Delphi、Lazarusにて 強引な手法で シリアル通信を行う

前回までの動画は、Windows10の 環境上で Delphi 10.4、Lazarusに関する比較的　初心者向けの内容でした。 前回で Delphi 10.4、Lazarusの動画は 終わろうと思っていました。

が、その前に　どうしても　一つ実現したいと思う事が ありました。それは、Delphi、Lazarusにて シリアル通信を 行う事です。 何故かというと、 この YouTubeチャンネルでは、組み込みマイコンを中心とする電子工作が　メインテーマになっています。 で、特にローエンド、ミドルレンジのマイコンと、パソコン間で　データ通信を行うとすれば、シリアル通信が、定番です。

しかし、Delphi 10.4、Lazarusに関しては、シリアル通信の機能をサポートするライブラリは、標準では　付いていません。 初心者には、難しい内容になりますが、よろしければ 見て下さい。

ということで、どうにかして、Windows10の 環境上の　Delphi 10.4、Lazarusにて、シリアル通信を行える環境を用意出来ないかと、いろいろ検討しました。

昔、1999年頃 Delphi5 にて、使用出来るRS232C通信コンポーネントを作られたN.Nさんという方がおられて、CQ出版社の トランジスタ技術か Interface どちらか忘れましたが、付録のCD-ROMに、シリアル通信を含む FA用途の　ビジュアルコンポーネントのソース等を公開しておられました。

私は WinXPの古いノートPCを、今もスタンドアロンで使用していますが、このPCに、上記シリアル通信コンポーネントを含めたDelphi5を　インストールしています。 動画で　Delphi5の rs232c コンポーネントを　ちょっと お見せします。

Delphi、Lazarusにて どのように シリアル通信を実現するか

N.Nさんが 作られた Delphi5用の rs232cコンポーネントですが、どのように Delphi 10.4、Lazarusで、使用できるか、3通りの方法を考えました。

① まず、一番正当なやり方で、rs232cコンポーネントのソースを Delphi 10.4に持って行き、ビジュアルコンポーネントとして登録する事が、考えられます。 実際に やってみましたが、コンポーネント用の基底クラスの構造が、全面的に変わっているようで、互換性が、無いです。 エラーが、滝のように出ます。 移植は、無理と判断しました。

② Delphi5のコンポーネント部分を DLL化して Delphi 10.4から DLL呼び出しして、シリアル通信は、出来ないか。？

これも、やる前から無理だろうな。という予測がありました。DLLは、実行時に連結するダイナミックライブラリです。 中身は、通常 フック関数等のAPI関数等を呼び出す用途で使用されます。DLLの中身の関数は基本 静的な関数です。 コンポーネントライブラリは、動的にObjectを生成して動かすものですし、特にrs232cコンポーネントは、内部で 送信、受信のループ処理に マルチスレッドを使用しています。DLLの中でマルチスレッドの処理を行うのは、無理と判断しました。

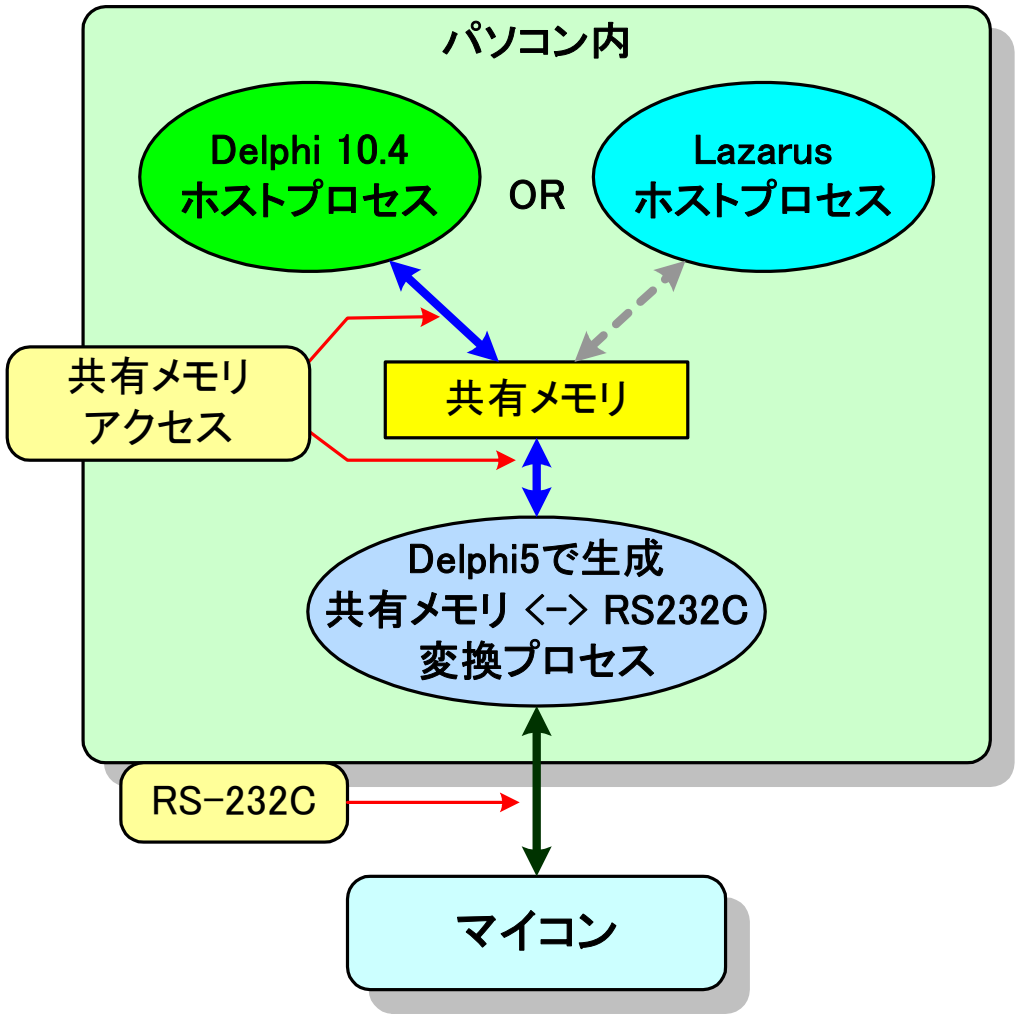
③ Delphi5側で、rs232cコンポーネントを アクセスするプログラムを作成し、プロセス間通信で、Delphi 10.4のアプリと データの やり取りを行わせるというやり方です。 結果から いうと この方法は何とか成功しました。

③の プロセス間通信とは、どのようなやり方なのか？

③ プロセス間通信とは、2つのプロセス間（ プロセスとは、実行プログラムが、動いている状態を示す。）で、通信を行う事です。
具体的な方法として、パイプ、セマフォ、共有メモリ、ソケット通信が ありま す。
今回は、アクセスが簡単な 共有メモリで プロセス間通信を実現しました。

右の図で、上にある楕円で、左側が Delphi 10.4 ホストプロセスと書いてますが、RS-232C を 使ってマイコンと通信したい というアプリです。Lazarusでも 共有メモリをアクセスできましたので、Lazarusでも同様の事が出来ます。

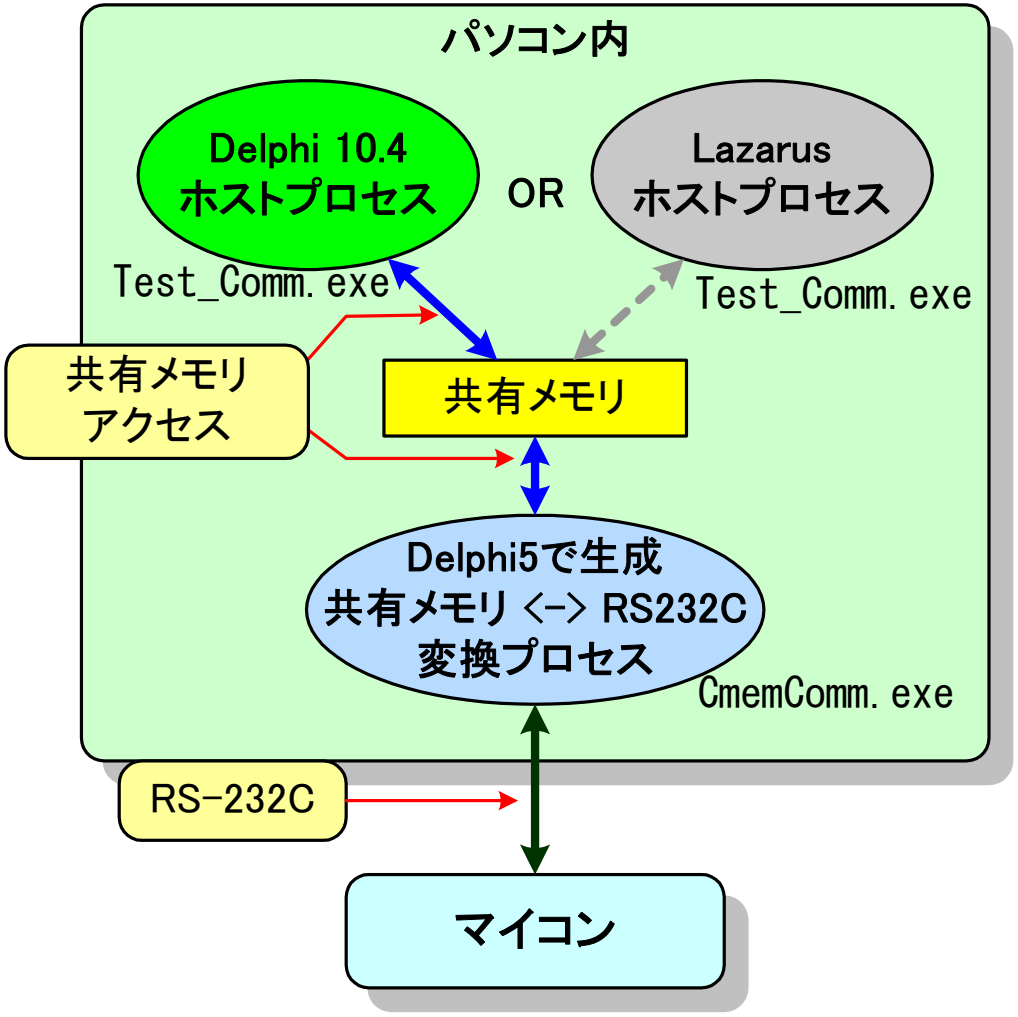
但し、2つのホストプロセスで 同時に 共有メモリは、アクセス出来ません。



前ページの続きですが、ホストプロセスは、Delphi 10.4 か、Lazarus のどちらか片方を 共有メモリに接続する事になります。 右の図では Delphi 10.4側の ホストプロセスを接続したイメージで、描いてます。

共有メモリの下の Delphi5で作成した共有メモリ、RS232C 変換プロセスですが、共有メモリ上のデータと RS232Cシリアル通信の　中継動作を行っています。 共有メモリの生成は、Delphi5で作成した変換プロセスにて、起動時に生成しています。 よって 最初に 変換プロセスを起動する必要が あります。そして、ホストプロセスが動作中は、変換プロセスは、メモリ上に常駐しておく必要が あります。通常は、フォームを最小化しておけば、いいです。

それと、変換プロセスには、いくつかのデバッグ機能を付けています。 順次説明して行きます。

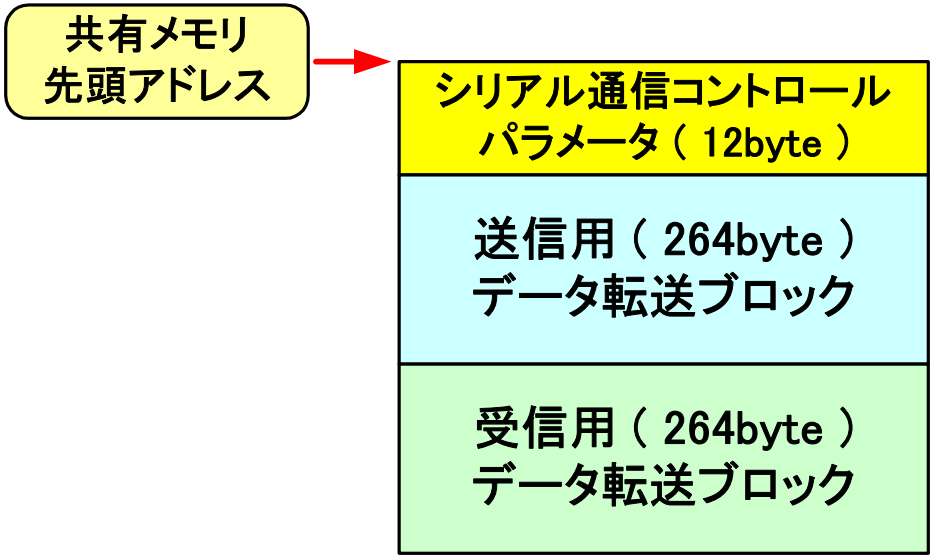


共有メモリ内の データマップ

まず、共有メモリ内で、どのようにデータを割り付けてあるかを 説明します。大きく 3つに 分かれます。

① シリアル通信コントロールパラメータ
② 送信用 データ転送ブロック
③ 受信用 データ転送ブロック

の3つです。 ②と ③は、転送方向は逆ですが構造的には 同じものです。 ②と ③の転送用バッファサイズは、260byteですが、最大 1ブロック 256byteとして扱って下さい。
1ブロックとは、連続して転送する 1本の電文サイズの事です。 それより大きいデータは、いくつかの複数ブロックに分けて転送して下さい。



① シリアル通信コントロールパラメータは、RS232Cオープン時必要となる、ポート番号、ボーレイト、データ長、パリテイ 等です。

①、②、③に共通して存在する hs というフラグが、あります。 これは、ホストプロセスと、変換プロセス間で、タイミング調停を 行うもので、ハンドシェークの略です。

共有メモリ内の 詳細 レコード宣言 1

```
TCMEM_SERIAL_PM = record // シリアル通信コントロールパラメータ
    hs:     Byte;       // 書き込み完了時=1／読み出し完了時=0
    pad_1:  BYTE;       // 予備_1
    dlen:   BYTE;       // データ長 7 or 8
    pbit:   BYTE;       // パリティビット 0=Non, 1=Odd, 2=Even
                        // STOP bit=1 固定
    bps:    Integer;    // 通信速度 b/s
    port:   BYTE;       // ComPort番号： 1 ～ 20
    opcl:   BYTE;       // シリアルポート Open=1 , Close=0
    dtr:    BYTE;       // 0 = OFF , 1 = ON
    rts:    BYTE;       // 0 = OFF , 1 = ON
end;
```

共有メモリ内の 詳細 レコード宣言 2

```
TCMEM_BLOCK = record     // データ転送ブロック
    hs:      Byte;       // 書き込み完了時=1／読み出し完了時=0
    pad:     Byte;       // 予備
    cnt:     WORD;       // データByte数
    buf: array [0..259] of Byte;    // Max 260Byteバッファ
end;

TCMEM_TABLE = record     // 共有メモリテーブル
    ctrl:    TCMEM_SERIAL_PM; // シリアル通信コントロールパラメータ
    send:    TCMEM_BLOCK;     // データ送信側 転送ブロック
    recv:    TCMEM_BLOCK;     // データ受信側 転送ブロック
end;

    PCMEM_TABLE = ^TCMEM_TABLE;  // 共有メモリのポインタ
var
  Cmem:  PCMEM_TABLE;            // 共有メモリのポインタ変数
```

```
Const
    cmem_name = 'TRANS_BUF';      // 共有メモリ名
```

この　共有メモリ名は、ファイル名みたいな物で、
ホストプロセス、変換プロセスで、同じ名前の 共有メモリ名を 指定します。
今回の場合、変換プロセス側で、最初に　共有メモリを生成します。
使用する API関数：　CreateFileMapping と　MapViewOfFile を　使用します。

ホストプロセス側で使用する API関数：
　OpenFileMapping と　MapViewOfFile を　使用します。

共有メモリ内を　読み書きするには、PCMEM_TABLE型の Cmem ポインタを使用します。
例）　　Cmem.recv.hs := 1;
　　// 共有メモリ内の recvブロックの ハンドシェークフラグを
　　// 書き込み完了通知で　1 に する。

終了する時の　API関数は　CloseHandle を、使います。

今回も、Test_Comm.exe の ソースを 公開しますので、
その中の CSCmodule.pas に 共有メモリアクセスの関数が、入ってます。

ホストプロセスと 変換プロセス間の　ハンドシェーク

　送信用 データ転送ブロックと、受信用 データ転送ブロックが、独立しているので、各転送ブロックの 転送方向は一方通行です。

送信用データ転送ブロックのデータの流れは ホストプロセス -> 共有メモリ -> 変換プロセス -> RS232C/Sendライン に　なります。

受信用 データ転送ブロックのデータの流れは RS232C/Recvライン -> 変換プロセス -> 共有メモリ -> ホストプロセス に　なります。

で、ハンドシェークですが、データの送り元が、hs = 0 である事を確認して、共有メモリにデータを書き込みます。 書き込み終わった時点で hs = 1　に します。

データの受け側が、hs = 1 である事を 確認してデータを読み出します。 読み出し終わったら hs = 0 に　します。 この動作を繰り返します。

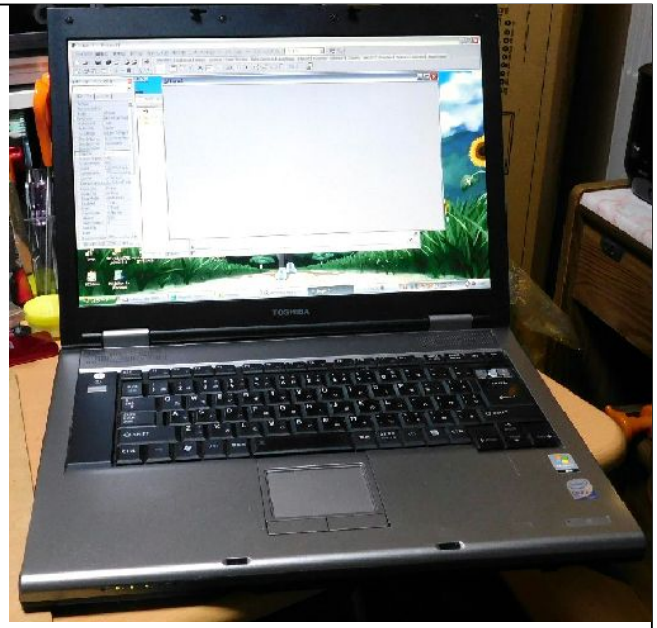
　例えば、書き込み時に hs = 1 の場合は、まだ受け側が、前のデータを 読み出して無い事になるので、hs = 0 の 待ち状態が、一時的に 発生します。相手がダウンした場合は、hsの条件待ちで、無限ループになる可能性もあります。

　本来であれば、待ちループの時間監視や、エラー処理のルールを決めておかなければなりませんが、今回は、そこまで シビアに作り込んで いません。
　という事で、今回の プログラムは、ベータ版という事にしておきます。

今回、動作確認に使用したプログラムの名前ですが

ホストプロセス側： Test_Comm.exe
変換プロセス側： CmemComm.exe

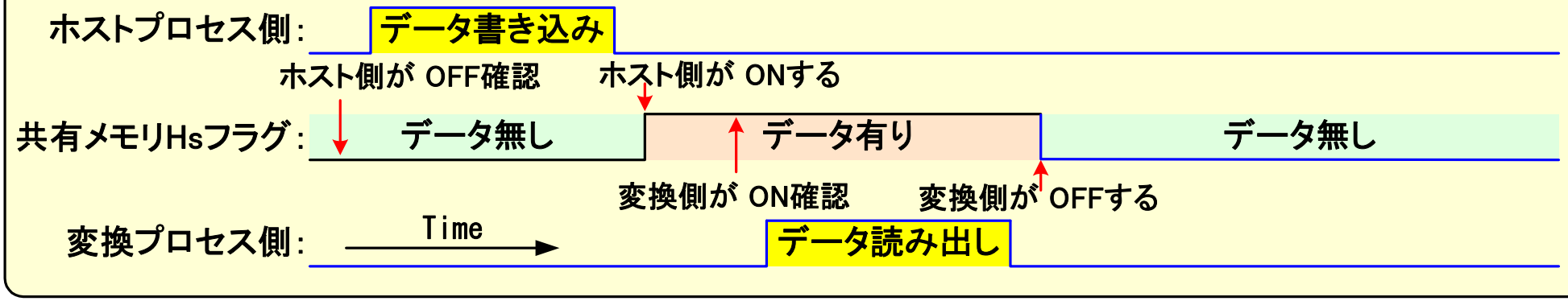
です。 最初は、右上の WindowsXPの入った いつもの
ノートPC上で Delphi5 を使って、開発を 行いました。



注意： ComPortの ComPort番号、ボーレイト、パリテイ、データ長は、ポートのオープン前に 設定して下さい。オープン後は 値を 変更しないで下さい。

DTR信号、RTS信号は オープン後でも、変更可能です。

データ書き込み、読み出し Hs信号の 変更タイミング（ ホストプロセスから 変換プロセスへ転送する場合 ）



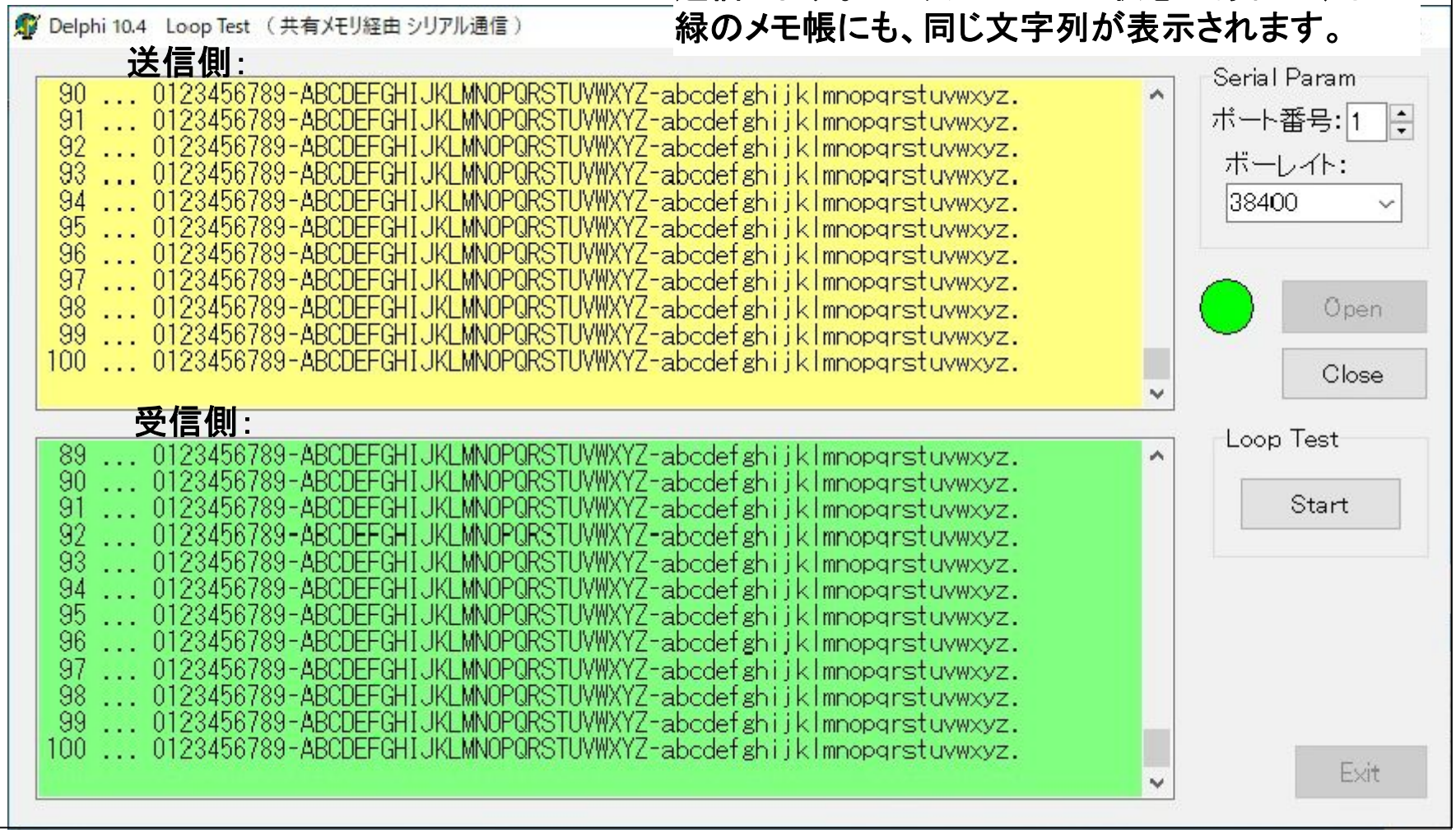
変換側 CmemComm.exeの フォーム画面

Clearは、ダンプ表示のメモ帳を両方消します。
Save SD は、送信側のダンプ表示をファイル出力します。 Save RDは、受信側ファイル出力です。

ホスト側 Test_Comm.exeの フォーム画面

Loop Test内の [Start]ボタンクリックで、行番号と数字、英字大文字、英字小文字の1行を 100回送信します。 シリアルループ状態であれば、下の緑のメモ帳にも、同じ文字列が表示されます。

RS232Cのパラメータは、ポート番号と ボーレイトの2つが、設定出来ます。 その他のパラメータは
語長： 8bit
パリテイ：無し
DTR： ON
RTS:： ON
の 固定です。
[Open] ボタンで RS232Cを オープンします。
灰色の丸が緑に変わります。
[Close]ボタンで RS232Cを クローズします。丸が灰色に戻ります。



新規Projectに CSCmoduleを 入れるには

CSCmoduleとは、共有メモリ経由で、RS232Cを アクセスする関数群を クラスにした物です。
メインフォーム側では、 CSCmd.メンバー関数; の 記述で 呼び出します。 下記に例を示します。

```
//  シリアル通信オープン
    CSCmd.rsopen( pn, bps );
//  シリアル通信 文字列送信
    CSCmd.send_string( tx );
//  シリアル通信 文字列受信
    txt := CSCmd.recv_string;
// シリアル通信クローズ
    CSCmd.rsclose;
```

で、このクラスを使うには、CSCmoduleを 新規プロジェクトに 追加する必要があります。

まず、新規プロジェクトのフォルダに、ファイルを 2本コピーします。 Delphi 10.4の プロジェクトの場合は CSCmodule.pas と CSCmodule.dfm の2本です。

Lazarusの 場合は
CSCmodule.pas と CSCmodule.lfm の2本です。

CSCmdの 関数を使うには、コピーした2本のファイルを、IDEの プロジェクトに認識させる作業が 必要になります。 それと メインの ソースファイルの implementationuses 部の下に

```
uses
    CSCmodule;
```

を、記述する必要があります。
それと、CSCmodule.pas の implementationuses 部の下に

```
uses
    メインフォームの ユニット名;
```

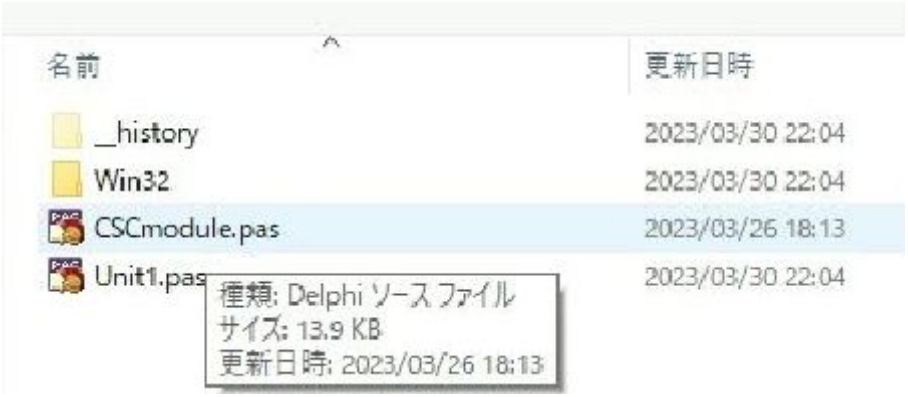
を、記述する必要があります。
次ページから、画像を使って説明します。

Delphiで 新規Projectに CSCmoduleを 入れる手順

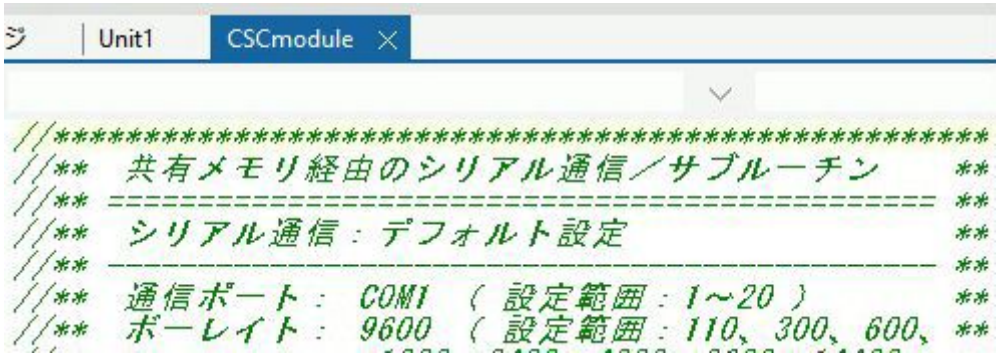
① 新規プロジェクトのフォルダに、ファイルを 2本コピーします。 コピーするファイルは、
CSCmodule.pas と CSCmodule.dfm の2本です。

② 次に Delphiのメインメニューにて、プロジェクト -> プロジェクトに追加を 選択します。

③ CSCmodule.pasを 選択し開くをクリックします。
（ CSCmodule.dfmは 一緒に付いてきます。）



④ CSCmodule.pasの ソースが 表示されます。

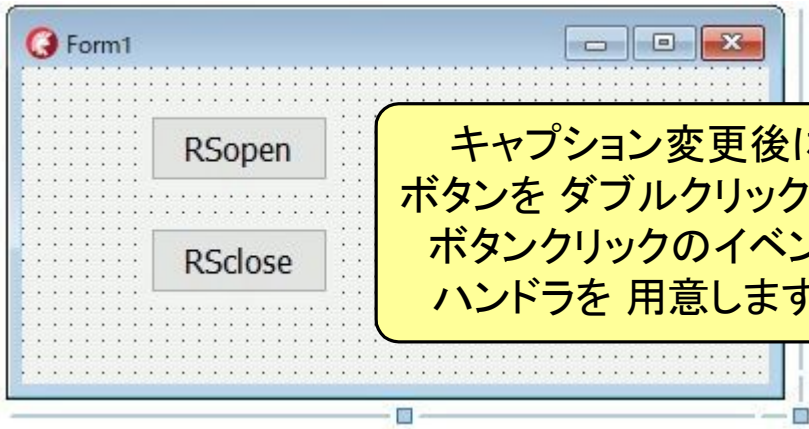


④ Unit1のコード表示に切り替えて、{$R *.dfm} の下に uses 改行 CSCmodule; を 入力します。

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private 宣言 }
  public
    { Public 宣言 }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses
    CSCmodule;
```

この2行を入力します。

⑤ Unit1のフォームに、ボタンを2個貼り付け、キャプションを、RSopen と RSclose に します。



キャプション変更後にボタンを ダブルクリックしてボタンクリックのイベントハンドラを 用意します。

⑥ 今回の例では、CSCmoduleの RS232Cの オープンと クローズの 関数呼び出しを 入れてみました。

```
uses
    CSCmodule;

// RSopen
procedure TForm1.Button1Click(Sender: TObject);
begin
    CSCmd.rsopen( 1, 38400 );    // 引数 Port番号、
end;

// RSclose
procedure TForm1.Button2Click(Sender: TObject);
begin
    CSCmd.rsclose;
end;
```

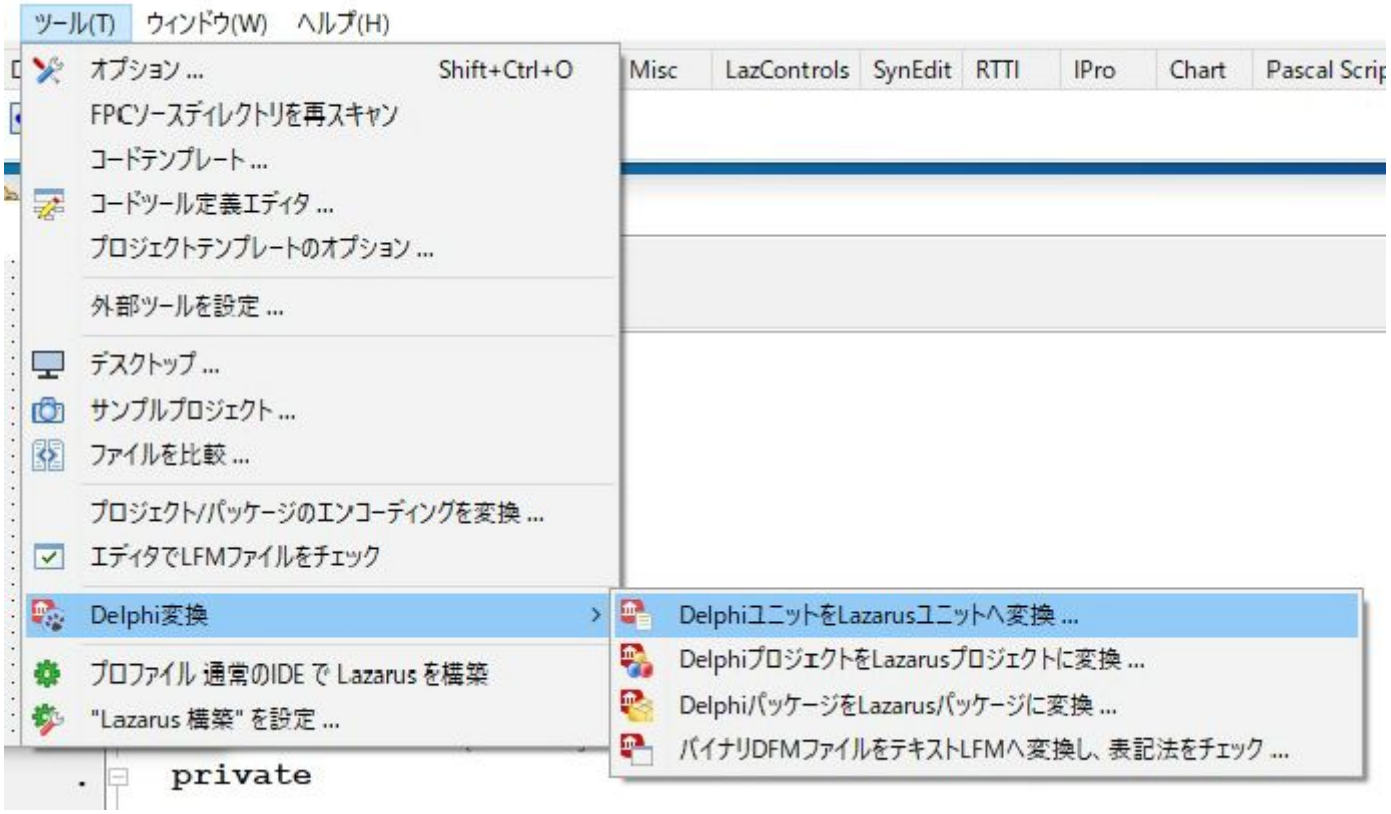
これで、Openと Closeの機能は、使えます。

⑦ これで、Delphi 10.4 の 新規プロジェクトに
CSCmoduleの 追加は出来ました。

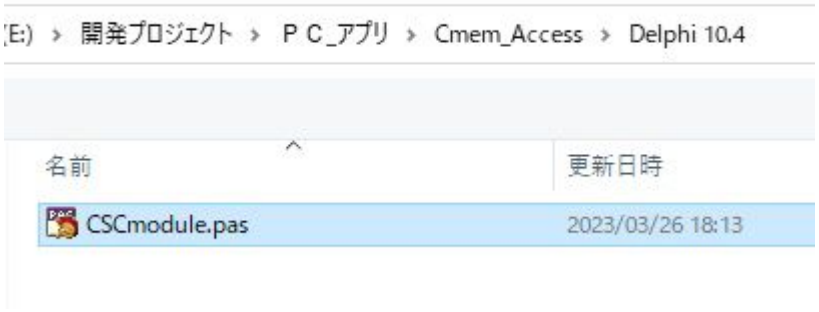
次は、Lazarus ですが、モジュールの追加は 結果
からいうと、うまく 出来ませんでした。

① ラザロの 新規プロジェクトに
CSCmoduleの 追加

メインメニューのプロジェクトに
プロジェクトに追加は、無いので
ツール の Delphi変換の
Delphi ユニットを Lazarus ユニットへ
変換 を 使用する事に
なります。

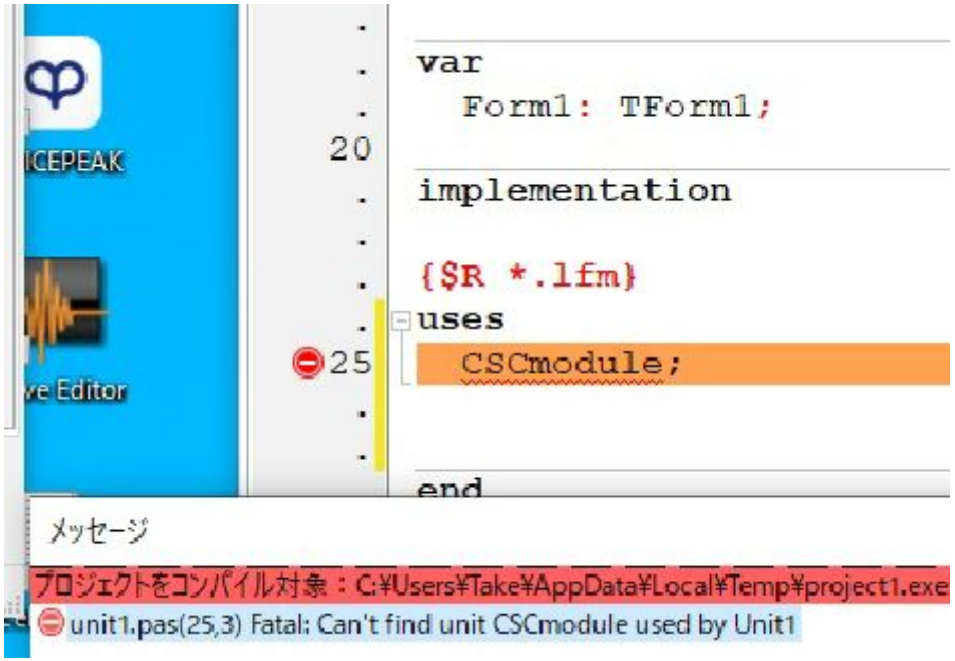


② ファイルを開くダイアログにて、CSCmodule.pasを開きます。



③ CSCmodule.pas の ソースが、表示されます。
うまく取り込めたかな。 と、思ったら
違ってました。

④ 2つのソースに uses節を 追加してビルドしたら
以下のようなエラーが、出ました。



ソースエディタ上に、ソースファイルは、取り込めたものの、CSCmoduleは、プロジェクト内の ソースとして IDEが、認識してないようです。 ここから先、全然進む事が 出来ませんでした。 プログラム的に、何とも、中途半端な取り込み機能ですよね。

その他、Delphi、Lazarusを 扱って分かった事

以前、Lazarusの 64bit版を、インストールして、Integerのサイズを見たら 4byteだったので、生成されるアプリは 32bitと判断します。 と、書いてましたがWindows 10 32bitの ノートPCで 実行したら このプログラムは、実行できません。
と、出たので 64bitの アプリの ようです。
Integerのサイズ 4byteは、過去の資産の データの互換性を取る処置なのかもしれません。

という事で、Lazarus 64bit版で 生成されるアプリは64bitだったという事です。 以前 間違った事を書いてしまい申し訳ありませんでした。

もうひとつ、生成した アプリの ファイルサイズに かなり差が、ある事にも 驚きました。
今回、作成したテストプログラム Test_Comm.exeのファイルサイズですが、

Delphi 5 で生成した Test_Comm.exe は
409,088 byte でした。

Delphi 10.4 で生成した Test_Comm.exe は
2,555,392 byte でした。

Lazarus で生成した Test_Comm.exe は
20,525,824 byte でした。

何で、こんなに差が あるのか。？
Delphi 5 の実行モジュールが 小さいのは、昔のWindows環境に合わせて作られているからと、思います。 今の、Windows 10 は 昔の Windowsに 比べ 1 Gbyteを 超える肥大化したサイズですし、Delphi 10.4 の 2,555,392 byte は 仕方ないかな。 とも思います。 が、Lazarus の 20,525,824 byte は、何なのでしょうか。？ 最初 Linux環境で開発されている事が、影響しているのでしょうか。？