

```

{*****}
{          S E R T R A N S . P A S          }
{*****}
{-----*-----}
{ Task      : File transfer via serial port }
{ with error correction                     }
{-----*-----}
{ Author    : Michael Tischer / Bruno Jennrich }
{ Developed on : 04/08/1994 }
{ Last update : 04/07/1995 }
{*****}
{$X+}                                { Function results optional }

```

Uses SERUTIL,ARGS;

```

Const
CCITT_POLYNOM = $1021;           { x^16 + x^12 + x^5 + x^0 = $11021! }
                                   { Certain ASCII codes }
ASCII_SOH      = $01;           { Start of header }
ASCII_STX      = $02;           { Start of text / data }
ASCII_EOT      = $03;           { End of transmission }
ASCII_ENQ      = $05;           { Inquiry }
ASCII_ACK      = $06;           { Positive acknowledge }
ASCII_NAK      = $15;           { Negative acknowledge }
ASCII_SYN      = $16;           { Synchronous idle }

PACKET_DATASIZE = 512;

PACKET_FILEOPEN = 0;             { Random packet types }
PACKET_FILECLOSE= 1;
PACKET_FILEDATA = 2;

                                   { Error messages }
TRANS_SUCCESS   = 0;
TRANS_TIMEOUT   = -1;
TRANS_TRANSERROR = -2;
TRANS_PROTOCOL  = -3;
TRANS_CRC       = -4;
TRANS_NOFILE    = -5;

                                   { Action codes for trans_SendPACKET() }
SEND_RECEIVER   = 0;
SEND_SENDING    = 1;
SEND_GETNAKLO   = 2;
SEND_GETNAKHI   = 3;
SEND_PUTCRCLO   = 4;
SEND_PUTCRCHI   = 5;
SEND_GETREPLY   = 6;

                                   { Action codes for trans_ReceivePACKET() }
RECEIVE_RECEIVE = 0;
RECEIVE_GETCRCLO = 1;
RECEIVE_GETCRCHI = 2;
RECEIVE_SIGNAL  = 3;
RECEIVE_PUTACK   = 4;
RECEIVE_PUTNAK   = 5;
RECEIVE_PUTNAKLO = 6;
RECEIVE_PUTNAKHI = 7;

PACKET_SUCCESS = 0;
PACKET_SEQUENCE = -10;
PACKET_UNKNOWN = -11;
PACKET_TIMEOUT = -12;

```

```

Type
PACKET = RECORD { Packets form the basis for communication }
  bType      : Byte;           { Packet type (user-defined) }
  uDataSize, : Word;           { Valid data within packet }
  uNr        : Word;           { Packet sequence number }
  bData      : Array[0..PACKET_DATASIZE - 1] of Char; { Data }
End;

```

```

var
iSerPort, iCom : Integer;
lBaud          : Longint;
Arguments      : NArgStrings;

```

```

{*****}
{ CalcCRC : Calculate cyclic redundancy checksum (CRC) }
{-----*-----}
{ Input : uCRC - previous checksum }
{        bData - byte to be incorporated }
{ Output : new checksum including passed byte }
{-----*-----}
{ Info : In the first call to this function the "previous" }

```

```

} {checksum must be set to $FFFF.}
{*****}
Function CalcCRC( uCRC : Word; bData : Byte ) : Word;

var j : Integer;

Begin
  for j := 0 to 7 do
    Begin
      uCRC := uCRC shl 1;
      if ( bData and $0080 ) <> 0 then { Modulo polynomial division }
        uCRC := uCRC xor CCITT_POLYNOM;
      bData := bData shl 1;
    End;
  CalcCRC := uCRC;
End;

{*****}
{ PACKET_Build : Initialize structure of type PACKET for sending }
{*****}
{
  Input : PACKET      - packet being initialized
        bType        - user type of packet
        pData        - address of packet data
        uDataSize    - number of valid bytes
                   in packet
        uNr          - packet number (consecutive)
}
{*****}
Procedure PACKET_Build( var PACKET      : PACKET;
                       bType          : Byte;
                       pData          : pointer;
                       uDataSize,
                       uNr            : Word );

var i          : Word;
  BufPtr      : ^char;

Begin
  BufPtr := pData; { Enter data into packet }
  PACKET.bType := bType;
  PACKET.uDataSize := uDataSize;
  PACKET.uNr := uNr;
  move( BufPtr^, PACKET.bData[0], uDataSize );
End;

{*****}
{ trans_WriteUINT : Send a single UINT via serial port }
{*****}
{
  Input : iSerPort - base port of interface
        uData      - UINT being sent
  Output : Error status
}
{*****}
{ Info : This function first sends the LO-, then the HI- }
{        byte of a UINT. Warning - no handshake! }
{*****}
Function trans_WriteUINT( iSerPort : Integer; uData : Word ) : Integer;

Begin
  trans_WriteUINT := TRANS_TRANSERROR;
  if ser_WriteByte(iSerPort, LO(uData), $8000,0,0)=SER_SUCCESS then
    if ser_WriteByte(iSerPort, HI(uData), $8000,0,0) = SER_SUCCESS then
      trans_WriteUINT := TRANS_SUCCESS;
End;

{*****}
{ trans_ReadUINT : Read a single UINT from port }
{*****}
{
  Input : iSerPort - base port of interface
        Data      - integer variable accepting the
                   received value.
  Output : Error status
}
{*****}
Function trans_ReadUINT( iSerPort : Integer;
                       var Data   : Word ) : Integer;

var bErrL, bErrH : Byte;

Begin
  trans_ReadUINT := TRANS_TRANSERROR;
  if ser_ReadByte(iSerPort, bErrL, $8000, 0, 0) = SER_SUCCESS then
    if ser_ReadByte(iSerPort, bErrH, $8000, 0, 0) = SER_SUCCESS then
      Begin
        Data := bErrH * 256 + bErrL;
        trans_ReadUINT := TRANS_SUCCESS;
      End;
End;

{*****}

```

```

{trans_WriteNak : Send negative acknowledge via port}
{-----*}
{ Input : iSerPort - base port of interface
      iCode      - extended error code
{ Output : Error status
{-----*}
Function trans_SendNak( iSerPort : Integer; iCode : Word ) : Integer;

Begin
  if ser_WriteByte( iSerPort, ASCII_NAK, $8000,0,0) = SER_SUCCESS then
    trans_SendNak := trans_WriteUINT( iSerPort, iCode )
  else
    trans_SendNak := TRANS_TRANSERROR;
End;

{-----*}
{ trans_WriteAck : Send (positive) acknowledge via port
{-----*}
{ Input : iSerPort - base port of interface
{ Output : Error status
{-----*}
Function trans_SendAck( iSerPort : Integer ) : Integer;

Begin
  if ser_WriteByte( iSerPort, ASCII_ACK, $8000,0,0) = SER_SUCCESS then
    trans_SendAck := TRANS_SUCCESS
  else
    trans_SendAck := TRANS_TRANSERROR;
End;

{-----*}
{ trans_SendPACKET : Send entire data packet via port
{-----*}
{ Input : iSerPort - base port of interface being used
      pPACKET      - address of data
      iPACKETSize - data quantity
{ Output : Error status
{-----*}
Function trans_SendPACKET( iSerPort      : Integer;
                          pPACKET      : pointer;
                          iPACKETSize : Word ) : Integer;

var uCRC, uAction,
    uTimeOut,
    uActByte      : Word;
    bData, bNakLo : Byte;           { First byte of NAK return code }
    bDataOk       : Boolean;
    bptr          : ^Byte;

Begin
  uTimeOut := $FFFF;
  uAction := SEND_RECEIVER;

  While uTimeOut <> 0 do
    Begin
      bDataOk := FALSE;           { No data available }
      if ser_IsDataAvailable( iSerPort ) then
        if ser_ReadByte( iSerPort, bData, 1, 0, 0 ) <> SER_SUCCESS then
          Begin
            trans_SendPACKET := TRANS_TRANSERROR;
            Exit;
          End
        else
          bDataOk := TRUE;           { Or is there? }

      if bDataOk then
        Begin
          case uAction of
            SEND_RECEIVER:
              if bData = ASCII_SYN then
                Begin
                  uActByte := 0;
                  uCRC := $FFFF;
                  uAction := SEND_SENDING;
                End;

              {-- NAK interrupts sending of packet! ---}
            SEND_SENDING:
              if bData = ASCII_NAK then uAction := SEND_GETNAKLO;

            SEND_GETNAKLO:
              Begin
                bNakLo := bData;
                uAction := SEND_GETNAKHI;
              End;

            SEND_GETNAKHI:

```

```

    Begin
        trans_SendPACKET := bData * 256 + bNakLo;
        Exit;
    End;

SEND_GETREPLY:
    Begin
        if bData = ASCII_ACK then
            begin
                trans_SendPACKET := TRANS_SUCCESS;
                exit; { End of finite-state machine }
            end;
        if bData = ASCII_NAK then
            begin
                uAction := SEND_GETNAKLO;
                trans_SendPACKET := TRANS_PROTOCOL;
            end;
        End;
    End;

End;
else
    if ser_IsWritingPossible( iSerPort ) then
        Begin
            case uAction of
                SEND_SENDING:
                    Begin
                        bptr := pPACKET;
                        Inc( bptr, uActByte );
                        if ser_WriteByte( iSerPort, bptr^,
                                         1, 0, 0 ) = SER_SUCCESS then

                            Begin
                                bptr := pPACKET;
                                Inc( bptr, uActByte );
                                uCRC := CalcCRC( uCRC, bptr^ );
                                Inc( uActByte );
                                if uActByte = iPACKETSize then
                                    uAction := SEND_PUTCRCLO;
                                    uTimeOut := $FFFF;
                                End
                            else
                                Begin
                                    trans_SendPACKET := TRANS_TRANSERROR;
                                    Exit;
                                End;
                            End;
                        End;

                    SEND_PUTCRCLO:
                        if ser_WriteByte( iSerPort, LO( uCRC ),
                                         1, 0, 0 ) = SER_SUCCESS then
                            uAction := SEND_PUTCRCHI
                        else
                            Begin
                                trans_SendPACKET := TRANS_TRANSERROR;
                                Exit;
                            End;

                    SEND_PUTCRCHI:
                        if ser_WriteByte( iSerPort, HI( uCRC ), 1,
                                         0, 0 ) = SER_SUCCESS then
                            uAction := SEND_GETREPLY
                        else
                            Begin
                                trans_SendPACKET := TRANS_TRANSERROR;
                                Exit;
                            End
                        else
                            { case else }
                            Dec( uTimeOut )
                        End;
                    End;
                End;
            End;
            trans_SendPACKET := TRANS_TIMEOUT;
        End;

{ ***** }
{ trans_ReceivePACKET: Receive data packet from port }
{ ***** }
{ -----*----- }
{ Input : iSerPort - base port of interface being used }
{         pPACKET - address of receive buffer }
{         iPACKETSize - buffer size }
{ Output : Error status }
{ ***** }
Function trans_ReceivePACKET( iSerPort : Integer;
                             pPACKET : Pointer;
                             iPACKETSize : Word ) : Integer;

```

```

var uCRC, uAction,

```

```

uTimeOut,
uActByte      : Word;
bData, bCrcLo : Byte;
iAckError     : Integer;
bDataOk       : Boolean;
bptr          : ^Byte;
               { First byte of CRC code }

Begin
uTimeOut := $FFFF;
uAction := RECEIVE_SIGNAL;

uCRC := $FFFF;      { Access variables at least once }
uActByte := 0;

while uTimeOut <> 0 do
  Begin
    bDataOk := FALSE;      { No data available }
    if ser_IsDataAvailable( iSerPort ) then
      if ser_ReadByte( iSerPort, bData, 1, 0, 0 ) <> SER_SUCCESS then
        Begin
          trans_ReceivePACKET := TRANS_TRANSERROR;
          Exit;
        End
      else
        bDataOk := TRUE;      { Or is there? }

    if bDataOk then
      Begin
        case uAction of
          RECEIVE_RECEIVE:
            Begin
              uCRC := CalcCRC( uCRC, bData );
              bptr := pPACKET;
              Inc( bptr, uActByte );
              bptr^ := bData;
              inc( uActByte );
              if uActByte = iPACKETSize then
                uAction := RECEIVE_GETCRCLO;
                uTimeOut := $FFFF;
              End;

          RECEIVE_GETCRCLO:
            Begin
              bCrcLo := bData;
              uAction := RECEIVE_GETCRCHI;
            end;

          RECEIVE_GETCRCHI:
            begin
              if uCRC = bData * 256 + bCrcLo then
                begin
                  uAction := RECEIVE_PUTACK;
                end
              else
                Begin
                  iAckError := TRANS_CRC;
                  uAction := RECEIVE_PUTNAK;
                End;
            end;
          End;
        else
          if ser_IsWritingPossible( iSerPort ) then
            Begin
              case uAction of
                RECEIVE_SIGNAL:
                  begin
                    if ser_WriteByte( iSerPort, ASCII_SYN,
                                      1, 0, 0 ) = SER_SUCCESS then
                      Begin
                        uCRC := $FFFF;
                        uActByte := 0;
                        uAction := RECEIVE_RECEIVE;
                      End
                    else
                      Begin
                        trans_ReceivePACKET := TRANS_TRANSERROR;
                        Exit;
                      End;
                    end;
                  RECEIVE_PUTACK:
                    Begin
                      if ser_WriteByte( iSerPort, ASCII_ACK,
                                        1, 0, 0 ) = SER_SUCCESS then
                        trans_ReceivePACKET := TRANS_SUCCESS
                      else

```

```

        trans_ReceivePACKET := TRANS_TRANSERROR;
        Exit; { End of finite-state machine }
    End;

RECEIVE_PUTNAK:
begin
    if ser_WriteByte( iSerPort, ASCII_NAK,
        1, 0, 0 ) = SER_SUCCESS then
        uAction := RECEIVE_PUTNAKLO
    else
        begin
            trans_ReceivePACKET := TRANS_TRANSERROR;
            Exit;
        end;
    end;
RECEIVE_PUTNAKLO:
begin
    if ser_WriteByte( iSerPort, LO( iAckError ),
        1, 0, 0 ) = SER_SUCCESS then
        uAction := RECEIVE_PUTNAKHI
    else
        Begin
            trans_ReceivePACKET := TRANS_TRANSERROR;
            Exit;
        End;
    end;
RECEIVE_PUTNAKHI:
    Begin
        if ser_WriteByte( iSerPort, HI( iAckError ),
            1, 0, 0 ) = SER_SUCCESS then
            trans_ReceivePACKET := iAckError
        else
            begin
                trans_ReceivePACKET := TRANS_TRANSERROR;
                Exit;
            end;
        End;
    else { case else }
        Dec( uTimeOut );
    End;
End;

End;
trans_ReceivePACKET := TRANS_TIMEOUT;
End;

{ ***** }
{ trans_ReceiveFile : Receive entire file via port }
{ ***** }
{ Input : iSerPort - base port of interface being used }
{ Output : Error status }
{ ***** }
Function trans_ReceiveFile( iSerPort : Integer ) : Integer;

var Pak      : PACKET;
    iTimeOut,
    i,
    ior,err  : Integer;
    fHandle  : File;
    w        : Word;
    DatName  : string;

Const uNr    : Word = 0;

Begin
    iTimeOut := 10;          { Wait a bit longer for first packet }
    while iTimeOut <> 0 do
        Begin
            err := trans_ReceivePACKET( iSerPort,          { Receive packet }
                @Pak,
                sizeof( PACKET ) );
            if err = TRANS_SUCCESS then
                Begin
                    iTimeOut := 4;                          { Shorten timeout }
                    if uNr <> Pak.uNr then
                        Begin
                            Close( fHandle );
                            trans_SendNak( iSerPort, Word ( PACKET_SEQUENCE ) );
                            trans_ReceiveFile := PACKET_SEQUENCE;
                            Exit;
                        End
                    else
                        Begin
                            Inc( uNr );
                            Write( #13, uNr );
                            case Pak.bType of
                                PACKET_FILEOPEN:

```

```

Begin
    DatName := '';
    for i := 0 to Pak.uDataSize do
        DatName := DatName + Pak.bData[i];

    {$i-}
        Assign( fhandle, DatName );
        Rewrite( fhandle, 1 );
    {$i+}

    ior := ioresult;
    if ior <> 0 then
        Begin
            trans_SendNak( iSerPort, ior );
            trans_ReceiveFile := ior;
            Exit;
        End;
    End;

PACKET_FILEDATA:
    Begin
        {$i-}
            Blockwrite( fhandle, Pak.bData, Pak.uDataSize, w );
        {$i+}
        if w <> Pak.uDataSize then
            Begin
                ior := ioresult;
                trans_SendNak( iSerPort, ior );
                Close( fHandle );
                trans_ReceiveFile := ior;
                Exit;
            End;
        End;

PACKET_FILECLOSE:
    Begin
        Close( fHandle );
        trans_ReceiveFile := PACKET_SUCCESS;
        Exit;
    End

    else { case else }
        Begin
            trans_SendNak( iSerPort, Word ( PACKET_UNKNOWN ) );
            trans_ReceiveFile := PACKET_UNKNOWN;
            Exit;
        End;
    End;
End;
    Dec( iTimeOut );
End;
trans_ReceiveFile := PACKET_TIMEOUT;
End;

{*****}
{ trans_SendFile : Send entire file via port }
{*****}
{-----*}
{ Input : iSerPort - base port of interface being used }
{ Name - name of file being sent }
{ Output : Error status }
{*****}
Function trans_SendFile( iSerPort : Integer; Name : String ) : Integer;

var fHandle : File;
    Pak : PACKET;
    iSize,
    err,
    iCnt,
    iLen : Integer;
    n : string;
Const uNr : Word = 0;

Begin
    {$i-}
        Assign( fHandle, Name );
        Reset( fhandle, 1 );
    {$i+}
    if ioresult = 0 then
        Begin
            iLen := Length( Name );
            while ( ( iLen > 0 ) and ( Name[iLen] <> '\' )
                and ( Name[iLen] <> ':' ) ) do
                Dec( iLen );

```

```

PACKET_Build( Pak, PACKET_FILEOPEN, @Name[iLen+1],
              Length( Name ) - iLen, uNr );
Inc( uNr );

iCnt := 25;
Write(#13, uNr );
Repeat
  Dec( iCnt );
  err := trans_SendPACKET( iSerPort, @Pak, sizeof( PACKET ) );
Until ( ( err <> TRANS_TIMEOUT ) or ( iCnt = 0 ) );

if err <> TRANS_SUCCESS then
  Begin
    Close( fHandle );
    trans_SendFile := err;
    Exit;
  End;

Repeat
  BlockRead( fHandle, Pak.bData, PACKET_DATASIZE, iSize );
  if iSize <> 0 then
    Begin
      PACKET_Build( Pak, PACKET_FILEDATA, @Pak.bData, iSize, uNr );
      Inc( uNr );
      iCnt := 10;
      Write(#13, uNr );
      Repeat
        Dec( iCnt );
        err := trans_SendPACKET( iSerPort, @Pak, sizeof( PACKET ) );
      Until ( ( err <> TRANS_TIMEOUT ) or ( iCnt = 0 ) );
      if err <> TRANS_SUCCESS then
        Begin
          Close( fHandle );
          trans_SendFile := err;
          Exit;
        End;
      End;
    End;
  Until ( iSize <> PACKET_DATASIZE );

  Close( fHandle );
  PACKET_Build( Pak, PACKET_FILECLOSE, @Name[iLen],
                Length( Name ) - iLen, uNr );
  Inc( uNr );

  iCnt := 10;
  Write(#13, uNr );
  Repeat
    Dec( iCnt );
    err := trans_SendPACKET( iSerPort, @Pak, sizeof( PACKET ) );
  Until ( ( err <> TRANS_TIMEOUT ) or ( iCnt = 0 ) );
  trans_SendFile := err;
  Exit;
End;
trans_SendFile := TRANS_NOFILE;
End;

```

```

{*****}
{ trans_PError : Output error message }
{*****}
{-----*}
{ Input : iError - error that occurred }
{*****}

```

```

Procedure trans_PError( iError : Integer );

```

```

Begin
  case iError of
    TRANS_SUCCESS:
      Writeln( 'Transmission OK!' );
    TRANS_TIMEOUT:
      Writeln( 'ERROR: Byte timeout' );
    TRANS_TRANSERROR:
      Writeln( 'ERROR: General error' );
    TRANS_PROTOCOL:
      Writeln( 'ERROR: Ack or NACK missing!' );
    TRANS_CRC:
      Writeln( 'ERROR: CRC error!' );
    TRANS_NOFILE:
      Writeln( 'ERROR: Cannot open file!' );
    PACKET_SEQUENCE:
      Writeln( 'ERROR: Invalid packet sequence!' );
    PACKET_UNKNOWN:
      Writeln( 'ERROR: Unknown packet ID!' );
    PACKET_TIMEOUT:
      Writeln( 'ERROR: Packet timeout' );
  End;
End;

```



```

{*****}
{ M A I N   P R O G R A M }
{*****}
Begin
if GetArg( '?', _none, NIL, 1 ) <> 0 then
  Begin
    Writeln( 'Call:' );
    Writeln( 'SERTRANS [SendeDatei] [-COM:port] [-BAUD:bdrate]' );
    Writeln( 'COM port = 1 or 2 (Default: 1)' );
    Writeln( 'Baud rate = 50 - 115200 (Default: 9600)' );
    Halt( 0 );
  End;

if GetArg( '-COM:', _int, @iCom, 1 ) <> 0 then
  Begin
    if iCom = 1 then { Only COM1 and COM2 are "standardized" }
      iSerPort := SER_COM1
    else
      if iCom = 2 then { Only COM1 and COM2 are "standardized" }
        iSerPort := SER_COM2
      else
        Begin
          Writeln( 'Unsupported COM port!' );
          Halt( 0 );
        End;
      End
    else { Only COM1 and COM2 are "standardized" }
      iSerPort := SER_COM1;

if GetArg( '-BAUD:', _long, @lBaud, 1 ) = 0 then
  lBaud := 9600; { Maximum baud rate for UART 8450A }
if lBaud > SER_MAXBAUD then
  Begin
    Writeln( 'Baud rate too high!' );
    Writeln( 'Maximum: ', SER_MAXBAUD, ' Bd' );
  End;

ser_FIFOLevel( iSerPort, 0 ); { Disable FIFO }
if ser_Init( iSerPort, lBaud, { Maximum for UART 8450A }
  SER_LCR_8BITS or
  SER_LCR_1STOPBIT or
  SER_LCR_NOPARITY ) = 0 then
  Begin
    Writeln( 'No port!' );
    Halt( 0 );
  End;

if GetNArg( '-', Arguments ) <> 0 then
  Begin
    Writeln( 'Sending' );
    trans_PError( trans_SendFile( iSerPort, Arguments[0] ) );
  End
else
  Begin
    Writeln( 'Receiving' );
    trans_PError( trans_ReceiveFile( iSerPort ) );
  End;
End.

```