シンプル通信　For　オムロン

上位リンク通信用 ActiveX コントロール

青山技術

2012 年 11 月 12 日

作成者: 青山

シンプル通信　For　オムロン

上位リンク通信用 ActiveX コントロール

オムロンのシーケンサとパソコンとの Rs-232C 上位リンク通信（FINS コマンド、C コマンド）を行う ActiveX コントロールです。

生産数の表示や、設定数の変更などに利用できます。

VB6.0、インターネットエクスプローラ、Visual Studio などで利用できます。

サポート OS:Windows Xp/7

サポート言語：Visual Basic 6.0(SP5)、VS2010 C#、IE9、エクセル 2010 他

上記以外の OS、言語においても、動作するものがあります。

また、64 ビット OS での開発の場合、本コントロールを用いたアプリケーションのプラットフォームを「Any CPU」から「x86」に変更する必要があることがあります。

試用期間と、登録について

本製品には、シェアウェア版と、プロ版があります。

シェアウェア版は、ポートを開く（Open メソッドの実行）時に、ダイアログが表示され、また、試用期間は、３０日となっております。試用期間を過ぎて、ご利用いただくには、ベクターにて登録ください。

登録料は、9,800 円です。

プロ版は、シェアウェア版から、ライセンスキーの確認機能のみを省いたものであり、通信機能は、シェアウェア版と同じです。ライセンスキーの確認機能がプログラムに含まれませんので、より安定しています。シンプル通信を本格的に、ご利用される方向けです。

プロ版は、シンプル通信を組み込んだ製品を自由に配布することができます。

販売金額は、20,000 円です。

ご購入は、弊社の HP から、お問い合わせください。

目次

紹介

シンプル通信の便利なところ

オムロンのシーケンサは、シリアルポートにパソコンを接続することにより、パソコンからシーケンサの状態を調べたり、設定を変更したりできます。

このような操作は、オムロンが提供するツールを用いるか、あるいは、プログラムを作成して行うことになります。 シンプル通信 For オムロンのコントロールを使えば、簡単に、プログラムを作成することができます。

例： データメモリの、0ch から、1ch 分のデータを取得する

[一般的なプログラム]

"@00RD00000001xx*"**という文字列を、シーケンサに送信し、**"@00RDxxXXXXxx"**を受け取ります。受信したデータから、データ部分（XXXX）を取り出すプログラムを作成します。**

[シンプル通信]

long data = omron1.RDsyn(0);

と記述すると、data **変数に、**0ch **の値が設定されます。**

他にも、232C **ケーブルを外した後、再度、接続しても、通信ができたり、データ受信にイベントを利用できるなど、シーケンサを簡単に制御できるようになっています。**

お使いの開発言語により、使い方の説明を分けていますが、もっとも詳しく説明しているのは、Visual Studio **の項目ですので、他の言語をお使いでも不明な点があれば、こちらの項目をご覧ください。**

シンプル通信の特徴

- Rs-232C 上位リンクにより、リレーとデータメモリのリード、ライトが可能です。
- 任意のコマンドを送信できますので、リンクリレー、タイマーなどのリード、ライトも可能です。
- 送信コマンドは、簡単に動作を試すことのできる同期型と、制御に優れる非同期の両方を用意しています。
- 上位リンク通信は、FINS コマンドと、C コマンドに対応しています。（Ver2 より、FINS コマンドに対応）
- ATL にて開発を行いましたので、コンパクトです。
- VB6 用のライブラリ（標準モジュール）を用意しています。これを利用することで、タイマやカウンタのリードライト、データの一括読み出しなどが可能です。

通信設定

通信ポート	**COM1**、COM2。 （その他、Windows で対応するポートを利用できます）
ボーレート	115200,57600,38400,19200,**9600**,4800,2400,1200
ビット	8,**7**
パリティ	**なし、奇数、偶数**
ストップビット	1、**2**
フロー制御	**なしに固定**

太文字がデフォルトの設定です。

バージョンについて

Ver2 は、Ver1 とソースレベルで、上位互換性があります。ただし、ストップビットの設定は異なります。

使い方

シンプル通信　For　オムロン（以下、シンプル通信コントロール）の使い方。

シンプル通信コントロールは、オムロンのシーケンサの値の読み出し、書き込みが、簡単にできる ActiveX コントロールです。一般的な ActiveX コントロールと同様に、フォームに貼り付けて利用します。フォームに貼り付けずに利用することも可能です。

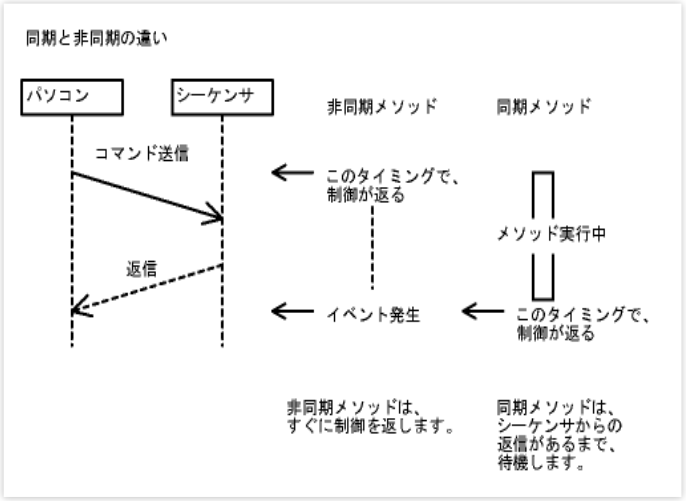
基本的な使い方としては、まず、コントロールをフォームに貼り付け、Rs232C 通信のポート設定を相手の機器の設定に合わせます。

Open メソッドを使ってポートを開き、ReadCH や WriteCH などのメソッドを使い、シーケンサの値を読み出したり、書き込んだりします。

同期と非同期の違い

同期タイプは、メソッドを実行すると、シーケンサからの返信が得られるまで、制御が返りません。つまり、シーケンサからの返信があるまで、次のメソッドを実行できません。

非同期タイプは、メソッドを実行すると、すぐに制御が返ります。シーケンサからの返信を待たないため、シーケンサへの命令が実行できたか把握する前に、次のメソッドが実行されることに注意が必要です。



図：1 同期と非同期の違い

例えば、

int ret = SendCH(...);

ret = SendCH(…);

この場合、２つめのコマンドは、エラーになります。

なぜなら、最初のコマンドの返信を受信し終わる前に、次のコマンドを送信しようとしたからです。

同期系であれば問題ありません。次のコードはエラーになりません。

int ret = SendCHs(…);

ret = SendCHs(…);

コントロールの登録

シンプル通信コントロールをご利用いただくには、Windows に、コントロールを登録する必要があります。VB6 や Visual Studio などの開発ツールでは、開発ツールに登録機能がありますので、そちらを利用してください。

インターネットエクスプローラー（IE）で、クライアントとして、アプリケーション（シンプル通信コントロールを使ったプログラム）を実行される場合は、IE の指示に従って、登録許可を与える必要があります。

エクセルや、IE でのプログラム開発などの場合は、付属の install.bat を、管理者権限で実行し、シンプル通信の ActiveX コントロールを登録します。同様に、uninstall.bat により、登録を解除できます。

install.bat の使い方

エクスプローラで、シンプル通信コントロール（PcToPLC.dll）のあるフォルダを開きます。

install.bat を選択し、右クリックで表示されるメニューから、「管理者として実行」を選びます。

無事に登録されるとメッセージが表示されます。

アンインストールするには、PcToPLC.dll のあるディレクトリで、uninstall.bat を実行してください。

注意事項

６４ビット OS での利用

６４ビット OS と、開発言語の組み合わせによっては、実行時に、「クラスが登録されていません」（エラー:0x80040154）が発生することがあります。（VS2008 の C#において確認）

この場合、開発するアプリケーションのプラットフォームを「Any CPU」から「x86」に変更してください。

デバッグ時に、コントロールが反応しなくなる場合

開発言語によっては、デバッグ時にコントロールが反応しなくなることがあります。（C++で確認）

この場合、デバッグ用のコントロール（debug フォルダにある）を利用して開発を行ってください。開発が終了しましたら、正式なコントロールに戻してください。

デバッグ用と正式用を切り替えるには、install.bat、uninstall.bat **のスクリプトを用いると簡単です。**

PcToPLC.dll **と同じフォルダに、**install.bat、uninstall.bat **ファイルをコピーします。**

正式版の PcToPLC.dll **と同じフォルダにある** uninstall.bat **を管理者権限で実行すると、コントロールの登録が解除されます。**

デバッグ用の PcToPLC.dll **と同じフォルダにある** install.bat **を管理者権限で実行すると、コントロールが登録されます。**

元に戻すには、デバッグ用のフォルダで、uninstall.bat **を実行し、正式版のフォルダで、**install.bat **を実行します。**

VB6.0

最初に、シンプル通信コントロールを VB に追加します。

［プロジェクト］－［コンポーネント］を選択します。

［コントロール］のリスト中に、シンプル通信 for オムロンがあれば、それをチェックします。

無い場合は、［参照］ボタンをクリックして、PcToPLC.dll ファイルを選択します。

ダイアログの［OK］ボタンをクリックして、ダイアログを閉じるとシンプル通信コントロールがツールに追加されます。

コントロールの追加を行わずに、サンプルプログラムを開くと、エラーが表示されますので、ご注意ください。

同期サンプルプログラム（vbsample1）

５秒ごとに、データメモリの 0ch の値を読み出すプログラムです。

1　PLC コントロールの設定

フォームにシンプル通信コントロールを貼り付けます。

シンプル通信コントロールのプロパティを設定します。

設定するのは、Port、Bps、Bit、Parity、StopBit です。

デフォルトでは、Port：1、Bps：9600、Bit：7、Parity：2（even パリティ）、StopBit：2（ストップビット２）となっています。

これをシーケンサの設定と一致させます。

（添付のプログラムでは、9600Bps に設定されています。）

2　ボタンの設置

フォームにラベルを１つ貼り付けます。

Label1 には、シーケンサからの値を表示します。

タイマーコントロールを１つ貼り付けます。

プロパティは、Interval を 5000、Enabled を False にします。

3　メソッドの記述

「フォーム」をクリックして、次を記述します。

```
If Rs1.Open = True Then
  Timer1.Enabled = True
End If
```

これで、プログラムを実行すると、シリアルポートが開き、シーケンサとの通信が始まります。

「Timer1」ボタンをクリックして、次を記述します。

```
Label1 = Rs1.RDsyn(0)
```

これで、５秒毎に、データメモリの０ｃｈの値が読み出されます。

4　実行

これで、プログラムが完成しました。実行してみます。

実行すると、シンプル通信コントロールの登録確認ダイアログが表示されます。

このダイアログが表示されないようにするには、登録が必要です。

ここでは、［OK］をクリックして、次に進みます。

５秒毎に、データメモリの値が読み出され、表示されます。

Label1 に-1 が表示された場合、エラーが発生しています。

シンプル通信コントロールのプロパティがシーケンサの設定と異なっていないか、ケーブルは正しく接続されているかを確認してください。

あるいは、ハイパーターミナルや他のソフトがシリアルポートを利用している可能性があります。

他にシリアルポートを利用しているソフトを閉じてください。

プログラムリスト（vbsample）

```
Private Sub Form_Load()
  If Rs1.Open = True Then
    Timer1.Enabled = True
  End If
End Sub

Private Sub Timer1_Timer()
  Label1.Caption = Rs1.RDsyn(0)
End Sub
```

非同期サンプルプログラム（vbsample2）

非同期メソッド（イベントが発生するメソッド）を使い、データメモリの、0ch の値を読み出します。

シンプル通信コントロールの追加、設定に関する説明は、vbsample と同じですので、省略します。

1　コントロールの追加、設定

フォームにシンプル通信コントロールを貼り付け、プロパティを設定します。

2　ボタンの設置

フォームにボタンを２つ貼り付けます。

Command1 の Caption を［Open］、Command2 の Caption を［View］とします。

フォームにラベルを２つ貼り付けます。

Label1 には、シーケンサからの値、Label2 には、メソッドの戻り値を表示します。

3　メソッドの記述

［Open］ボタンをクリックして、次を記述します。

```
Label2 = Rs1.Open
```

これで、［Open］ボタンをクリックすると、シリアルポートが開き、シーケンサとの通信が始まります。

［View］ボタンをクリックして、次を記述します。

```
Label2 = Rs1.RD(0)
```

これで、［View］ボタンをクリックすると、データメモリの０ｃｈの値が読み出されます。

4　イベントの記述

シンプル通信コントロールをクリックして、次を記述します。

```
Label1 = Res
```

これにより、読み出されたデータメモリの値が、表示されます。

次に通信が出来なかった場合の処理を記述します。

Rs1 の OnError を選択してください。

Private Sub Rs1_OnError(ByVal lErr as Long)の次の行に、次を記述します。

Label1 = "エラー"

通信が出来ない場合、”エラー”と表示されます。

5　実行

これで、プログラムが完成しました。実行してみます。

［Open］をクリックすると、シンプル通信コントロールの登録確認ダイアログが表示されます。

このダイアログが表示されないようにするには、登録が必要です。

ここでは、［OK］をクリックして、次に進みます。

Label2 に True が表示されるはずです。

False が表示された場合、ハイパーターミナルや他のソフトがシリアルポートを利用している可能性があります。他のソフトを閉じてください。

次に、［View］をクリックします。

Label2 に True が表示され、Label1 に数字が表示されれば正常です。

Label2 に False が表示される場合、［View］ボタンをダブルクリックした可能性があります。シーケンサには、１度に１つのコマンドしか送信できません。ダブルクリックにより、１つめの処理が終わらないうちに、次のコマンドを送ると False が返ります。

Label1 に Err が表示された場合、通信に異常があります。シンプル通信コントロールのプロパティが間違っていないか、ケーブルは正しく接続されているかを確認してください。

プログラムリスト（vbsample1）

```
Private Sub Command1_Click()
Label2 = Rs1.Open              ’・・・ポートを開く
End Sub

Private Sub Command2_Click()
Label2 = Rs1.RD(0)             ’・・・データメモリ０ｃｈを読む
End Sub

Private Sub Rs1_OnCom(ByVal Res As Long)
                               ’・・・シーケンサから、データメモリの値が返る
Label1 = Res                   ’・・・値を表示する
End Sub

Private Sub Rs1_OnError(ByVal lErr As Long)  ’・・・通信エラーの発生
Label1 = "Err"
End Sub
```

連続してデータメモリの値を表示する（vbsample3）

データメモリの 0ch の値を表示し続けます。

vbsample1 では、タイマーを使って定期的に値を読み出しましたが、次は、非同期メソッドを用いて、効率的に読み出してみます。

1　フォームの作成

フォームに、シンプル通信コントロールを 1 個、ボタンを 2 個、ラベル 1 個貼り付けます。

ボタンは、通信のスタートとストップに使用します。

Command1 の Caption を［スタート］、Command2 の Caption を［ストップ］とします。

シンプル通信コントロールのプロパティを設定してください。

2　プログラムの記述

プログラムリスト（vbsample2）

```
Private Sub Command1_Click()
    Rs1.RD 0
End Sub

Private Sub Command2_Click()
    Rs1.Close
End Sub

Private Sub Form_Load()
    Rs1.Open
End Sub

Private Sub Rs1_OnCom(ByVal Res As Long)
    Label1 = Res
    Rs1.RD 0
```

```
End Sub

Private Sub Rs1_OnError(ByVal lErr As Long)
   Label1 = "エラー"
   Rs1.RD 0
End Sub
```

３　プログラムの説明

フォームをロードする時に、通信をオープンします。

［スタート］ボタンをクリックすると、データメモリの０ｃｈの値を読み、表示し続けます。データメモリの値が変われば、表示が変わります。

ここで、ケーブルが外れたり、通信エラーが発生した場合、エラーが表示されます。

（ただし、エラーが表示されるまでは、５秒ほど掛かります。これは、シーケンサからの返信が、一定時間経っても来ない場合にエラーとしているためです。）

通信が回復すると、再び、表示されます。

ストップボタンをクリックすると、通信が止まります。

再実行するには、一度、プログラムを終了させる必要があります。

４　改良

ラダープログラムを組むか、ラダーサポートソフトで変更しないと、データメモリの値は変わりません。そこで、パソコンから、データメモリの値を変更してみます。

［スタート］ボタンをクリックすると、データメモリへの書き込みと読み込みを交互に繰り返すプログラムです。書き込み時に、それまでの値＋１を書き込んでいますので、実行すると、数字が徐々に大きくなっていきます。

［ストップ］ボタンをクリックすると、止まります。

プログラムリスト（vbsample3）

```
Dim Flg ' 0：読み込み 1：書き込み 2：ストップ
```

```
Dim n

Private Sub Command1_Click()
   Flg = 0
   n = 0
   Rs1.RD 0
End Sub

Private Sub Command2_Click()
   Flg = 2
End Sub

Private Sub Form_Load()
   Rs1.Open
End Sub

Private Sub Rs1_OnCom(ByVal Res As Long)
   If n > 100 Then n = 0
   Select Case Flg
      Case 0                  '・・・読み込み中だった
         Label1 = Res
         Flg = 1
         Rs1.WD 0, n          '・・・次は書き込む
         n = n + 1
      Case 1                  '・・・書き込み中だった
         Flg = 0
         Rs1.RD 0             '・・・次は読み込む
      Case 2
   End Select
```

```
End Sub

Private Sub Rs1_OnError(ByVal lErr As Long)
   Label1 = "エラー"
   If Flg <> 3 Then
      Flg = 0
      Rs1.RD 0
   End If
End Sub
```

５　改良プログラムの説明

変数 Flg を使って、現在、読み込み中であるか、書き込み中であるかをチェックします。

OnCom イベントにおいて、読み込み中であれば、次の動作である書き込みを実行します。

このとき、変数ｎを１加えています。

書き込み中であれば、次の動作である読み込みを実行します。

［ストップ］がクリックされると、Flg が２になり、ストップします。

もう一度、［スタート］をクリックすると、読み込みから始めます。

任意のコマンドを使用する（vbsample4）

リレーやデータメモリ以外の値を操作したい場合、あるいは、連続したアドレスのデータを読み込みたい場合などでは、上位リンクのコマンドを直接、送信することができます。

サンプルプログラムでは、0～40CH に、1 から 40 までの値を設定したり、0～40CH の値を一度に読み込む例を表しています。

（注意）この例は、オムロンの C コマンドを用いています。FINS コマンドを用いる場合は、メソッドが異なり、文字数の制限も異なります。

設定の場合

```
a = "@00WD0000000100020003000400050006000700080009001 0"
a = a & "0011001200130014001500160017001800190020"
a = a & "0021002200230024002500260027002800290 "
b = "0030003100320033003400350036003700380039004 0"
ret = Rs1.SendExsyn(a, 1) 'デリミタ
ret = Rs1.SendExsyn(b, 0) 'ターミネータ
```

設定の場合、132 文字以上の文字列データを送信することになるため、送信データを分割することになります。サンプルの例では、文字列データを a,b の二つの変数に分割して、記録しています。最初の文字列 a を送信する場合、最後の文字列と区別するために、デリミタを付加しなければなりません。そこで、SendExsyn に第２引数を１にします。最後の文字列 b を送信する場合には、通常どおり、ターミネータを付加するために、第２引数を０にします。

読み込みの場合

```
a = "@00RD00000040"
ret1 = Rs1.SendExsyn(a, 0) '受信１
l = Len(ret1)
ret1 = Mid(ret1, 1, l - 2)
ret2 = Rs1.SendExsyn(Chr(13), 2) '受信２
Text1 = ret1 & ret2
```

読み込みの場合、シーケンサから 132 文字以上の文字列が送信されることになるため、分割して送られてきます。受信１では、通常どおりにコマンドを送信し、シーケンサから受信データ１を受け取ります。受信２では、次のデータを受け取るために、デリミタを送信します。この場合、FCS やターミネータを付加しないため、SendExsyn の第２引数を２にします。

変数 ret1、ret2 には、デリミタ（CR）が含まれていないことに注意してください。

VB6 用のライブラリを使用する（vbsample5）

ライブラリ（mdlomron.bas）を作成するプログラムに組み込みます。

これにより、ライブラリの関数を利用することができます。

読み出し、書き出しのエリアを指定します。

対象のチャネルと、要素数を指定します。

「Write」をクリックすると、指定したエリアに、データが書き込まれます。データは、チャネル毎に、＋１加えた値が書き込まれます。

「Read」をクリックすると、指定したエリアのデータが読み出されます。

その他に、「ステータス読み出し」や、一括読み出しを試すことができます。

FINS コマンドの使い方（vbsample6）

FINS コマンドを使って、データメモリの 0ch から、２チャネル分を読み出します。

行っていることは、vbsample2 と、ほぼ変わりませんが、発生するイベントが異なります。

OnCo mFINS イベントでは、正常時は、値が配列として入りますが、エラー時には、配列(0)に、"Error"が、配列(1)に、エラー番号が入ります。万が一、エラーが発生した場合に備えて、処理を分けていますが、通常、通信時のエラーは、タイムアウトのみであるため、読み出すデータの種類やチャネルが正しければ処理を分ける必要はありません。

Excel

シンプル通信コントロールは、エクセルでもご利用いただけます。

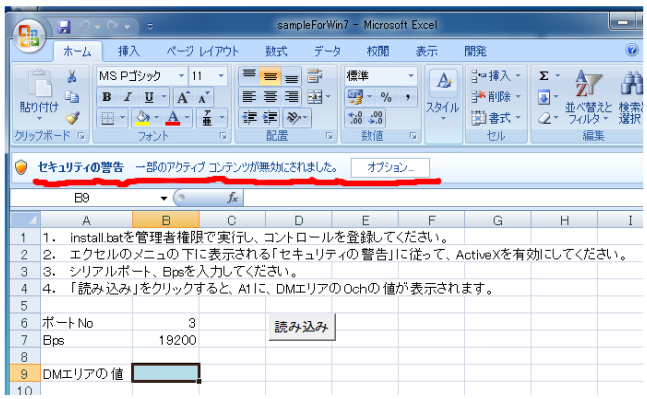
コントロールの登録を参照し、シンプル通信コントロールを登録してください。ただし、
Visual Studio **や、他の開発ツールを利用して、既に登録済みであれば必要はありません。**

セキュリティ対策

エクセル（Excel）のサンプルプログラムを実行するには、セキュリティを解除する必要があります。

以下は、Office2007 での例です。

プログラムを開くと、「セキュリティの警告」が表示されますので、「オプション」をクリックします。

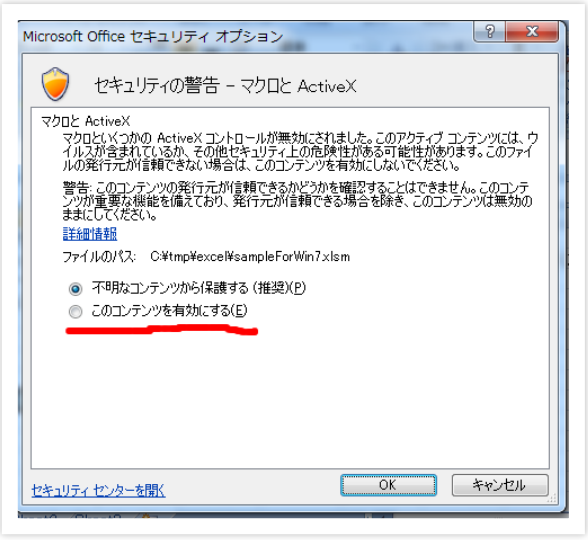


図：2 セキュリティの警告

「セキュリティ　オプション」ダイアログが表示されますので、「このコンテンツを有効にする」に、チェックを入れ、「OK」をクリックします。

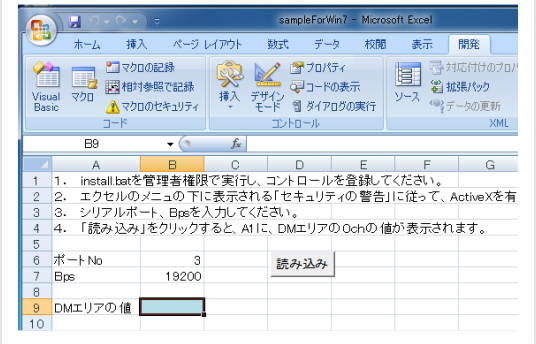
これにより、マクロと、ActiveX コントロールを利用できるようになります。

ただし、シンプル通信コントロールを登録していない場合、サンプルプログラムを実行しても、エラーになります。



図：3 セキュリティオプション

エクセルで、プログラム（マクロ）を開発する場合、エクセルのオプションを開き、「開発」タブが利用できるように設定します。



図：4 開発タブ

開発方法

以下は、Excel 2010 について説明します。

最初に、「開発」タブが表示されるようにします。それには、エクセルの「ファイル」タブから、「オプション」を選びます。「リボンのユーザー設定」で、「開発」にチェックします。

次に、新規ファイルを、マクロ有効ブック（.xlsm）として保存します。

コントロールとして使わない場合

利用するには、エクセルのマクロ内で、new を使って、宣言します。

例

```
Dim rs1 as new Rs
```

コントロールとして使う場合

コントロールとして使う場合、コントロールを、シートに置く方法と、フォームに置く方法があります。

シートに置く場合は、「開発」タブにある「挿入」を選択します。

「コントロールの選択」ダイアログが表示されますので、「カスタム　コントロールの登録」をクリックし、シンプル通信（SimpleComOmron）を登録します。

フォームに置く場合は、Visual Basic の開発画面において、ツールボックスで右クリックし、コントロールの追加ダイアログで、シンプル通信（SimpleComOmron）を登録します。

サンプルプログラム（sample.xlsm）

サンプルプログラムの作成方法を説明します。

先に説明したように、マクロ有効ブック（xlsm）として、sample.xlsm を作成します。

「開発」タブにある「挿入」を選択します。「コントロールの選択」ダイアログが表示されますので、「カスタム　コントロールの登録」をクリックし、シンプル通信（SimpleComOmron）を設置します。プロパティを確認し、シーケンサと通信設定が異なっていれば設定を合わせます。

次に、「開発」タブ「挿入」から、「コマンドボタン（ActiveX コントロール）」を選び、シートに設置します。

設置したボタンを、ダブルクリックすると、コードを入力する画面になります。

ここで、次のように、記述します。

```
Private Sub CommandButton1_Click()
    Dim r


    Rs1.Open
    r = Rs1.ReadCHs("DM", 0, 2)
    If r(0) = "Error" Then
        ActiveSheet.Cells(1, 2) = r(0) & " " & r(1)
    Else
        ActiveSheet.Cells(1, 1) = r(0) & " " & r(1)
    End If
    Rs1.Close
End Sub
```

「デザインモード」をクリックして、デザインモードを解除し、ボタンをクリックします。

プログラムが実行され、シートに、データメモリの値、または、エラー番号が表示されます。

プログラムの説明

Rs1.Open で、シリアルポートを開き、通信できる状態にします。ここでは、開かれたかどうかを確認していませんが、できれば確認した方が良いでしょう。

r = Rs1.ReadCHs では、データメモリの 0ch から２チャネル分を取得します。

r には、正しく値を読み出せたなら値が入り、エラーが発生した場合は、r(0)に、″Error″、r(1)に、エラー番号が入ります。

ActiveSheet.Cells()では、値を表示します。

Rs1.Close で、ポートを閉じます。

非同期型のサンプルプログラム（sample2.xlsm）

サンプルプログラム（sample.xlsm）と同様の処理を、非同期型のメソッドを使って記述したサンプルです。こちらでは、イベントを用いて、データを取得しています。

プログラム

```
Private Sub CommandButton1_Click()
    Dim r
    Rs1.Open
    r = Rs1.ReadCH("DM", 0, 2)
End Sub

Private Sub Rs1_OnComFINS(ByVal command As Variant, ByVal Val As Variant)
    If Val(0) = "Error" Then
        ActiveSheet.Cells(2, 1) = Val(0) & " " & Val(1)
    Else
        ActiveSheet.Cells(1, 1) = Val(0) & " " & Val(1)
    End If
End Sub

Private Sub Rs1_OnError(ByVal Err As Long)
    ActiveSheet.Cells(2, 1) = "TimeOut"
End Sub
```

注意点は、Rs1.Close をボタンクリック内で行っていないことです。非同期型のメソッドは、通信処理中でも制御が戻るため、通信が終わらないうちにポートを閉じてしまう（Rs1.Close を実行する）と、イベントが発生しません。

プログラムとしては、Rs1.Close をプログラムの終了時に行うことが望ましいのですが、シンプル通信コントロールは、自動的にポートを閉じるようになっているため、Close メソッドは必ずしも必要ではありません。

指定の時間内に、シーケンサからのデータを受け取ることができなかった場合、タイムアウトエラーが発生します。これは、OnError イベントで受けることができます。

インターネットエクスプローラ

インターネットエクスプローラでも、ご利用いただけます。

ただし、インターネット上にシンプル通信コントロールを使ったページを設置し、それをユーザーが利用される場合には、セキュリティの設定を変更する必要があります。

セキュリティの設定変更

メニューから、［ツール］－［インターネットオプション］を選択して、インターネットオプションダイアログを表示します。

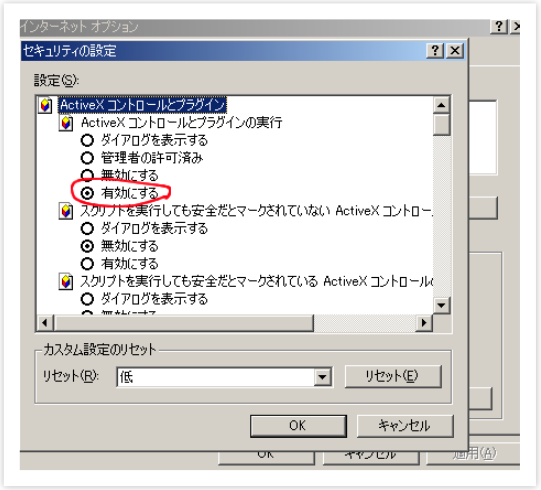
［セキュリティ］タブをクリックします。

［インターネット］、または、［信頼済みサイト］を選択します。

［信頼済みサイト］を選択した場合は、［サイト］をクリックして、このホームページを登録してください。

［レベルのカスタマイズ］をクリックします。

右の画面において、［ActiveX コントロールとプラグインの実行］を有効にします。

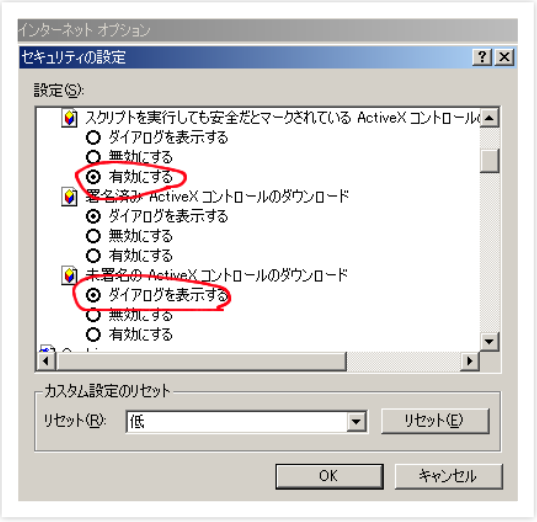


図：5IE でのセキュリティの変更

［スクリプトを実行しても安全だとマークされていない ActeveX コントロール］の項目は、変更する必要はありません。

［スクリプトを実行しても安全だとマークされている ActeveX コントロール］を有効、または、ダイアログを表示するに設定します。

［未署名の ActiveX コントロールのダウンロード］を有効、または、ダイアログを表示するに設定します。



図：6IE でのセキュリティの変更２

注意事項

ホームページで提供する ActiveX コントロールが新しくなったときには、古いコントロールを削除して新しいコントロールをインストールしてください。 ブラウザで再読み込みを行っても新しいバージョンに自動的に変更されることはありません。

サンプルプログラム

ファイル名　osample.htm、osamplej.htm

リレー、または、データメモリの値を読み出したり、書き込んだりするプログラムです。

サーバー用と、ローカル用では、コントロールの記述部分に、CODEBASE の記述が異なるだけです。

<OBJECT ID="Rs" CLASSID="CLSID:75C5FBA8-444F-4130-BFEC-987D93A7C4E7"
CODEBASE="PcToPLC.dll">

赤字の部分が、サーバー用にはあり、シンプル通信コントロールの場所を示しています。ローカル用では、この記述はありません。

使用している言語は、osample.htm **が** VBScript。osamplej.htm **が** JScript **です。**

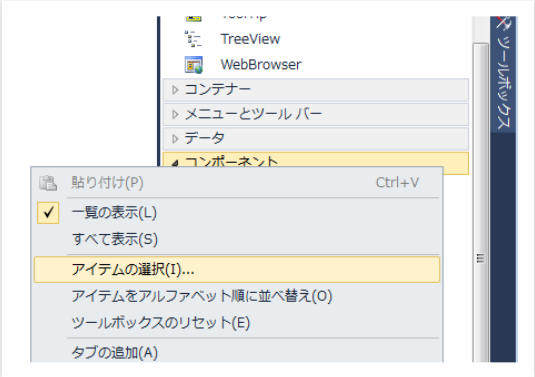
Visual Studio（C#、VB）

シンプル通信コントロールは、VisualStudio でも、ご利用いただけます。

ここでは、Visual Studio 2010 を例に使い方を説明します。

コントロールの登録

ツールボックスを開き、右クリックのメニューから、「アイテムの選択」を開きます。

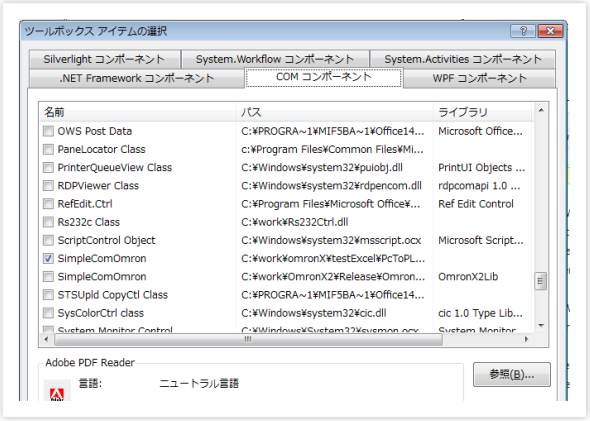


図：7 アイテムの選択

「ツールボックス　アイテム」の選択では、COM タブをクリックし、「参照」ボタンをクリックします。

シンプル通信コントロールを指定して、登録します。

install.bat を実行するなど、既に、他の方法で登録している場合は、チェックだけで済みます。



図：8 アイテムの登録

これで、ツールボックスに、コントロールが追加されますので、他のコントロールと同様に使用します。

サンプルプログラム（C#）

フォルダ：sampleC

データメモリに、値を読み書きするプログラムを作成します。

プログラムの手順としては、最初に、シリアルポートを開きます。

シーケンサに値を書き込む命令を実行します。

シーケンサの値を読み出す命令を実行します。

フォームを閉じる時に、シリアルポートを閉じます。

サンプルプログラムの作成をしながら、具体的に説明します。

コンポーネント、コントロールの配置

シンプル通信コントロールを１つ、ラベルを２つ、ボタンを２つフォームに貼り付けます。

シンプル通信のプロパティを開き、シーケンサと、シリアル通信設定が一致するようにします。

シリアルポートの制御

フォームをクリックして、エディタを開き、プログラムを実行したときに、シリアルポートを開くように、プログラムします。Open **がポートを開くメソッドです。**

```
private void Form1_Load(object sender, EventArgs e) {
    axRs1.Open();
}
```

フォームのイベントで、フォームを閉じたときに、シリアルポートを閉じるようにします。

```
private void Form1_FormClosed(object sender, FormClosedEventArgs e)  {
    axRs1.Close();
}
```

本格的に、シンプル通信コントロールを使われる場合は、「接続」「解除」などのボタンに、これらのメソッドを割り当てて、ポートの ON**、**OFF **を切り替えた方が良いでしょう。**

同期型のメソッド

ボタンをクリックすると、データメモリに値が書き込まれ、それを読み出すようにします。
button1 **をクリックして、ソースを記述します。**

```
private void button1_Click(object sender, EventArgs e)  {
```

```
        int[] send = { 2, 3 };
        object[] dm;
        string s;

        axRs1.WriteCHs("DM", 0, 9, 1);
        axRs1.WriteCHs("DM", 1, send, 2);

        dm = (object[])axRs1.ReadCHs("DM", 0, 3);
        s = "";
        try {
            for (int i = 0; i < dm.Length; i++) {
                s += " " + dm[i].ToString();
            }
            label1.Text = s;
        } catch {
            label1.Text = "Error " + dm[1].ToString();
        }
    }
```

ここでは、WriteCHs メソッドの使い方を説明するために、データメモリの 0ch に書き込む部分と、1ch から 2ch 分書き込む部分に分けています。

最初の、WriteCHs では、 1 ch 分のみであるため、設定する値を、そのまま記述しています。

次の WriteCHs では、2ch 分を書き込むため、配列に値を入れて、メソッドに渡しています。1ch 分でも配列に入れて渡すこともできます。

ReadCHs **メソッドを使い、**0ch **から** 3ch **分を読み出します。**

受け取ったデータにエラーがある場合、例外処理を行います。ただし、このプログラムでは、例外が発生することはありませんので、Error **が表示されることはありません。受け取ったデータを数値として扱うようにプログラムし、エラーを文字列として受け取った場合には、例外が発生します。**

ここで、一度、プログラムを実行してみます。正常であれば、データメモリに書き込んだ値が表示されます。

非同期型のメソッド

次に、非同期型のメソッドの使い方の説明をします。

非同期型のメソッドは、コマンドを送信する部分、データを受け取る部分を別々に記述する必要があります。

準備として、現在、どのメソッドを実行しているのかを記憶する変数を用意します。変数 cnt を用意し、初期化します。

```
public partial class Form1 : Form
{
    public Form1() {
        InitializeComponent();
        cnt = 0;
    }
    private int cnt;
```

赤字の部分を記述します。

button2 をクリックして、最初のコマンドを送るメソッドを記述します。

```
    private void button2_Click(object sender, EventArgs e) {
        cnt = 1;
        axRs1.WriteCH("DM", 0, 11, 1);
    }
```

現在、実行しているメソッドが分かるように、cnt に 1 を設定しています。また、WriteCH メソッドは、データメモリの 0ch に、１１を書き込みます。

WriteCH メソッドは、WriteCHs メソッドと異なり、通信処理の終了を待たずに制御が返ります。ret が正常な値であっても、それは、送信に成功しただけであり、受信を含めた通信が正常に終了したとは限らないことに注意してください。

受信の成功と、エラーはイベントで判断します。

一定時間、経っても、返信がない場合は、エラーとして、OnError イベントが発生します。FINS コマンドに対して返信が返ってきた場合は、OnComFINS イベントが発生します。C

コマンドに対しては、OnCom イベントが、SendEx メソッドに対しては、OnComEx が発生します。

シンプル通信コントロールのプロパティで、イベントを表示し、プログラムを記述します。

```
    private void axRs1_OnError(object sender, AxPCTOPLCLib._IRsEvents_OnErrorEvent
e)
    {
      label1.Text = "タイムアウト";
    }
```

OnError イベントが発生した場合、「タイムアウト」と表示します。

OnComFINS イベントが発生した場合は、少し、複雑になります。このイベントが発生した場合、データメモリへの書き込みで発生したのか、読み込みで発生したのかなど、判別して、処理を行います。ここでは、cnt が 1 の場合は、データメモリへ 1ch 書き込み命令、2 の場合は、2ch 書き込み命令、3 の場合は読み込み命令と決めています。

```
    private void axRs1_OnComFINS(object sender,
AxPCTOPLCLib._IRsEvents_OnComFINSEvent e)
    {
      int[] send = { 21, 31 };
      long ret;
      string s;

      object[] str = (object[])e.command;
      object[] val = (object[])e.val;

      label2.Text = str[0].ToString() + " " + str[1].ToString() + " " + str[2].ToString()
        + " " + str[3].ToString() + " " + str[4].ToString();

      try {
        switch (cnt) {
          case 1:
            if ((string)val[0] == "Ok") {
```

```
                cnt = 2;
                ret = axRs1.WriteCH("DM", 1, send, 2);
            }
            break;
        case 2:
            if ((string)val[0] == "Ok") {
                cnt = 3;
                ret = axRs1.ReadCH("DM", 0, 3);
            }
            break;
        case 3:
            s = "";
            for (int i = 0; i < val.Length; i++) {
                s += " " + val[i].ToString();
            }
            label1.Text = s;
            break;
        }
    } catch {
        label1.Text = "エラー";
    }
}
```

全体的な流れを説明します。

最初に、button2 **がクリックされると、**WriteCH **が実行され、データメモリに値を書き込む命令をシーケンサに送ります。**

シーケンサからの返信により、イベント OnComFINS **が発生します。**

イベント内では、cnt **変数により、最初の** WriteCH **メソッドに対する返信であることを確認し、正常に処理されたのであれば、次の** WriteCH **メソッドを実行します。この** WriteCH **では、**2ch **分の書き込みを行います。**

再び、シーケンサからの返信により、イベント OnComFINS が発生します。

イベント内では、cnt 変数により、２回目の WriteCH メソッドに対する返信であることを確認し、正常に処理されたのであれば、データメモリの値を読み出す ReacCH メソッドを実行します。

シーケンサからの返信により、イベント OnComFINS が発生します。

イベント内では、ReadCH メソッドに対する返信であり、かつ、正常に処理されたのであれば、受け取ったデータを表示します。

OnComFINS 内の処理を順に説明します。

```
object[] str = (object[])e.command;
object[] val = (object[])e.val;
label2.Text = str[0].ToString() + " " + str[1].ToString() + " " + str[2].ToString()
    + " " + str[3].ToString() + " " + str[4].ToString();
```

イベントでは、command と、val の２つの変数を受け取ります。

command は、受信したテキストデータが分割され、配列として入っています。

val は、FINS コマンドにより、値が異なります。

WriteCH は、FINS コマンド（0102）を送信します。返信としては、正常に書き込めたか否かが返ってきますので、val(0)に"Ok"が入っていれば正常に処理されたことが分かります。

ReadCH は、FINS コマンド（0101）を送信します。返信としては、メモリエリアの値が返ってきますので、val には、値が配列として返ります。

プログラムを実行し、button2 をクリックすると、データメモリの値が表示されます。

フォルダ：sampleC1

こちらのサンプルプログラムは、行っていることは、sampleC と同じです。エラー発生時の処理の方法が異なります。実用的なプログラムを作成する場合は、エラー発生時の処理が不可欠です。

サンプルプログラム（VB）

フォルダ：sampleVB

フォルダ：sampleVB1

プログラムの内容は、C#と同じです。

リファレンス

FINS コマンドに関して

プロパティ

名前	意味	設定値	説明
Password (ライセンスキー)	ライセンスキー	ライセンスキー ベクターで得られるライセンスキー	ベクターにて、送金すると、ライセンスキーが得られます。 このプロパティに、そのライセンスキーを設定すると、制限が解除されます。 IE において、PARAM で表記する場合は、"ライセンスキー"という名前で設定します。
Ver	バージョン	読取専用	コントロールのバージョンを取得します。
シリアルポート			
Port	シリアルポートの番号	1,2 など	その他、Windows で対応するポートを設定できます。
Bps	ボーレート	1200,2400,4800, 9600,19200,38400, 57600,115200	シリアルポートのボーレートを設定します。
Bit	ビット	7,8	シリアルポートのビットを設定します。
Parity	パリティ	0:ノンパリティ 1:奇数パリティ 2:偶数パリティ	シリアルポートのパリティを設定します。
StopBit	ストップビット	1：1 ビット 2：2 ビット	シリアルポートのストップビットを設定します。 Ver 1 とは、設定値が異なりますので、ご注意ください。
MachineNo	号機 No	0～255	号機 No を設定します。 RR、RD などのメソッドを使って値を設定、取得する場合に、号機 No を指定します。

名前	意味	設定値	説明
Ready	送信可能	読取専用。	通信コマンド（Send、Read、Write など）を実行できる場合、TRUE を返します。 同期コマンドのみを利用している場合は、利用する必要はありません。非同期コマンドを利用する場合に使います。 これは、主に、直前の通信処理が終わっているかを調べるために使います。
RsWait	ウェイト時間	0～15	FINS のウェイト時間を設定します。
ICF	ネットワーク中継	0:直接 1:ネットワーク	FINS の ICF を設定します。 ネットワークの設定は、サポート対象外
DNA	相手先ネットワークアドレス	0～127	FINS の DNS を設定します。
DA1	相手先ノードアドレス	0～254	FINS の DA1 を設定します。
DA2	相手先号機アドレス	0～255	FINS の DA2 を設定します。
TimeOut	タイムアウト時間 設定単位：ミリ秒		コマンドの返信を待つ時間を指定します。この時間を経ても返信が無い場合、エラーになります。

メソッド

オムロンのシーケンサには、C コマンドと、FINS コマンドの２つの上位リンク用のコマンド体系があります。FINS コマンドの方が多機能であり、最新の機種の性能を生かすことができます。

基本メソッド

名前	説明
Open() as Boolean	ポートを開く。
Close()	ポートを閉じる。

FINS コマンド系メソッド

オムロンのシーケンサの FINS コマンドを使って、通信を行うメソッド

名前		説明
非同期	同期	
ReadCH	ReadCHs	指定エリアのチャネルデータを、指定の数、読み出します。
ReadIO	ReadIOs	指定エリアのチャネルやビットデータを、指定の数、読み出します。
WriteCH	WriteCHs	指定エリアに、チャネルデータを指定の数、書き込みます。
WriteIO	WriteIOs	指定エリアに、チャネルデータやビットデータを、指定の数、書き込みます。
SendFINS	SendFINSs	FINS コマンドを送信します。
	GetRun	シーケンサの動作モードを取得します。
	SetRun	シーケンサの動作モードを設定します。

パソコンからのコマンドに対するシーケンサの返信を受け取ることはできますが、シーケンサから送信されたコマンドを処理することはできません。

C コマンド系メソッド

オムロンのシーケンサの C コマンドを使って、通信を行うメソッド

名前		説明
非同期	同期	
RR	RRsyn	リレーの値を読み出します。
WR	WRsyn	リレーの値を書き込みます。
RD	RRsyn	データメモリの値を読み出します。
WD	WDsyn	データメモリの値を書き込みます。
SendEx	SendExsyn	任意のデータを送信します。 このメソッドは、C コマンド以外を送信することもできます。

FINS コマンド

コマンド	名称	機能
0101	I/O メモリエリアの読出	連続した I/O メモリエリアの内容の読み出し
0102	I/O メモリエリアの書込	連続した I/O メモリエリアの内容の書き込み

その他のコマンドに関しては、シーケンサのマニュアルを参照のこと

OnComFINS イベントにおいて、上記の FINS コマンドに関しては、扱いやすいように値が変換されます。

0101 の場合、受け取ったデータは、数値配列として返ります。

0102 の場合、正常に書き込まれた場合、文字列の"Ok"が返ります。

ここに記述がない FINS コマンドの場合、イベント内では、エラー（エラー番号 -201）となりますが、これは、通信自体は、正常に行われていますので、受け取ったテキストデータを解析して処理を行うようにプログラムします。

例は、OnComFINS イベントの例をご覧ください。

他の FINS コマンドについても、扱いやすいように変換して欲しいとの要望は、弊社まで、ご連絡ください。有料で対応するプログラムを作成いたします。

エリア種別

I/O エリアを下記の文字列で表します。

ReadCH や、WriteCH などのメソッドで、エリアを指定する際に利用します。

文字列	意味
CIO	チャネル I/O
WR	内部補助リレー
HR	保持リレー
AR	特殊補助リレー
TIM	タイマ
CNT	カウンタ
DM	データメモリ
TK	タスクフラグ
IR	インデックスレジスタ
DR	データレジスタ
CP	クロックパルス
CF	コンディションフラグ
EMCB	カレントの拡張データメモリ
EMCN	カレントの拡張データメモリの番号
EM00 ～ EM0F EM10 ～ EM18	拡張データメモリ “EM00”、“EM08”のように指定します。

データの種類

ReadIO、WriteIO メソッドで、データの種類を指定する際に利用します。

文字列	データの意味
B	ビット
BF	強制 ON/OFF 付 ビット
C	チャネル（ワード）
CF	強制 ON/OFF 付 チャネル（ワード）
U	アップフラグ
UF	強制 ON/OFF 付 アップフラグ
NOW	現在値
S	ステータス

I/O メモリ種別 設定表 (CS/CJ モード)

ReadIO、WriteIO メソッドで、エリア種別とデータの種類により、指定できる組み合わせを示します。

例えば、"CIO"と"B"の組み合わせでは、リレーのビット（30）を取得できます。

"TIM"と"C"の組み合わせでは、対象がないため、エラーとなります。

	B	BF	C	CF	U	UF	NOW	S
CIO	30	70	B0	F0				
WR	31	71	B1	F1				
HR	32	72	B2	F2				
AR	33		B3					
TIM					09	49	89	
CNT					09	49	89	
DM	02		82					
TK	06							46
IR							DC	
DR							BC	
CP	07							
CF	07							
EMCB	0A		98					
EMCN							BC	
EM00 - EM0F **EM10 - EM18**	xx		xx					

EM の"xx"は、EM00 の場合、20。EM0F の場合、2F と、文字が変わります。

FINS 系メソッドのエラー値

値	
-1	シーケンサの SEND 命令によるコマンドを受けました。 通常、パソコンからのコマンドに対してシーケンサが返信しますが、シーケンサからの自発的なデータ送信があった場合に発生します。
-11	チャネル（ワード）の指定範囲を超えました。
-12	ビットの指定範囲を超えました。
-13	取得するデータ数が範囲を超えました。
-21	エリア種別の設定が不正です。「CIO」、「WR」などの設定を見直してください。
-22	データの種類の設定が不正です。「B」「C」などの設定を見直してください。
-23	I/O メモリ種別を特定できません。エリア種別と、データの種類の指定を見直してください。
-24	FINS コマンドが不正です。
-30	書き込みデータに不正な値が含まれています。
-101	送信に失敗しました。ポートが開かれていないか、コマンド送信中のため、送信できない状態です。
-102	タイムアウトしました
-201	実装していない FINS コマンドです
-202	データの長さが一致しません
-203	送信コマンドと受信コマンドが一致しません
-205	受信データに異常がありました。
正の値	FINS コマンドの異常を知らせるレスポンスコードが、そのまま、返ります
0	（参考）正常です。

ReadIO、ReadCH などの違い

FINS 系のコマンド送信のベースは、SendFINS と、SendFINSs です。

このメソッドの違いは、SendFINS では、データ送信の後、すぐに、制御が返り、データの受信は、イベントで処理を行います。

SendFINSs は、データの送信後、データを受信するか、タイムアウトすると制御が返り、受信データを処理できます。

SendFINSs は、プログラムは作りやすいのですが、タイムアウトの時間を考慮に入れておく必要があります。

SendFINS と、SendFINSs を使えば、FINS コマンドの処理をプログラムできますが、よく利用される、リレーの状態の参照や、データメモリへの書き込みなどを、これらのメソッドを使ってプログラムするのは、手間が掛かります。

ReadIO や、ReadIOs などは、この手間を省くことができます。さらに、ReadCH や、ReadCHs などは、扱うデータをチャネルに特化することで、より簡単に扱えるようになっています。

従いまして、ReadCH でできることは、ReadIO でもでき、ReadIO でできることは、SendFINS でもできます。このような上下関係になっています。

Open() as Boolean

OpenS() as Boolean

ポートを開き、通信を開始します。 ポートを開けない場合、false **を返します。**

OpenS **は、**Open **の特別なものです。 ライセンスキー（**Password**）が設定されていない場合、**Open **を実行すると、シェアウェアの確認ダイアログが 表示されますが、**OpenS **では、この表示はありません。 しかし、ライセンスキーが設定されていない場合、 使用期限が過ぎると使用できなくなるのは共通です。**OpenS **は、試用期間中に、ダイアログ表示を行いたくない場合や、**ASP **での利用の場合に、使います。**

返り値

正常時は、True **が返ります。エラー時は、**False **が返ります。**

Close()

ポートを閉じます。

ReadCH(area as String, ch as Long, num as Long) as Long

I/O エリアのチャネルデータを、読み出し開始チャネルから指定数、読み出します。（非同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
ch	チャネル	読み出し開始チャネルを指定します
num	データ数	

イベント

OnComFINS	正常時、データが、数値の配列として返ります。 通信が正常で、エラーが発生した場合、エラーコードが、文字列配列として返ります。
OnError	通信に失敗した場合、このイベントが発生します。

返り値

正常時は、０が返ります。エラー時は、エラー値が返ります。

例１

データメモリの０チャネルから、２チャネル分を取得する

```
ReadCH("DM", 0, 2);
```

ReadCHs(area as String, ch as Long, num as Long) as Variant

I/O エリアのチャネルデータを、読み出し開始チャネルから指定数、読み出します。（同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
ch	チャネル	読み出し開始チャネルを指定します
num	データ数	

返り値

正常時は、データの配列が返ります。

エラー時は、文字列配列が返ります。ret(0)には、"Error"、ret(1)には、エラー値が文字列として入ります。

例 1

データメモリの 0 チャネルから、2 チャネル分を取得する

```
Dim ret
ret = ReadCHs("DM", 0, 2);
```

正常であれば、ret(0),ret(1),ret(2)に、データが入っています。

エラーの場合は、ret(0)には、"Error",ret(1)には、エラー番号が文字列として入っています。

ReadIO(area as String, dataType as String, ch as Long, bit as Long, num as Long) as Long

I/O エリアのチャネルデータやビットデータを、読み出し開始チャネルから指定数、読み出します。（非同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
dataType	データの種類	チャネルやビットなどのデータの種類を文字列で指定します。
ch	チャネル	読み出し開始チャネルを指定します。
bit	ビット	読み出し開始ビットを指定します。
num	データ数	

イベント

OnComFINS	正常時、データが、数値の配列として返ります。 通信が正常で、エラーが発生した場合、エラーコードが、文字列配列として返ります。
OnError	通信に失敗した場合、このイベントが発生します。

返り値

正常時は、０が返ります。エラー時は、エラー値が返ります。

例１

データメモリの０チャネルから、２チャネル分を取得する

```
ReadIO("DM","C", 0, 0, 2);
```

例２

リレーの２チャネルの１ビットから、３ビット分を取得する

```
ReadIO("CIO","B", 2, 1, 3);
```

ReadIOs(area as String, dataType as String, ch as Long, bit as Long, num as Long) as Variant

I/O エリアのチャネルデータやビットデータを、読み出し開始チャネルから指定数、読み出します。（同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
dataType	データの種類	チャネルやビットなどのデータの種類を文字列で指定します。
ch	チャネル	読み出し開始チャネルを指定します
bit	ビット	読み出し開始ビットを指定します。
num	データ数	

返り値

正常時は、データの配列が返ります。

エラー時は、文字列配列が返ります。ret(0)には、”Error”、ret(1)には、エラー値が文字列として入ります。

例 1

データメモリの 0 チャネルから、 3 チャネル分を取得する

```
Dim ret

ret = ReadIOs("DM","C", 0, 0, 3)
```

正常であれば、ret(0),ret(1),ret(2)に、データが入っています。

エラーの場合は、ret(0)には、"Error",ret(1)には、エラー番号が文字列として入っています。

WriteCH(area as String, ch as Long, data as Variant, num as Long) as Long

I/O エリアのチャネルデータを、書き込み開始チャネルから指定数、書き込みます。（非同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
ch	チャネル	書き込み開始チャネルを指定します
data	データ	チャネルデータが１つの場合は、そのまま。 複数の場合は、配列で指定します。
num	データ数	data に与えられた配列数と、データ数が異なる場合、少ない方が優先されます。

イベント

OnComFINS	正常時、val(0)に文字列の"Ok"が返ります。 通信が正常で、エラーが発生した場合、エラーコードが、文字列配列として返ります。
OnError	通信に失敗した場合、このイベントが発生します。

返り値

正常時は、０が返ります。エラー時は、エラー値が返ります。

例１

データメモリの０チャネルにデータ（４）を書き込む。

```
WriteCH("DM", 0, 4, 2)
```

例２

データメモリの０チャネルに、２チャネル分を書き込む。

```
d[0] = 1;
d[1] = 3;
WriteCH("DM", 0, d, 2);
```

WriteCHs(area as String, ch as Long, data as Variant, num as Long) as Long

I/O エリアのチャネルデータを、書き込み開始チャネルから指定数、書き込みます。（同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
ch	チャネル	読み出し開始チャネルを指定します
data	データ	チャネルデータが１つの場合は、そのまま。 複数の場合は、配列で指定します。
num	データ数	data に与えられた配列数と、データ数が異なる場合、少ない方が優先されます。

返り値

正常時は、０が返ります。エラー時は、エラー値が返ります。

例１

データメモリの０チャネルにデータ（４）を書き込む。

```
WriteCHs("DM", 0, 4, 2);
```

例２

データメモリの０チャネルから、２チャネル分を書き込む。

```
d[0] = 1;
d[1] = 3;
WriteCHs("DM", 0, d, 2);
```

WriteIO(area as String, dataType as String, ch as Long, bit as Long, data as Variant, num as Long) as Long

I/O エリアのチャネルデータやビットデータを、書き込み開始チャネルから指定数、書き込みます。（非同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
dataType	データの種類	チャネルやビットなどのデータの種類を文字列で指定します。
ch	チャネル	読み出し開始チャネルを指定します。
bit	ビット	読み出し開始ビットを指定します。
data	データ	チャネルデータが１つの場合は、そのまま。 複数の場合は、配列で指定します。
num	データ数	data に与えられた配列数と、データ数が異なる場合、少ない方が優先されます。

イベント

OnComFINS	正常時、val(0)に文字列の”Ok”が返ります。 通信が正常で、エラーが発生した場合、エラーコードが、文字列配列として返ります。
OnError	通信に失敗した場合、このイベントが発生します。

返り値

正常時は、０が返ります。エラー時は、エラー値が返ります。

例１　データメモリの０チャネルにデータ（４）を書き込む。

```
WriteIO("DM", "C", 0, 0, 4, 2);
```

例２　データメモリの０チャネルから、２チャネル分を書き込む。

```
d[0] = 1;
d[1] = 3;
WriteIO("DM", "C", 0, 0, d, 2);
```

WriteIOs(area as String, dataType as String, ch as Long, bit as Long, data as Variant, num as Long) as Long

I/O エリアのチャネルデータやビットデータを、書き込み開始チャネルから指定数、書き込みます。（同期）

パラメータ

area	エリア種別	I/O エリアを文字列で指定します
dataType	データの種類	チャネルやビットなどのデータの種類を文字列で指定します。
ch	チャネル	読み出し開始チャネルを指定します
bit	ビット	読み出し開始ビットを指定します。
data	データ	チャネルデータが 1 つの場合は、そのまま。 複数の場合は、配列で指定します。
num	データ数	data に与えられた配列数と、データ数が異なる場合、少ない方が優先されます。

返り値

正常時は、 0 が返ります。エラー時は、エラー値が返ります。

例 1

データメモリの 0 チャネルにデータ（4）を書き込む。

```
WriteIOs("DM", "C", 0, 0, 4, 2);
```

例 2

データメモリの 0 チャネルから、2 チャネル分を書き込む。

```
d[0] = 1;
d[1] = 3;
WriteIOs("DM", "C", 0, 0, d, 2);
```

SendFINS(command as String, data as String) as Long

FINS コマンドの送信（非同期）

このメソッドにより、全ての FINS コマンドを利用することができます。

パラメータ

command	FINS コマンド	FINS コマンドを文字列で指定します。
data	データ	FINS コマンドに続く任意の文字列を指定します。

イベント

OnComFINS	送信した FINS コマンドにより、イベントで受け取る変数の値が変わります。 エリアデータの読み出し（0101）の場合には、データを配列で受け取ります。 エリアデータの書き込み（0102）の場合には、val(0)に文字列の"Ok"が渡されます。 その他の FINS コマンドは、解析が実装されていないため、エラー番号（-201）が渡されますが、これは、通信自体は正常に行われています。command 引数には受け取った文字列が含まれていますので、このデータを用いて、解析プログラムを作ります。 通信が正常で、エラーが発生した場合、エラーコードが、文字列配列として返ります。 詳しくは、イベントの項目を参照してください。
OnError	通信に失敗した場合、このイベントが発生します。

返り値

正常時は、０が返ります。エラー時は、エラー値が返ります。

例１　リレーの０チャネルから、２チャネル分を取得する。

```
SendFINS("0101", "B00000000002");
```

例２　シーケンサの動作モードを得る。動作モードの判別は、OnComFINS の項目を参照

```
omron1.SendFINS("0601", "");
```

SendFINSs(command as String, data as String) as String

FINS コマンドの送信（同期）

このメソッドにより、全ての FINS コマンドを利用することができます。

パラメータ

command	FINS コマンド	FINS コマンドを文字列で指定します。
data	データ	FINS コマンドに続く任意の文字列を指定します。

返り値

正常時は、返信データが文字列で返ります。エラー時は、空文字列が返ります。

この文字列データは、FINS コマンド、終了コード、テキストデータを含みます。@や、FCS は含みません。

例 1

データメモリの 1 チャネルから、2 チャネル分を取得する

```
String ret = SendFINSs("0101", "820001000002");
```

データは文字列で返りますので、数値に分解するプログラムを組む必要があります。

GetRun() as Long

シーケンサの動作を取得します。

返り値

-1	エラー。
0	プログラムモード
1	モニタモード
2	運転モード

SetRun(mode as Short) as Long

シーケンサの動作を設定します。

パラメータ

mode に下記の値を設定します。

0	プログラムモード
1	モニタモード
2	運転モード

返り値

正常時は、 0 が返ります。エラー時は、エラー値が返ります。

RR(ch as Long) as Boolean

リレー読み出し（非同期）

チャネル（ch）のリレー（内部リレー）の値を読み出すコマンドを送信します。

シーケンサのデータは、OnCom イベントを使って受け取ります。

パラメータ

ｃｈ：読み出すチャネル

イベント

OnCom	正常時、データが、数値として返ります。
OnError	通信に失敗した場合、このイベントが発生します。

戻り値

正常時は、True が返ります。エラー時は、False が返ります。

RRsyn(ch as Long) as Long

リレー読み出し（同期）

チャネル（ch）のリレー（内部リレー）の値を読み出します。

パラメータ

ｃｈ；読み出すチャネル

戻り値

正常時は、リレーの値が返ります。エラー時は、－１が返ります

WR(ch as Long, val as Long) as Boolean

リレー書き込み（非同期）

チャネル（ch）のリレー（内部リレー）に値（val）を書き込むコマンドを送信します。

シーケンサのデータは、OnCom イベントを使って受け取ります。

パラメータ

ch	**チャネル**	**書き込むチャネル**
val	**データ**	**書き込む値**

イベント

OnCom	**正常時、0 が返ります。**
OnError	**通信に失敗した場合、このイベントが発生します。**

戻り値

正常時は、リレーの値が返ります。エラー時は、－1 が返ります

WRsyn(ch as Long, val as Long) as Long

リレー書き込み（同期）

チャネル（ch）のリレー（内部リレー）に値（val）を書き込みます。

パラメータ

ch	**チャネル**	**書き込むチャネル**
val	**データ**	**書き込む値**

戻り値

正常時は、0 が返ります。エラー時は、－1 が返ります

RD(ch as Long) as Boolean

データメモリ読み出し（非同期）

チャネル（ch）のデータメモリの値を読み出すコマンドを送信します。

シーケンサのデータは、OnCom イベントを使って受け取ります。

パラメータ

ｃｈ：読み出すチャネル

イベント

OnCom	**正常時、データが、数値として返ります。**
OnError	**通信に失敗した場合、このイベントが発生します。**

戻り値

正常時は、True が返ります。エラー時は、False が返ります。

RDsyn(ch as Long) as Long

データメモリ読み出し（同期）

チャネル（ch）のデータメモリの値を読み出します。

パラメータ

ｃｈ；読み出すチャネル

戻り値

正常時は、データメモリの値が返ります。エラー時は、－１が返ります

WD(ch as Long, val as Long) as Boolean

データメモリ書き込み（非同期）

チャネル（ch）のデータメモリに値（val）を書き込むコマンドを送信します。

シーケンサのデータは、OnCom イベントを使って受け取ります。

パラメータ

ch	**チャネル**	**書き込むチャネル**
val	**データ**	**書き込む値**

イベント

OnCom	**正常時、０が返ります。**
OnError	**通信に失敗した場合、このイベントが発生します。**

戻り値

正常時は、データメモリの値が返ります。エラー時は、－１が返ります

WDsyn(ch as Long, val as Long) as Long

データメモリ書き込み（同期）

チャネル（ch）のデータメモリに値（val）を書き込みます。

パラメータ

ch	**チャネル**	**書き込むチャネル**
val	**データ**	**書き込む値**

戻り値

正常時は、０が返ります。エラー時は、－１が返ります

SendEx(val as String, Type as Long) as Boolean

任意のデータの送信（非同期）

シーケンサに任意のコマンドを送信することができます。

コントロールは、FCS とターミネータを自動的に付けて送信しますので、FCS 以降の文字列をメソッドに与える必要はありません。

シーケンサのデータは、OnComEx イベントを使って受け取ります。

パラメータ

val	**送信データ**	
type	**送信のタイプ** **0：通常** **1：デリミタ** **2：なし**	**0 が指定されると、与えられた文字列に、FCS とターミネータを自動的に付けて送信します。** **1 が指定されると、与えられた文字列に、FCS とデリミタを自動的に付けて送信します。** **2 が指定されると、与えられた文字列のみを送信します。** **通常は、0 を指定します。**

イベント

OnComEx	**正常時、受信した文字列が返ります。**
OnError	**通信に失敗した場合、このイベントが発生します。**

返り値

正常時は、True。エラー時は、False が返ります。

詳細

C コマンドの場合、132 文字以上のコマンドを、一度に送信することができませんので、分割して、送信することになります。

この場合、最後の文字列と、途中の文字列を区別するために途中の文字列は、デリミタを付加した文字列となります。

例えば、300 文字を送信する場合、３分割して、送信しますが、１、２番目の分については、デリミタを、最後の分についてターミネータを付加する必要があります。

例

SendEx("@00TSE",0)

この例を実行すると、OnComEx イベントが発生し、@00TSE02*を得ることができます。

SendExsyn(val as String, Type as Long) as String

任意のデータの送信（同期）

シーケンサに任意のコマンドを送信し、返信を受け取ります。

SendEx が送信後、すぐに制御を返すのに対して、この関数は、シーケンサからの返信を待って、制御を返します。

パラメータ

val	**送信データ**	
type	**送信のタイプ** **0：通常** **1：デリミタ** **2：なし**	**0 が指定されると、与えられた文字列に、FCS とターミネータを自動的に付けて送信します。** **1 が指定されると、与えられた文字列に、FCS とデリミタを自動的に付けて送信します。** **2 が指定されると、与えられた文字列のみを送信します。** **通常は、0 を指定します。**

返り値

正常に終了した場合、シーケンサからのレスポンスデータが返ります。

ただし、シーケンサからレスポンスデータのうち、デリミタ（CR）は、含まれません。

エラーが発生した場合、空文字列を返します。

イベント

イベントは、非同期メソッドを実行した場合のみ発生します。

ただし、シーケンサから、FINS コマンドを送信した場合も発生することがあります。

OnComFINS	**FINS コマンドに対する返信を受信**
OnCom	**C コマンド（RD、RR）のデータ受信**
OnComEx	**SendEx の返信値を受信。**
OnError	**タイムアウト発生**

OnComFINS(command as Variant, val as Variant)

FINS コマンドに対する返信を受信

シーケンサに非同期のメソッドを使用して、FINS コマンドを送信し、シーケンサより応答が返ってきた場合、イベントが発生します。

また、シーケンサの SEND 命令によるデータを受信した場合は、エラー（エラー番号-1）が発生します。

パラメータ

<table>
<tr><td>command</td><td>返信が分割され配列として渡されます。配列の文字列を連結すると、シーケンサからの返信になります。
<table>
<tr><td>command(0)</td><td>コマンド前までのデータ。
ICF、DA2 などが含まれます。</td></tr>
<tr><td>command(1)</td><td>コマンドコード。
“0101”、”0102”など、４文字のコマンドコード。</td></tr>
<tr><td>command(2)</td><td>終了コード
“0000”、”0101”など、４文字のレスポンスコード。</td></tr>
<tr><td>command(3)</td><td>終了コードに続くデータ
データの読み出し系のコマンドの場合、読み出されたデータ。</td></tr>
<tr><td>command(4)</td><td>FCS。２文字。</td></tr>
</table>
通常、このデータを扱うことはありません。
I/O メモリエリアの複合読出コマンドを、SendFINS メソッドを使って送信するなど、送信メソッドを作成する場合に使います。</td></tr>
<tr><td>val</td><td>データ読み出し系のコマンド（ReadCH、ReadIO）を送信した場合、チャネルやビットデータが配列として渡されます。
データ書き込み系のコマンド（WriteCH、WriteIO）を送信した場合、val(0)に文字列の”Ok”が渡されます。
その他の FINS コマンドは、解析が実装されていないため、エラー番号（-201）が渡されますが、これは、通信自体は正常に行われています。command 引数には受け取った文字列が含まれていますので、このデータを用いて、解析プログラムを作ります。
エラーが発生した場合、val(0)に文字列の”Error”が渡され、val(1)にエラー番号が渡されます。</td></tr>
</table>

FINS コマンドの受信データは、対応済みの FINS コマンド（0101、0102）であれば、利用しやすいように、値の配列や、文字列に変換されます。未対応の FINS コマンドの場合、変換方法が決まっていないため、変換が行われません。このため、エラー（-201）が返ります。しかし、このエラーは、通信自体は正常に行われているため、command(3)の文字列データを解析して、処理を行います。

例　シーケンサの動作モードを得る

ボタンなどで、

```
omron1.SendFINS("0601", "");
```

を実行します。これは、シーケンサの動作モードを取得する FINS コマンドを送信します。

OnComFINS イベント内では、

```
object[] command = (object[])(e.command);
object[] resCode = (object[])e.val;

try{
  if (int.Parse((string) resCode[1]) == -201){
    string resData = (string)command[3];
    int runMode = int.Parse( resData.Substring(2, 2));
    switch (runMode) {
      case 0:
        lblTest.Text = "プログラム";
        break;
      case 2:
        lblTest.Text = "モニタ";
        break;
      case 4:
        lblTest.Text = "運転";
        break;
    }
```

```
  }
 }catch{
 }
```

最初に、シーケンサからの文字列データ（command）と、それを分析したデータ（val）を、object の配列として取得します。

未対応の FINS コマンドのため、分析では、エラー（-201）となります。

シーケンサからの文字列データの内、状態を示す文字列（command[3]）から、動作モードを示す文字列を抜き出し、数値に変換します。

この数値により、「プログラム」「モニタ」「運転」のモードを判別します。

例　C#（VS2010）

```
private void plc_OnComFINS(object sender, OnComFINSEvent e)
{
    object[] obj = (object[])e.val;
    string data = "";
    foreach (object o in obj) {
        data += ":" + o.ToString();
    }
    txtEventData.Text = data;
}
```

テキストボックス（txtEventData）に、返信データが表示されます。

読み出し系のコマンドを送信し、正常に受信した場合、チャネルやビットなどのデータが表示されます。エラーが発生した場合は、"Error"、書き込み系の場合は、"Ok"が表示されます。

OnCom(Res as Long)

C コマンド（RR、RD）のデータ受信

シーケンサに非同期のメソッドを使用して、コマンドを送信し、シーケンサより正常に応答が返ってきた時、イベントが発生します。

RR、RD コマンドを送信した場合は、Res に値が入ります。

パラメータ

Res	RR、RD コマンドを送信した場合は、Res に値が入ります。

このイベント内で、RR、RD などのコマンドを実行できますが、その場合は、すぐに、この関数を抜けるようにしてください。長い処理を行うと、この関数を実行中に、再びイベントが発生し、動作が不安定になります。

例：危険な記述

```
Private Sub Rs1.OnCom(Byval Res as Long)
  Rs1.RR 0
  for i = 0 to 1000000
    …
    この関数を実行中に、次のイベントが発生し２重に実行されてしまいます。
  next
End Sub
```

また、コマンド送信後の構文にブレークポイントを設定し停止させると、再開してもイベントが発生しません。

上記の例では、For 文にブレークポイントを設定すると、シーケンサからデータが返ってきても、次のイベントが発生しません。

OnComEx(Res as String)

SendEx を使った場合、返信値を、このイベントで取得します。

シーケンサに SendEx 関数を使ってデータを送信し、シーケンサより正常に応答が返ってきた時、イベントが発生します。

パラメータ

Res	シーケンサからのレスポンスデータ

シーケンサからレスポンスデータのうち、デリミタ（CR）は、Res に含まれません。

例えば、

SendEx("@00TSE",0)

を実行した場合、「@00TSE02*」が Res に入り、シーケンサからのデータに含まれる（CR）は除かれます。

OnError(Err as Long)

タイムアウトエラー。

シーケンサにコマンドを送信し、シーケンサより応答が返ってこない場合、イベントが発生します。

パラメータ

Err	エラー番号

VB6 用ライブラリ

C コマンドを利用した通信を行うライブラリです。

使い方の詳細は、付属の refvb.htm **をご覧ください。**

Q&A

・.NET 版との違いは、どこですか？

.NET 版は、.NET Framework で作られています。付属のライブラリも、.NET Framework 用になっています。コントロールの基本的な機能は、「シンプル通信 For オムロン」の方が、FINS コマンドに対応している分、上です。

.NET 版に付属のライブラリが不要であれば、機能的に上位の「シンプル通信 For オムロン」をお使いください。

・C コマンドと、FINS コマンドを混在できますか？

基本的に問題なく、ご利用いただけます。

同期系のコマンドを使う場合は気にする点はありませんが。非同期系に関しては、発生するイベントが異なることなど、注意が必要です。

付録

履歴

Ver2.0.1（2011/12/11）**ベータ２版公開**

・SendFINS **コマンドの仕様を変更**

・FINS **系のメソッドで、正常に終了した場合、０を返すように変更**

Ver2.0.2（2011/12/19）**正式公開**

・**シーケンサの** SEND **命令によるデータについて、イベントが発生するように改良。**

Ver2.0.2（2012/11/12）**マニュアルの改訂**

・**マニュアルに、６４ビット** OS **での注意事項を追記。**

・**デバッグ用の** ActiveX **コントロールを同個。**