAC" (文字列 4 Word 占有 ※ターミネーター (Null) 含む)

```
Private Sub ReadWords()
   Dim StartAddress As New WordAddress(1000, WordDeviceFlags.D)
   Dim Response As Byte()
   Dim DataBuilder As DataBuilder

   '1) D1000 を先頭に 13 Word 読み出し、DataBuilder のコンストラクタへ与えます
   Using WordReader As New WordReader(_QCPU)
       Response = WordReader.Read(StartAddress, 13)
       DataBuilder = New DataBuilder(Response)
   End Using

   Dim UInt16Response As UInt16
   Dim Int16Response As Int16
   Dim UInt32Response As UInt32
   Dim Int32Response As Int32
   Dim StringResponse1 As String
   Dim StringResponse2 As String

   '2) 読み出したデータを DataBuilder を通じて型変換・取得します
   UInt16Response = DataBuilder.GetUInt16
   Int16Response = DataBuilder.GetInt16
   UInt32Response = DataBuilder.GetUInt32
   Int32Response = DataBuilder.GetInt32
   StringResponse1 = DataBuilder.GetString(4, PLCEncoding.ShiftJIS)
   StringResponse2 = DataBuilder.GetString(3, PLCEncoding.ShiftJIS)
End Sub
```

Read メソッドの戻り値を使用して DataBuilder を初期化している点に注目してください。
こうして作成された DataBuilder は 内部に PLC レジスタのコピーとなるバイト配列が作成されます。
その後、Get*** メソッドが呼び出されるたびに、このバイト配列の先頭から適切なバイト数を使用して要求された型に変換します。

- Get*** メソッドを呼び出す順序が PLC のレジスタの順序と一致するように注意してください。
- Get*** メソッドを使用して値を取得すると、変換に使用したバイト配列要素は削除されます。すなわち、Get*** メソッドを呼び出すたびに、DetaBuilder 内部の配列数は減少し、最終的に配列要素数は 0 になります。

GetString メソッドの FormatWords 引数に、ターミネーター 1 Word を考慮する必要はありません。
25 行目の GetString の記述では 3 Word 分の変換を要求しています。この場合、変換範囲 (D1010 - D1012) にターミネーターは含まれませんが、正常に "SYSMAC" (3 Word) を得られます。

次の例では、Read メソッドの戻り値 (Byte()) を DeviceConverter クラスの共有メソッドで解釈しています。
DataBuilder 経由で型変換を行う場合と異なり、Read メソッドの戻り値を柔軟に取り扱うことできます。

```
Private Sub ReadWords()
   Dim StartAddress As New WordAddress(1000, WordDeviceFlags.D)
   Dim Response As List(Of Byte)

   '1) D1000 を先頭に 13 Word (=26 bytes) 読み出します
   Using WordReader As New WordReader(_QCPU)
       Response = WordReader.Read(StartAddress, 13).ToList
   End Using

   Dim UInt16Response As UInt16
   Dim Int16Response As Int16
   Dim UInt32Response As UInt32
   Dim Int32Response As Int32
   Dim StringResponse1a As String
   Dim StringResponse2a As String
   Dim StringResponse1b As String
```

```
    Dim StringResponse2b As String

    '2) 読み出したデータを DeviceConverter クラスの共有メソッドで型変換・取得します
    UInt16Response = DeviceConverter.ByteToUInt16({Response.Item(0), Response.Item (1)})
    Int16Response = DeviceConverter.ByteToInt16({Response.Item (2), Response.Item (3)})
    UInt32Response = DeviceConverter.ByteToUInt16(Response.GetRange(4, 4).ToArray)
    Int32Response = DeviceConverter.ByteToInt32(Response.GetRange(8, 4).ToArray)
    StringResponse1a = PLCConverter.ByteToString(Response.GetRange(12, 8).ToArray,
        PLCEncoding.ShiftJIS)
    StringResponse2a = DeviceConverter.ByteToString(Response.GetRange(20, 6).ToArray,
        PLCEncoding.ShiftJIS)

    'ByteToString に渡す Byte 配列に終端データが含まれていなくても変換可能です
    StringResponse1b = DeviceConverter.ByteToString(Response.GetRange(12, 6).ToArray,
        PLCEncoding.ShiftJIS)
    StringResponse2b = DeviceConverter.ByteToString(Response.GetRange(20, 4).ToArray,
        PLCEncoding.ShiftJIS)
End Sub
```

次の例では、WordReader クラスの共有メソッド (QuickRead) を使用して記述しています。
見かけ上、WordReader のインスタンスを作成する必要がないので、簡潔な記述で済みます。
ただし、QuickRead メソッドの中で WordReader の作成、廃棄を行っているので、多用するとパフォーマンスに影響が出る可能性があります。

```
Private Sub ReadWords()
    Dim UInt16Response As UInt16
    Dim Int16Response As Int16
    Dim UInt32Response As UInt32
    Dim Int32Response As Int32
    Dim StringResponse1 As String
    Dim StringResponse2 As String

    UInt16Response = CUShort(WordReader.QuickRead(
        _QCPU, New WordAddress("D1000"), NumericDataTypeFlags.UnsignedSingleInt))
    Int16Response = CShort(WordReader.QuickRead(
        _QCPU, New WordAddress("D1001"), NumericDataTypeFlags.SignedSingleInt))
    UInt32Response = CUInt(WordReader.QuickRead(
        _QCPU, New WordAddress("D1002"), NumericDataTypeFlags.UnsignedDoubleInt))
    Int32Response = CInt(WordReader.QuickRead(
        _QCPU, New WordAddress("D1004"), NumericDataTypeFlags.SignedSingleInt))
    StringResponse1 = WordReader.QuickRead(_QCPU, New WordAddress("D1006"), 4, PLCEncoding.ShiftJIS)
    StringResponse2 = WordReader.QuickRead(_QCPU, New WordAddress("D1010"), 3, PLCEncoding.ShiftJIS)
End Sub
```

WordWriter クラス

WordWriter クラスは PLC の連続するワードデバイスへ、データを一括で書き込む機能を提供します。

対応する MC プロトコル コマンド

MELSEC-Q/L シリーズ	1401 0000
MELSEC-iQ-R シリーズ	1401 0002

継承

PLCProtocol → WordWriter

コンストラクター

WordWriter(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

共有メソッド

QuickWrite(MelsecCPU, WordAddress, UInt16) As List(Of Integer)

QuickWrite(MelsecCPU, WordAddress, Int16) As List(Of Integer)

QuickWrite(MelsecCPU, WordAddress, UInt32) As List(Of Integer)

QuickWrite(MelsecCPU, WordAddress, Int32) As List(Of Integer)

QuickWrite(MelsecCPU, WordAddress, Single) As List(Of Integer)

QuickWrite(MelsecCPU, WordAddress, Double) As List(Of Integer)

書式

```
Public Shared Function QuickWrite(
     MelsecCPU As MelsecCPU, Address As WordAddress, Value As <Type>) As Object
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。
Address	WordAddress 型オブジェクトにより、書き込むデバイス種別、アドレスを指定します。
Value [As Type]	書き込む値を指定します。

戻り値

正常終了	書き込み完了を表す PLC からの応答値。
返信異常	Nothing

解説

指定された数値型データを、指定されたワードアドレスへ書き込みます。

見かけ上、WordWriter のインスタンスを作成する必要が無いので、ユーザーコードの記述が簡潔になります。

ただし、QuickWrite メソッド内部で WordWriter の作成、解放を行っているため、多用するとパフォーマンスに影響が出る可能性があります。

QuickWrite(MelsecCPU, WordAddress, String, UShort, System.Text.Encoding)

書式

```
Public Shared Function QuickWrite(
     MelsecCPU As MelsecCPU, Address As WordAddress, Value As String, FormatWords As UShort,
     Encoding As Encoding) As List(Of Integer)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。
Address	WordAddress 型オブジェクトにより、書き込むデバイス種別、アドレスを指定します。
Value	書き込む文字列を指定します。
FormatWords	文字列を書き込むために確保するワード数。
Encoding	文字列を書き込むために使用するエンコード形式。

戻り値

正常終了	書き込み完了を表す PLC からの応答値。
返信異常	Nothing

解説

指定された文字列データを、指定されたアドレスを先頭に書き込みます。

見かけ上、WordWriter のインスタンスを作成する必要が無いので、ユーザーコードの記述が簡潔になります。

ただし、QuickWrite メソッド内部で WordWriter の作成、解放を行っているため、多用するとパフォーマンスに影響が出る可能性があります。

メソッド

Write(WordAddress, DataBuilder) As List(Of Integer)

書式

```
Public Function Write(StartAddress As WordAddress, WriteData As DataBuilder) As List(Of Integer)
```

パラメーター

StartAddress	デバイス種別、書き込み先頭アドレスを指定する WordAddress 型オブジェクトを指定します。
WriteData	書き込むデータを保持している DataBuilder 型オブジェクトを指定します。

戻り値

正常終了	書き込み完了を表す PLC からの応答値。
返信異常	Nothing

解説

指定されたデバイス種別・アドレスを先頭に、指定されたバイト配列のデータを同期処理で PLC へ書き込みます。

Write メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

例

次の例では、ワードレジスタ D1000 を先頭に下記数値・文字列を書き込みます。

・D1000 = 1234 (符号無し 16 ビット整数)

・(D1001 =) 2345 (符号付き 16 ビット整数)

・(D1002 =) 3456 (符号無し 32 ビット整数)

・(D1004 =) 4567 (符号付き 32 ビット整数)

・(D1006 - D1009 =) "MELSEC" (文字列 4 Word 占有)

・(D1010 - D1012 =) "SYSMAC" (文字列 3 Word 占有)

D1001 以降のアドレスを明示的に指定することはできません。直前のデータに密着して配置されることに注意してください。

また、SetString メソッドの FormatWords 引数には、ターミネーター (Null) が 1 Word が含まれることに注意してください。

11 行目の SetString では "SYSMAC" を書き込むために 3 Word を要求していますが、ターミネーターが 1 Word 占拠するため、実際に書き込まれるデータは "SYSM" (2 Word) となります。

```
Private Sub WriteWords()
   Dim StartAddress As New WordAddress(1000, WordDeviceFlags.D)
   Dim DataBuilder As New DataBuilder

   With DataBuilder
       .SetUInt16(1234)
       .SetInt16(2345)
       .SetUInt32(3456)
       .SetInt32(4567)
       .SetString("MELSEC", 4, PLCEncoding.ShiftJIS)
       .SetString("SYSMAC", 3, PLCEncoding.ShiftJIS)
   End With

   Using WordWriter As New WordWriter(_QCPU)
       WordWriter.Write(StartAddress, DataBuilder)
   End Using
End Sub
```

次の例では、上記の例を異なった方法で記述していますが、同じ結果を得ます。

・WordAddress オブジェクトの引数をアドレス文字列で指定しています (2 行目)。

・DataBuilder の SetAnyBytes メソッドを使用して、ユーザー定義の Byte 配列を取り込んでいます (6, 8, 10 行目)。

```
Private Sub WriteWords()
   Dim StartAddress As New WordAddress("D1000")
   Dim DataBuilder As New DataBuilder

   With DataBuilder
       .SetAnyBytes(DeviceConverter.UInt16ToByteArray(1234))
       .SetInt16(2345)
       .SetAnyBytes(DeviceConverter.UInt32ToByteArray(3456))
       .SetInt32(4567)
       .SetAnyBytes(DeviceConverter.StringToByteArray("MELSEC", 4, PLCEncoding.ShiftJIS))
       .SetString("SYSMAC", 3, PLCEncoding.ShiftJIS)
   End With

   Using WordWriter As New WordWriter(_QCPU)
       WordWriter.Write(StartAddress, DataBuilder)
   End Using
End Sub
```

次の例では、WordWriter クラスの共有メソッドを使用して記述しています。

```
Private Sub WriteWords()
   WordWriter.QuickWrite(_QCPU, New WordAddress("D1000"), 1234US)
   WordWriter.QuickWrite(_QCPU, New WordAddress("D1001"), 2345S)
   WordWriter.QuickWrite(_QCPU, New WordAddress("D1002"), 3456UI)
   WordWriter.QuickWrite(_QCPU, New WordAddress("D1004"), 4567I)
   WordWriter.QuickWrite(_QCPU, New WordAddress("D1006"), "MELSEC", 4, PLCEncoding.ShiftJIS)
   WordWriter.QuickWrite(_QCPU, New WordAddress("D1010"), "SYSMAC", 3, PLCEncoding.ShiftJIS)
End Sub
```

BitReader クラス

BitReader クラスは PLC の連続するビットデバイスを、ビット点数を指定して一括で読み出す機能を提供します。

対応する MC プロトコル コマンド

MELSEC-Q/L シリーズ	0401 0001
MELSEC-iQ-R シリーズ	0401 0003

継承

PLCProtocol → BitReader

コンストラクター

BitReader(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

共有メソッド

QuickRead(MelsecCPU, BitAddress) As Boolean

書式

```
Public Shared Function QuickRead(MelsecCPU As MelsecCPU, Address As BitAddress) As Boolean
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。
Address	BitAddress 型オブジェクトにより、読み出すデバイス種別、アドレスを指定します。

戻り値

正常終了	Boolean 型
返信異常	Nothing

解説

指定されたビットアドレスの状態を Bool 値 で取得します。

見かけ上、BitReader のインスタンスを作成する必要が無いので、ユーザーコードの記述が簡潔になります。

ただし、QuickRead メソッド内部で BitReader の作成、解放を行っているため、多用するとパフォーマンスに影響が出る可能性があります。

メソッド

Read(BitAddress, Integer) As List(Of Boolean)

書式

```
Public Function Read(StartAddress As BitAddress, RequestBitCount As Integer) As List(Of Boolean)
```

パラメーター

StartAddress	デバイス種別、読み込み先頭アドレスを指定する BitAddress 型オブジェクトを指定します。
RequestBitCount	読み出すビット数を指定します。

解説

指定されたデバイス種別・アドレスを先頭に、指定された数のビットデータを同期処理で PLC から取得します。

PLC からの返信が不正だった場合、Nothing を返します。

戻り値

正常終了	要求されたデータの Boolean 型リスト。
返信異常	Nothing

解説

指定された先頭アドレスから、指定された数のビットデータを同期処理で PLC から取得します。

Read メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

例

次の例では、ビットアドレス M1000 を先頭に 連続した 8 Bit の状態を読み出します。

Read メソッドの戻り値 (List(Of Boolean)) の Item 引数に StartAddress に対するオフセットを与えてビットの状態を得ています。

```
Private Sub ReadBits()
   Dim StartOffset As Integer = 1000
   Dim StartAddress As New BitAddress(StartOffset, BitDeviceFlags.M)
   Dim Response As List(Of Boolean)

   Using BitReader As New BitReader(_QCPU)
       Response = BitReader.Read(StartAddress, 8)
   End Using

   Dim M1000 As Boolean = Response.Item(StartOffset + 0)
   Dim M1001 As Boolean = Response.Item(StartOffset + 1)
   Dim M1002 As Boolean = Response.Item(StartOffset + 2)
   Dim M1003 As Boolean = Response.Item(StartOffset + 3)
   Dim M1004 As Boolean = Response.Item(StartOffset + 4)
   Dim M1005 As Boolean = Response.Item(StartOffset + 5)
   Dim M1006 As Boolean = Response.Item(StartOffset + 6)
   Dim M1007 As Boolean = Response.Item(StartOffset + 7)
End Sub
```

次の例では、BitReader クラスの共有メソッドを使用して記述しています。

```
Private Sub ReadBits()
   Dim M1000 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1000"))
   Dim M1001 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1001"))
   Dim M1002 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1002"))
   Dim M1003 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1003"))
   Dim M1004 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1004"))
   Dim M1005 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1005"))
   Dim M1006 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1006"))
   Dim M1007 As Boolean = BitReader.QuickRead(_QCPU, New BitAddress("M1007"))
End Sub
```

BitWriter クラス

BitWriter クラスは PLC の連続するビットデバイスへ、ビット状態を一括で書き込む機能を提供します。

対応する MC プロトコル コマンド

MELSEC-Q/L シリーズ　1401 0001

MELSEC-iQ-R シリーズ　1401 0003

継承

PLCProtocol → BitWriter

コンストラクター

BitWriter(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

共有メソッド

QuickRead(MelsecCPU, BitAddress, Boolean) As List(Of Integer)

書式

```
Public Shared Function QuickWrite(
   MelsecCPU As MelsecCPU, Address As BitAddress, Status As Boolean) As List(Of Integer)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。
Address	BitAddress 型オブジェクトにより、読み出すデバイス種別、アドレスを指定します。
Status	書き込む状態を指定します。

戻り値

正常終了	書き込み完了を表す PLC からの応答値。
返信異常	Nothing

解説

指定されたビットアドレスの状態を、指定されたビットアドレスへ書き込みます。

見かけ上、BitWriter のインスタンスを作成する必要が無いので、ユーザーコードの記述が簡潔になります。

ただし、QuickWrite メソッド内部で BitWriter の作成、解放を行っているため、多用するとパフォーマンスに影響が出る可能性があります。

メソッド

Write(BitAddress, Boolean()) As List(Of Integer)

書式

```
Public Function Write(StartAddress As BitAddress, WriteData As Boolean()) As List(Of Integer)
```

パラメーター

StartAddress	デバイス種別、書き込み先頭アドレスを指定する BitAddress 型オブジェクトを指定します。
WriteData	書き込むビット状態を保持する配列 (Boolean()) を指定します。

戻り値

正常終了	書き込み完了を表す PLC からの応答値。
返信異常	Nothing

解説

指定されたデバイス種別・アドレスを先頭に、指定された Boolean 配列の状態値を同期処理で PLC へ書き込みます。

Write メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

例

次の例では、ビットアドレス M1000 を先頭に 連続した 8 Bit に対して下記状態を書き込みます。

- M1000 = True
- M1001 = False
- M1002 = True
- M1003 = True
- M1004 = False
- M1005 = True
- M1006 = False
- M1007 = True

```
Private Sub WriteBits()
   Dim StartAddress As New BitAddress(1000, BitDeviceFlags.M)
   Dim WriteData As New List(Of Boolean)

   With WriteData
       .Add(True) 'M1000
       .Add(False) 'M1001
       .Add(True) 'M1002
       .Add(True) 'M1003
       .Add(False) 'M1004
       .Add(True) 'M1005
       .Add(False) 'M1006
       .Add(True) 'M1007
   End With

   Using BitWriter As New BitWriter(_QCPU)
       BitWriter.Write(StartAddress, WriteData.ToArray)
   End Using
End Sub
```

次の例では、Boolean 配列の初期化子を使用して書き込み状態を定義しています。

```
Private Sub WriteBits()
   Dim StartAddress As New BitAddress("M1000")
   Dim WriteData As Boolean() = {True, False, True, True, False, True, False, True}

   Using BitWriter As New BitWriter(_QCPU)
       BitWriter.Write(StartAddress, WriteData)
   End Using
End Sub
```

次の例では、BitWriter クラスの共有メソッドを使用して記述しています。

```
Private Sub WriteBits()
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1000"), True)
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1001"), False)
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1002"), True)
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1003"), True)
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1004"), False)
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1005"), True)
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1006"), False)
   BitWriter.QuickWrite(_QCPU, New BitAddress("M1007"), True)
End Sub
```

RandomWordReader クラス

RandomWordReader クラスは PLC の不連続なワードデバイスを読み出す機能を提供します。

対応する MC プロトコル コマンド

MELSEC-Q/L シリーズ	0403 0000 (※モニタ条件指定は行えません)
MELSEC-iQ-R シリーズ	0403 0002 (※モニタ条件指定は行えません)

継承

PLCProtocol → RandomWordReader

コンストラクター

RandomWordReader(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

メソッド

Read(WordDeviceCollection) As Dictionary(Of String, Object)

書式

```
Public Function Read(Devices As WordDeviceCollection) As Dictionary(Of String, Object)
```

パラメーター

Devices	読み出すワードデバイス情報 (アドレス、デバイス種別、データ型) を構築します。

戻り値

正常終了	要求されたデータを格納する Dictionary(Of String, Object) 型。
返信異常	Nothing

解説

WordDeviceCollection 型オブジェクトに格納された不連続デバイスのデータを PLC から同期処理で読み出します。

正常に読み込みが完了した場合には、PLC からの読み出しデータ (Dictionary(Of String, Object)) を返します。

読み出しデータは Key プロパティに String 型のデバイスアドレス (例: "D1000") を指定することで取得できます。

この方法で取得できるデータは、Read メソッド実行時に指定した型を内包する Object 型です。

Read メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

例

次の例では、以下のデータを読み出します。デバイスが不連続である点に注目してください。

・D1000 (符号無し 16 ビット整数)

・D1002 (符号付き 16 ビット整数)

・D1004 (符号無し 32 ビット整数)

・D1007 (符号付き 32 ビット整数)

```
Private Sub RandomWordRead()
   Dim RandomWords As New WordDeviceCollection

   '1) 読み出すデータを定義します
   With RandomWords
       .Add(New UInt16Device(New WordAddress("D1000"), 0))
       .Add(New Int16Device(New WordAddress("D1002"), 0))
       .Add(New UInt32Device(New WordAddress("D1004"), 0))
       .Add(New Int32Device(New WordAddress("D1007"), 0))
   End With

   Dim Response As Dictionary(Of String, Object)
   Dim D1000 As UInt16
   Dim D1002 As Int16
   Dim D1004 As UInt32
   Dim D1007 As Int32

   '2) データを読み出します
   Using RandomWordReader As New RandomWordReader(_QCPU)
       Response = RandomWordReader.Read(RandomWords)
   End Using

   D1000 = CUShort(Response.Item("D1000"))
   D1002 = CShort(Response.Item("D1002"))
   D1004 = CUInt(Response.Item("D1004"))
   D1007 = CInt(Response.Item("D1007"))
End Sub
```

RandomWordWriter クラス

RandomWordWriter クラスは PLC の不連続なワードデバイスへ、データを書き込む機能を提供します。

対応する MC プロトコル コマンド

MELSEC-Q/L シリーズ　1402 0000

MELSEC-iQ-R シリーズ　1402 0002

継承

PLCProtocol → RandomWordWriter

コンストラクター

RandomWordWriter(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

メソッド

Write(WordDeviceCollection) As List(Of Integer)

書式

```
Public Function Write(Devices As WordDeviceCollection) As List(Of Integer)
```

パラメーター

Devices	書き込むワードデバイス情報 (アドレス、デバイス種別、データ型) を構築します。

戻り値

正常終了	書き込み完了を表す PLC からの応答値。
返信異常	Nothing

解説

WordDeviceCollection 型オブジェクトに格納された不連続デバイスのデータを PLC へ同期処理で書き込みます。

Write メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

例

次の例では、以下のデータを書き込みます。デバイスが不連続である点に注目してください。

・D1000 = 1234 (符号無し 16 ビット整数)

・D1002 = 2345 (符号付き 16 ビット整数)

・D1004 = 3456 (符号無し 32 ビット整数)

・D1007 = 4567 (符号付き 32 ビット整数)

また、書き込みデバイスに挟まれているレジスタ (D1001, D1003, D1006) に影響が出ないことにも注目してください。

```
Private Sub RandomWordWrite()
   Dim RandomWords As New WordDeviceCollection

   With RandomWords
       .Add(New UInt16Device(New WordAddress(1000, WordDeviceFlags.D), 1234US))
       .Add(New Int16Device(New WordAddress(1002, WordDeviceFlags.D), 2345S))
       .Add(New UInt32Device(New WordAddress(1004, WordDeviceFlags.D), 3456UI))
       .Add(New Int32Device(New WordAddress(1007, WordDeviceFlags.D), 4567I))
   End With

   Using RandomWordWriter As New RandomWordWriter(_QCPU)
       RandomWordWriter.Write(RandomWords)
   End Using
End Sub
```

RandomBitReader クラス

RandomBitReader クラスは PLC の不連続なビットデバイスを読み出す機能を提供します。

コンストラクター

RandomBitReader(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

メソッド

Read(BitDeviceCollection) As Dictionary(Of String, Boolean)

書式

```
Public Function Read(Devices As BitDeviceCollection) As Dictionary(Of String, Boolean)
```

パラメーター

Devices	読み出すビットデバイス情報(アドレス、デバイス種別、データ型) を構築します。

戻り値

正常終了	要求されたビット状態を格納する Dictionary(Of String, Boolean) 型。
返信異常	Nothing

解説

BitDeviceCollection 型オブジェクトに格納された不連続ビットデバイスの状態を PLC へ同期処理で読み出します。

Read メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

PLC との交信には内部で BitReader クラスを利用していますが、ユーザーコードからは見かけ上、ランダムアクセスしているように振る舞います。

BitReader.Read メソッドの呼び出し回数が最少となるように、Devices 引数で与えられた読み出し対象デバイスは内部で適切に再構成されます。

正常に読み込みが完了した場合は、PLC からの読み出しデータ (Dictionary(Of String, Boolean)) を返します。

デバイスのビット状態は Key プロパティに String 型のビットアドレス (例 "M100") を指定することで取得できます。

例

次の例では、以下の不連続な 8 点のビット状態を読み出します。

- M100
- M102
- M103
- X5
- X9
- XA
- Y12
- Y1C
- Y1D

```
1  Private Sub RandomBitRead()
2     Dim RandomBits As New BitDeviceCollection
3
4     '1) 読み出すデータを定義します
5     With RandomBits
6         .Add(New BitDevice(New BitAddress("M100"), False))
7         .Add(New BitDevice(New BitAddress("M102"), False))
8         .Add(New BitDevice(New BitAddress("M103"), False))
9         .Add(New BitDevice(New BitAddress("X5"), False))
10        .Add(New BitDevice(New BitAddress("X9"), False))
11        .Add(New BitDevice(New BitAddress("XA"), False))
12        .Add(New BitDevice(New BitAddress("Y12"), False))
```

```
        .Add(New BitDevice(New BitAddress("Y1C"), False))
        .Add(New BitDevice(New BitAddress("Y1D"), False))
    End With

    Dim Response As Dictionary(Of String, Boolean)

    '2) データを読み出します
    Using RandomBitReader As New RandomBitReader(_QCPU)
        Response = RandomBitReader.Read(RandomBits)
    End Using

    Dim M100, M102, M103 As Boolean
    Dim X5, X9, XA As Boolean
    Dim Y12, Y1C, Y1D As Boolean

    M100 = Response.Item("M100")
    M102 = Response.Item("M102")
    M103 = Response.Item("M103")
    X5 = Response.Item("X5")
    X9 = Response.Item("X9")
    XA = Response.Item("XA")
    Y12 = Response.Item("Y12")
    Y1C = Response.Item("Y1C")
    Y1D = Response.Item("Y1D")
End Sub
```

このクラスは内部で BitReader.Read メソッドを利用してビット状態を読み出していますが、Read メソッドの呼び出し回数は BitDeviceCollection に登録したビットデバイスの点数と必ずしも一致しません。Read メソッドの呼び出し回数が最少となるように最適化されるためです。
上記の例で発行される Read メソッドは次の通りです。

```
BitReader.Read(New BitAddress("M100"), 1)
BitReader.Read(New BitAddress("M102"), 2)
BitReader.Read(New BitAddress("X5", 1)
BitReader.Read(New BitAddress("X9", 2)
BitReader.Read(New BitAddress("Y12", 1)
BitReader.Read(New BitAddress("Y1C", 2)
```

9 点の読み出しに対して、Read メソッドの呼び出しが 6 回であることに注目してください。
連続しているアドレス (M102 と M103、X9 と XA、Y1C と Y1D) をそれぞれ 2 点分の連続読み出しとして取り扱われるため、Read メソッドの発行が 6 回に抑えられています。
なお、最適化処理の過程でデバイスの並び替えが行われるため、BitDeviceCollection への登録順序に留意する必要はありません。すなわち、6 ～ 14 行目の記述が次のような順序でも、Read メソッドの呼び出し回数や内容は冒頭の例と同じです。

```
    With RandomBits
        .Add(New BitDevice(New BitAddress("Y1C"), False))
        .Add(New BitDevice(New BitAddress("M100"), False))
        .Add(New BitDevice(New BitAddress("M103"), False))
        .Add(New BitDevice(New BitAddress("Y1D"), False))
        .Add(New BitDevice(New BitAddress("X5"), False))
        .Add(New BitDevice(New BitAddress("XA"), False))
        .Add(New BitDevice(New BitAddress("M102"), False))
        .Add(New BitDevice(New BitAddress("Y12"), False))
        .Add(New BitDevice(New BitAddress("X9"), False))
    End With
```

RandomBitWriter クラス

RandomBitWriter クラスは PLC の不連続なビットデバイスへ、状態を書き込む機能を提供します。

対応する MC プロトコル コマンド

MELSEC-Q/L シリーズ　1402 0001

MELSEC-iQ-R シリーズ　1402 0003

継承

PLCProtocol → RandomBitWriter

コンストラクター

RandomBitWriter(MelsecCPU)

書式

```
Public Sub New(MelsecCPU As MelsecCPU)
```

パラメーター

MelsecCPU	対象 PLC への接続情報を保持する MelsecCPU オブジェクトを指定します。

解説

指定された接続設定を基にクラスの新しいインスタンスを初期化します。

メソッド

Write(BitDeviceCollection) As List(Of Integer)

書式

```
Public Function Write(Devices As BitDeviceCollection) As List(Of Integer)
```

パラメーター

Devices	書き込むビットデバイス情報 (アドレス、デバイス種別、状態 (Boolean)) を構築します。

戻り値

正常終了	書き込み完了を表す PLC からの応答値。
返信異常	Nothing

解説

BitDeviceCollection 型オブジェクトに格納された不連続ビットデバイスの状態を PLC へ同期処理で書き込みます。

Write メソッド実行時、コンストラクターに使用した MelsecCPU が PLC に未接続だった場合エラーになります。

例

次の例では、以下のデータを書き込みます。

・M1000 = True
・M1003 = Flase
・Y3 = True
・Y1A = False
・Y1C = True
・Y1D = False

書き込みデータは BitDeviceCollection オブジェクトを使用して構築します。

```
Private Sub RandomBitWrite()
   Dim RandomBits As New BitDeviceCollection

   With RandomBits
       .Add(New BitDevice(New BitAddress("M1000"), True))
       .Add(New BitDevice(New BitAddress(1003, BitDeviceFlags.M), False))
       .Add(New BitDevice(New BitAddress("Y3"), True))
       .Add(New BitDevice(New BitAddress(26, BitDeviceFlags.Y), False))
       .Add(New BitDevice(New BitAddress(&H1C, BitDeviceFlags.Y), True))
       .Add(New BitDevice(New BitAddress(1, &HD, BitDeviceFlags.Y), False))
   End With

   Using RandomBitWriter As New RandomBitWriter(_QCPU)
       RandomBitWriter.Write(RandomBits)
   End Using
End Sub
```

11. 改訂履歴

1.0.0　(初版)

1.1.0　・ランダムアクセスクラス (RandomWordReader, RandomWordWriter, RandomBitWriter) を追加
・ランダムアクセスクラス追加に伴う、データクラスを追加
・コードの整理、見直し (一部 1.0.0 版と互換性なし)
・ドキュメント誤記の訂正、加筆

1.2.0　・RandomBitReader を追加

1.2.1　・通信が遮断されると、Try ブロックで補足できない例外 (破損状態例外) が発生してしまうコードを修正
場所: MelsecCommunication.PLCProtocol.SendMessage
修正後:

```
Protected Function SendMessage(Message As Byte()) As Byte()
(中略)
   _MelsecCPU.CommClient.GetStream.Write(Message, 0, Message.Length)

   PollingStart = Now
   Do Until _MelsecCPU.CommClient.Available > 0
       If PollingStart.AddSeconds(PollingInterval) < Now Then
           Throw New Exception()
       End If
       Application.DoEvents()
   Loop

   ReadBuffer = New Byte(_MelsecCPU.CommClient.Available - 1) {}
   _MelsecCPU.CommClient.GetStream.Read(ReadBuffer, 0, ReadBuffer.Length)
```

修正前:

```
Protected Function SendMessage(Message As Byte()) As Byte()
(中略)
   Try
       _MelsecCPU.CommClient.GetStream.Write(Message, 0, Message.Length)

       PollingStart = Now
       Do Until _MelsecCPU.CommClient.Available > 0
           If PollingStart.AddSeconds(PollingInterval) < Now Then
               Throw New Exception()
           End If

           Application.DoEvents()
       Loop

       ReadBuffer = New Byte(_MelsecCPU.CommClient.Available - 1) {}
       _MelsecCPU.CommClient.GetStream.Read(ReadBuffer, 0, ReadBuffer.Length)

   Catch ex As Exception
       Return Nothing
   End Try
```

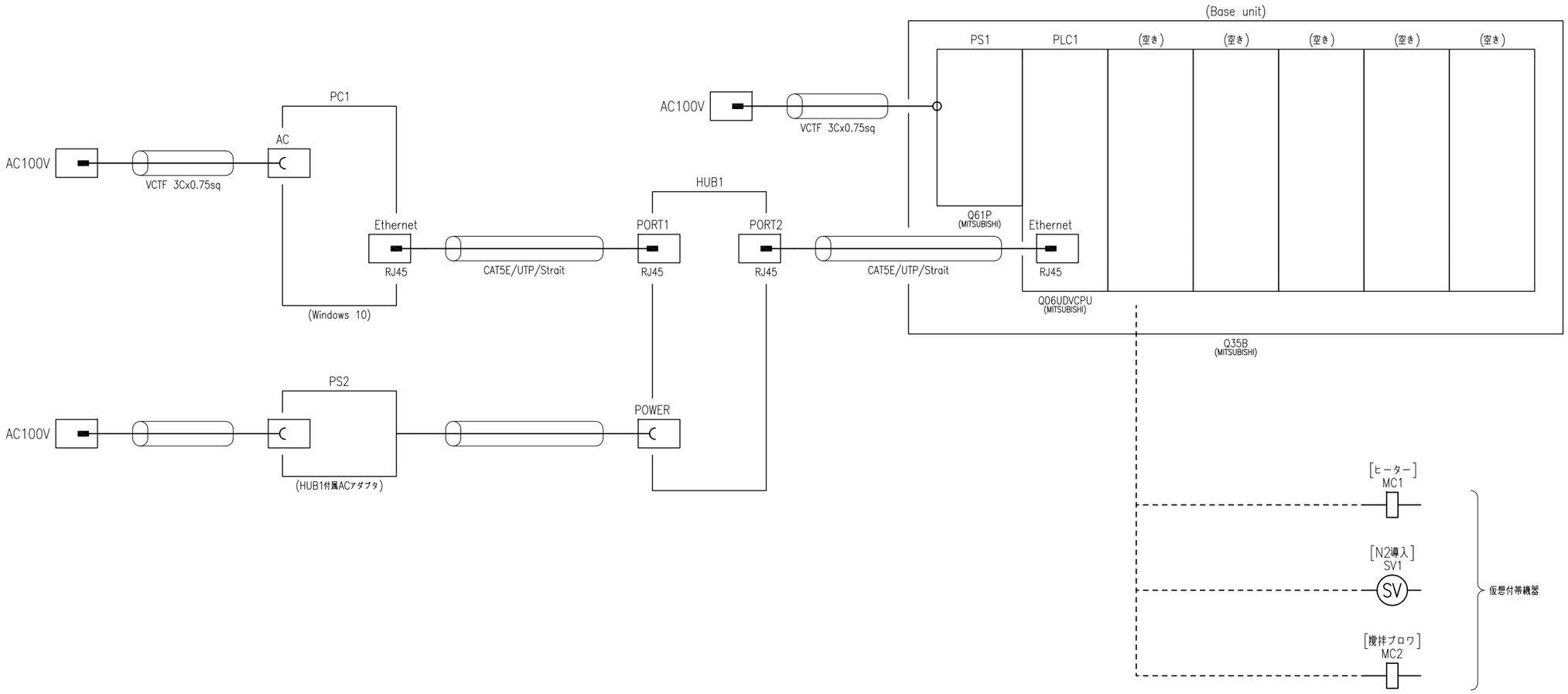
12. 最後に

ご興味、ご関心をお寄せいただきましてありがとうございます。
役立つ内容だったでしょうか?

もしかしたら FX ともお話しできるかもしれません。
KV は MC モードにすればお話できそうです。
CJ と仲良くなれるかはなんとも言えません。

この先は基本、ご自身で何とかしてください。
FA 業なりにしんどいこと多いので、苦情や問合せはもう聞きたくないです...
喜ばしい内容に限り下記にご連絡ください。
fameansfastautomation@gmail.com

FA は努力とひらめき、ひと添えの遊び心



承認	検図	設計	製図

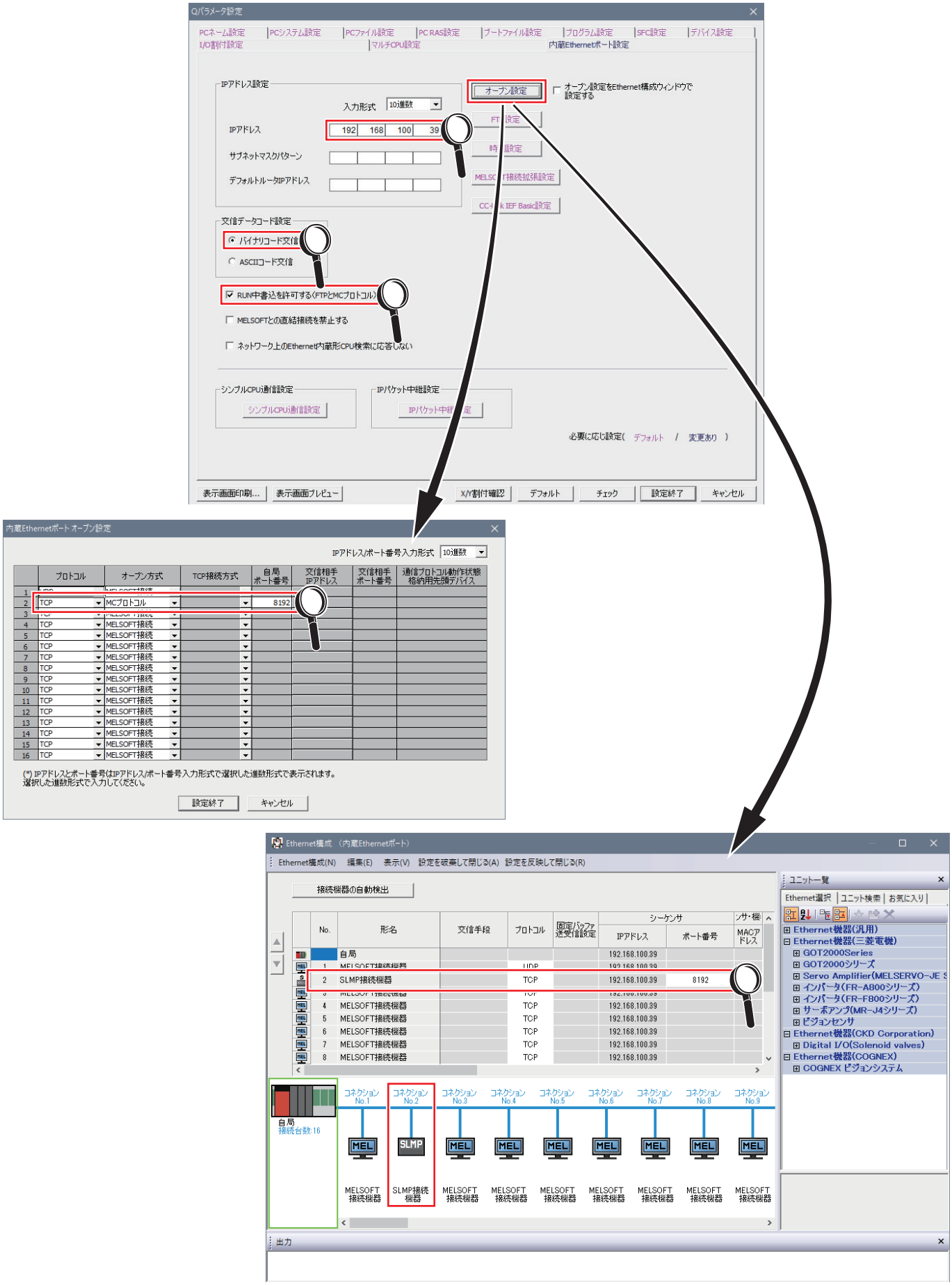
名称 通信検証回路

MelsecCommunicationライブラリ

図番

改番 K0 | シート 1/1 | A3

GX Works 2 での設定例



GX Works 3 での設定例