Column #89 September 2002 by Jon Williams:

Data Exchange with Visual BASIC

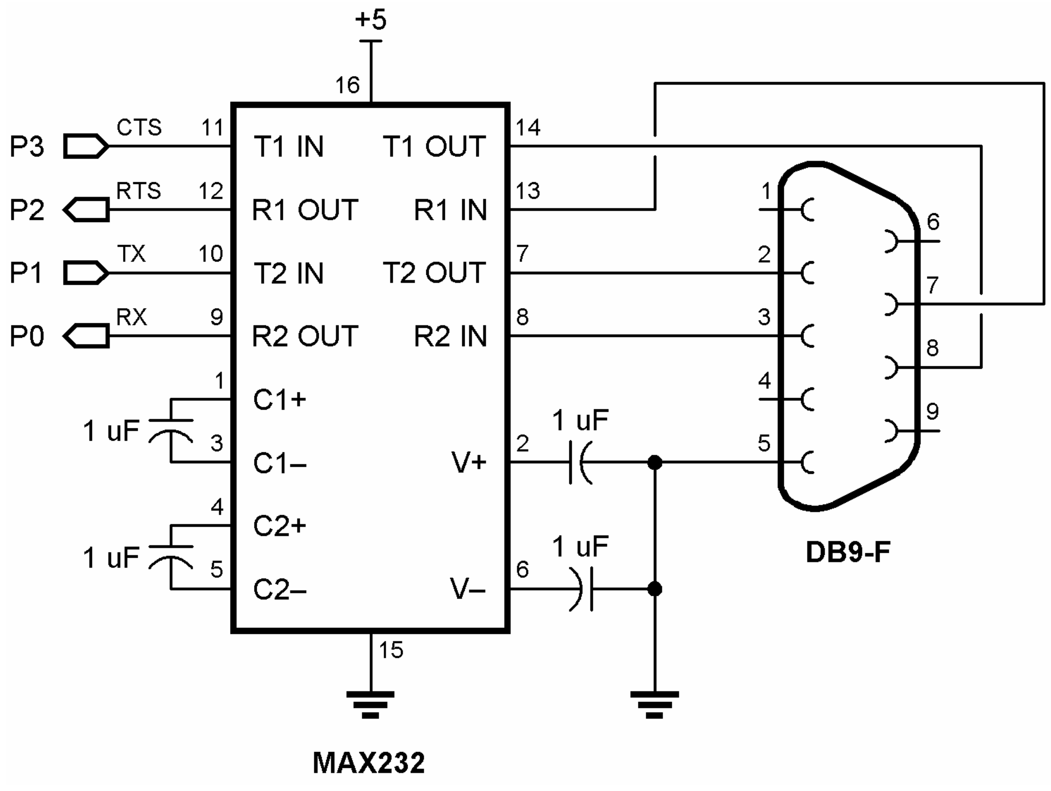
Last month we created a little self-contained digital data recorder with the BASIC Stamp that used ***DEBUG*** *to provide a user interface. Well, what if we want to get a bit more fancy and, say, create a nice GUI application with Visual BASIC? The trick is to allow the Stamp to do its thing until our VB application wants to send or receive some data. Okay ... how do we do it?*

In past Stamp-to-VB projects we've used the Stamp's programming port because it's convenient. Convenience has it's price, though. When using the programming port to communicate with other applications, we have two issues to contend with: 1) The programming port echoes everything sent to it so we have to filter that out of our application's receive stream and, 2) There is no flow control. Without flow control and we have a busy Stamp program like the data logger, our PC application would be forced to send some kind of query character until the Stamp had time to look for it, catch it, and respond to indicate it's now ready for an exchange. This can be done, but it's tedious at best and the timing is always problematic.

Problems Solved

We can solve these problems with a bit of hardware and a few Stamp pins. The hardware is a standard MAX232 serial interface chip, a handful of capacitors and a DB9-F connector (see Figure 88.1). The MAX232 is a level-shifter. It converts the RS-232 level signals from our PC to the TTL levels required by the Stamp – and goes the other way too.

Figure 89.1: MAX232 Circuit for PC-Stamp Communication

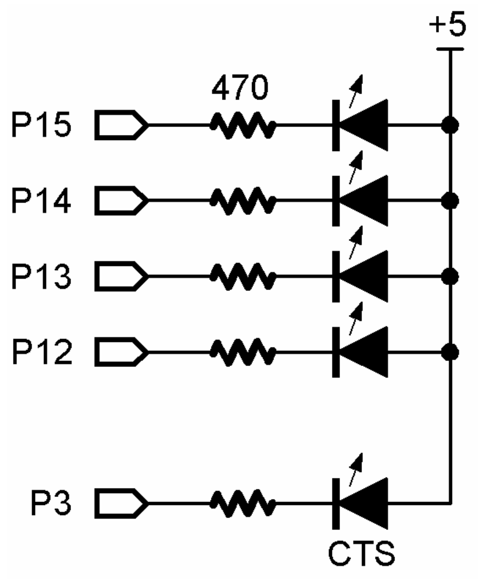


The capacitors are used by its charge-pump to create the negative voltages required by they RS-232 standard. Using proper RS-232 levels is especially important when there is a lot of distance between devices.

You can assemble the circuit on a breadboard, but to make things easy, my Texas buddy Al Williams created a neat little kit called the RS-I. This is a breadboard-friendly device that just takes a few minutes to assemble with your soldering iron.

When using the MAX232, we're using separate transmit and receive pins so there is no longer a problem with data being echoed back to our VB application. Our next challenge is flow control.

Figure 89.2: LED Monitoring Circuit



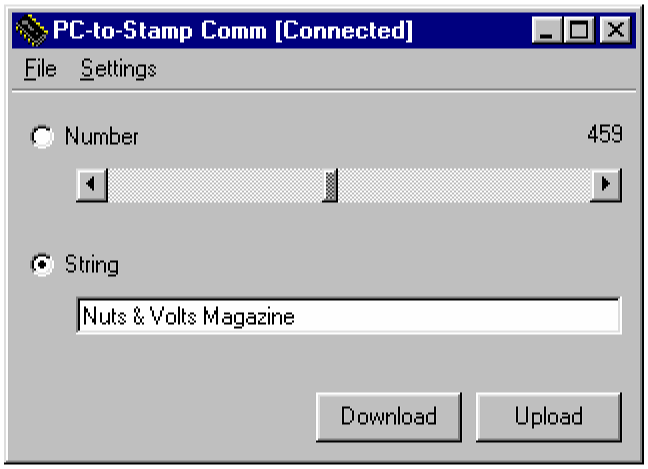
Well, as you can see in the schematic, the MAX232 has enough drivers to handle a couple flow control pins as well.

Going With The Flow

Long before the BASIC Stamp, engineers encountered the issue of two devices wanting to "talk" to each other, but one or the other not being ready. One of the solutions developed – the one we'll use here – is called Hardware Flow Control. The concept is actually pretty simple. When one device wants to talk to another, it sends a "request" signal then waits for the "all clear" from the other device. Once the "all clear" signal is detected, the data can be sent. In our setup, the PC will signal its desire to send data with the RTS (Request To Send) line. When the Stamp is ready it will indicate this with the CTS (Clear To Send) line. At this point, the PC will start transmitting data.

Let's make it work, shall we? Wire up the circuit in Figure 89.2 so you can watch the Stamp code at work and the activity of the CTS line.

Figure 89.3: Visual BASIC BASIC Stamp Example



A Simple Plan

In my book, simple is better and generally more reliable, so we're going to handle this PC-to-Stamp data exchange in the simplest possible manner. When it comes right down to it, there's really only two things we can exchange: numbers and text. With that in mind, let's build a little VB app that will allow us to exchange numbers and text.

Author's Note: This column is about BASIC Stamps, not Visual BASIC so I am going to skip the contruction details of the VB app and just focus on important aspecst of the code.

Figure 89.3 shows a simple VB app that lets us handle the exchange from the PC end. Numbers are selected with a scrollbar, text is entered into a box. Simple. Now let's go under the hood.

To perform serial communications with Visual BASIC we have add the MSComm control to our project. For the settings critical to operation, my preference is to set them in code. for this project, here's what we're going to do (in the Form_Initialize() event handler):

```
With MSComm1
  .Settings = "9600,N,8,1"
  .DTREnable = False
  .Handshaking = comRTS                        ' use hardware flow control
  .RTSEnable = True
```

```
    .InputMode = comInputModeText
  End With
```

Notice that we've selected hardware handshaking via the RTS pin and enabled the RTS line so we can use it. What this means is that when we put data into the output buffer of the MSComm control, it will exert the RTS line and send it when it detects the CTS signal.

Before we get too far ahead of ourselves, let's define our data exchange strategy. The PC will initiate the exchange by sending a command by that tells the Stamp what it wants to do: 1) Send a number to the Stamp, 2) Retrieve a number from the Stamp, 3) Send a string of characters to the Stamp or, 4) Retrieve a string of characters from the Stamp.

Now don't get the idea that with flow control we can simply stuff the MSComm output buffer full of data and let the handshaking handle it – the unbuffered Stamp still needs us to help it a bit, especially since we don't know what's coming after the command byte until we've looked at it.

From the PC side, we'll send the command and make sure it's gone before sending anything else. What this does is let the Stamp receive the command and get ready without us having to resort to padding the PC program with artificial delays. Here's how it's done:

```
Private Sub FlushTxBuf()
  Do
    DoEvents
  Loop Until (MSComm1.OutBufferCount = 0)
End Sub
```

Using this strategy, the Stamp will be able to receive the command, then remove the CTS signal while it figures out what to do. Before we start sending information, let's examine the Stamp side of things to see how command reception works.

The Stamp program is constructed in a manner that allow us to break it up into small, discrete sections and to check for a serial input in between them – the task switcher design we've used many times before. This allows to check the serial input fairly frequently so we don't keep the PC waiting.

The bulk of our main code looks like this:

```
Main:
  SERIN RX\CTS, Baud, 5, Do_Task, [cmd]
  LOOKDOWN cmd, [RxNum, TxNum, RxStr, TxStr], cmd
  BRANCH cmd, [RX_Number, TX_Number, RX_String, TX_String]

Do_Task:
  BRANCH task, [Task_0, Task_1, Task_2, Task_3]
  task = 0                                        ' fix bad task spec
  GOTO Main

Task_0:
  LEDs = %1110                                    ' show current task
  PAUSE 250                                       ' take some time doing it
  LEDs = Off
  task = task + 1 // NumTasks                     ' point to next task
  GOTO Main
```

Notice that the **SERIN** code specifies the [defined] CTS pin for flow control and it really doesn't wait around long; only five milliseconds. Even at 9600 baud, this is plenty of time for the PC to detect the CTS signal and send the command. If a command is received, it is decoded with a **LOOKDOWN** table and **BRANCH** is used to call the code for the command that was sent. If there is no command or it's bad, the code finds its way to Do_Task and the Stamp runs the next chunk of task code.

For our little test program, each task lights a given LED. When the program is running, you'll see the LEDs light sequentially with a quick blip of the CTS LED in between.

Back to our VB app. Let's say we want to send a number to the Stamp. We'll select the Number radio button, move the slider to the value we want, then click the Download button. This will send the command, then the number, low byte first. I decided to use 16-bit variables because that is the largest single value that the Stamp can handle. If you create an application that only needs to send bytes, the process is simpler ... there's no need to extract individual bytes from the larger value.

```
    ' send number command
    MSComm1.Output = Chr$(CmdTxN)
    Call FlushTxBuf
    ' send word value; low byte first
    MSComm1.Output = Chr$(scrValue.Value Mod 256)_
                   + Chr$(scrValue.Value \ 256)
    Call FlushTxBuf
```

Notice that the MSComm transmit buffer is flushed after the number as well. What this does make sure the Stamp has had a change to receive the value before we send it anything else.

The reception side is equally straightforward:

```
RX_Number:
  SERIN RX\CTS, Baud, [aNumber.LowByte, aNumber.HighByte]
  GOTO Do_Task
```

As soon as the number comes in we get back to work by jumping to **Do_Task**. When the next task is complete we'll be back at the top and looking for another command.

While we're focused on this process, let me point something out. Did you notice how we put two bytes into the MSComm output buffer and the Stamp **SERIN** command was constructed to accept two bytes? Follow this rule: Only put as many bytes in the to MSComm output buffer as the [active] **SERIN** command is setup to handle. As I told you earlier, this will allow the Stamp to receive the data without problems and then remove the CTS while it's working with what it just received.

Receiving a number from the Stamp is not complicated at all: We send the command then wait for two bytes to show up in the MSComm input buffer.

```
    ' send command
    MSComm1.Output = Chr$(CmdRxN)
    Call FlushTxBuf
    ' wait for two-byte value
    Do
      DoEvents
    Loop Until (MSComm1.InBufferCount = 2)
    ' grab input buffer
    rxBuf = MSComm1.Input
    ' display it
    scrValue.Value = Asc(Mid$(rxBuf, 1, 1)) + (Asc(Mid$(rxBuf, 2, 1)) * 256)
    ' update scroller
    Call scrValue_Change
```

After the data comes in, we extract it from the string buffer with VB's Asc function and send the value to the scoller and label. Easy, right? You betcha.

The other kind of data that we might want to exchange between the PC and Stamp is text or strings. While working for my previous employer, I designed a Stamp-based alarm device that monitored four independent channels. Within the data structure for the alarm system, the site location name and labels for each channel were stored.

As you know, the Stamp has no string type and precious little RAM, so we've got to construct a strategy around these limits. Don't worry, it's not tough. What were going to do is send the string one byte at a time. This will allow the Stamp to receive the byte and store it in its EEPROM. We'll tell the Stamp that we've finished by sending a zero. Some would suggest using a CR as the terminator, but using a zero allows us to embed carriage returns in our string.

Here's the PC side of sending a string:

```
    ' send string command
    MSComm1.Output = Chr$(CmdTxS)
    Call FlushTxBuf
    ' send one character at a time
    For chrPos = 1 To Len(txtStringData.Text)
      MSComm1.Output = Mid$(txtStringData.Text, chrPos, 1)
      Call FlushTxBuf
    Next
    ' send terminating character
    MSComm1.Output = Chr$(0)
    Call FlushTxBuf
```

By now this should be fairly self-evident. We send the command and flush the buffer to let the Stamp get ready to receive the string. Then, one character at time, we send the string. Finally, we send a zero to complete the process.

The Stamp side is easy too, but a little more involved since it has to accept a character then examine to see if anything else is coming from the PC or not.

```
RX_String:
  idx = 0                                       ' reset address pointer

RX_Char:
  SERIN RX\CTS, Baud, [dByte]                   ' receive char from PC
  WRITE (Msg + idx), dByte                      ' save it to EEPROM
  IF (dByte = 0) THEN RX_Str_Done               ' wait for another if not 0
  idx = idx + 1                                 ' update address pointer
  GOTO RX_Char

RX_Str_Done
  WRITE StrLen, idx                             ' save string length
  GOTO Do_Task
```

The variable called idx is going to be used to keep track of the length of the string that comes in. We'll see why we need this in just a moment. The core of the code is at RX_Char. A byte is taken in, written to the Stamp's EEPROM and, if not zero, the idx variable is updated. Once a zero is received, the string length is written to the EEPROM and the program resumes at the next task.

Sending a string from the Stamp to the PC is a two step process. First, we'll send the string length (now you know why we saved it) so that the PC knows how many characters to look for in the input buffer. We then wait for a second "send string" command before sending the actual string data.

```
TX_String:
  READ StrLen, idx                              ' get string length
  SEROUT TX\RTS, Baud, [idx]                    ' send string length
  SERIN RX\CTS, Baud, 100, Do_Task, [cmd]        ' wait for restart
  IF (cmd <> TxStr) THEN Do_Task                 ' abort if bad command
  idx = Msg

TX_Char:
  READ idx, dByte                                ' get char from EEPROM
  IF (dByte = 0) THEN TX_Str_Done                ' check for end
  SEROUT TX\RTS, Baud, [dByte]                   ' send char
  idx = idx + 1                                  ' point to next
  GOTO TX_Char

TX_Str_Done:
  GOTO Do_Task
```

Notice that we do give the PC just a bit of time to send the second string command. If that command never comes, we abort the process and go back to running tasks. Once we do get the command, idx is used as a pointer to characters in the string. We loop through and transmit each character to the PC until we find a zero.

Other than dealing with visual controls, the PC side matches up identically:

```
    ' clear text box
    txtStringData.Text = ""
    ' send string command
    MSComm1.Output = Chr$(CmdRxS)
    Call FlushTxBuf
    ' wait for string length
    Do
      DoEvents
    Loop Until (MSComm1.InBufferCount = 1)
    ' extract string length
    rxBuf = MSComm1.Input
    strLen = Asc(Mid$(rxBuf, 1))
    ' resend command to start upload
    MSComm1.Output = Chr$(CmdRxS)
    Call FlushTxBuf
    ' wait for string
    Do
      DoEvents
    Loop Until (MSComm1.InBufferCount = strLen)
    ' show it
    txtStringData.Text = MSComm1.Input
```

The code listing should make it pretty clear. We send the command, wait for the length, resend the command to start the actual string upload and then wait for the proper number of characters to show up. Once they do, we move them to the text field for display.

Okay, that wasn't bad, was it? Keep in mind that there are many ways to handle the process we've described here; this just happens to be a very simple method. It is particularly suited to those times when the PC program is requesting specific information from the Stamp, or wants to send specific data to it. Experience VB programmers will have noticed that we didn't bother with the OnComm event handler.

OnComm is useful when the Stamp is spontaneously sending data or there are a lot of other things going on with the PC while data is being sent. That wasn't the case with this project.

If you end up creating a project where the Stamp and the PC are always sending the same kind of information back and forth, you can simplify things by creating a fixed-length protocol. The first byte would be the command/identifier and the other bytes would be the data. By doing this, you can have one **SERIN** line to take in the data, then deal with it as dictated by the first [identifier] byte. On the PC side, OnComm can handle the data reception in the background and deal with the data based on the identifier.

Data Collection For Non PC Programmers

There's probably more than a few of you who want to collect and analyze Stamp data but don't have the skills or choose to do custom programming in Visual BASIC. Well, you're in luck. Long-time user, teacher, guru and all-around nice guy, Marty Hebel, has created a neat little product for Parallax called StampDAQ. StampDAQ is a special macro embedded in a Microsoft Excel spreadsheet that allows the spreadsheet to collect data from the Stamp. Once it's collected, you can use Excel for analysis and display with the various tools available in Excel.

Marty has created a special control that deals with the peculiar aspects of communicating via the Stamp's programming port, and has even created a simple language that the Stamp uses to control the spreadsheet. For those of you who have used StampPlot Lite or Pro, it works very similarly – StampDAQ comes from the same creator.

Best of all ... it's FREE. You can download StampDAQ from the Parallax web site. You'll need Microsoft Excel 2000 or later to run it. Sorry, but it won't run in Excel97 ... yet. I don't think Marty has given up on making that work, but there are no promises that it ever will.

A Different Kind Of Experiment

Larry (the editor of this great magazine) and I had a conversation a few days back and concluded that nobody – well, almost nobody – actually types in the listings from the text, especially since they can be downloaded from the Nuts & Volts web site. With that in mind, this month we've tried a different approach to the column: we've only printed the portions of the listings that the text focuses on.

We create this column for you, the Stamp user. Please let us know how you like this new style and share any comments that we can use to improve it.

Have fun with your Stamp-to-PC experiments. Oh ... one last thought. If you only have one serial port (like me), you can easily add a second with the BAFO USB to Serial converter. It works great with the Stamp and is generally easy to find. If you can't find it locally, you can order directly from Parallax. Having a second serial port sure makes serial experiments easy to troubleshoot since one port can be used by your VB app and the other one by the Stamp to send messages and information to its **DEBUG** window.

Oh, oh, oh ... one more thing before I sign off. For those of you who read the article and are thinking to yourself, "Geez, I wish PBASIC had IF-THEN-ELSE and DO-WHILE loops and all that neat control structure stuff...." Good news! It's coming! Soon! Very soon.

Until next time, Happy Stamping.

Al Williams RS-I
http://www.al-williams.com/awce/rs1.htm

```
' ===============================================================================
'
'   Program Listing 89.1
'   File...... PC2Stamp.vbp
'   Purpose... Serial Comm with BASIC Stamp using Flow Control
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallaxinc.com
'   Started... 01 AUG 2002
'   Updated... 02 AUG 2002
'
' ===============================================================================

Option Explicit

Dim txBuf As String
Dim rxBuf As String

Const CmdTxN = &HC0                             ' send number to Stamp
Const CmdRxN = &HC1                             ' get number from Stamp
Const CmdTxS = &HC2                             ' send string to Stamp
Const CmdRxS = &HC3                             ' get string from Stamp

Private Sub cmdDnload_Click()
  Dim chrPos As Integer

  ' stop other transfers until this one complete
  Call DisableButtons

  If optDataType(0).Value = True Then
    ' send number command
    MSComm1.Output = Chr$(CmdTxN)
    Call FlushTxBuf
    ' send word value; low byte first
    MSComm1.Output = Chr$(scrValue.Value Mod 256) + Chr$(scrValue.Value \ 256)
    Call FlushTxBuf
  Else
    ' send string command
    MSComm1.Output = Chr$(CmdTxS)
    Call FlushTxBuf
    ' send one character at a time
    For chrPos = 1 To Len(txtStringData.Text)
      MSComm1.Output = Mid$(txtStringData.Text, chrPos, 1)
      Call FlushTxBuf
    Next
    ' send terminating character
    MSComm1.Output = Chr$(0)
    Call FlushTxBuf
  End If

  ' allow transfers
  Call EnableButtons
End Sub

Private Sub cmdUpload_Click()
  Dim strLen As Integer

  ' make sure input buffer is clear
  rxBuf = MSComm1.Input

  ' stop other transfers until this one complete
  Call DisableButtons

  If optDataType(0).Value = True Then
    ' send command
    MSComm1.Output = Chr$(CmdRxN)
    Call FlushTxBuf
    ' wait for two-byte value
    Do
      DoEvents
    Loop Until (MSComm1.InBufferCount = 2)
    ' grab input buffer
    rxBuf = MSComm1.Input
    ' display it
    scrValue.Value = Asc(Mid$(rxBuf, 1, 1)) + (Asc(Mid$(rxBuf, 2, 1)) * 256)
    ' update scroller
    Call scrValue_Change
  Else
    ' clear text box
    txtStringData.Text = ""
    ' send string command
    MSComm1.Output = Chr$(CmdRxS)
    Call FlushTxBuf
    ' wait for string length
    Do
      DoEvents
    Loop Until (MSComm1.InBufferCount = 1)
    ' extract string length
    rxBuf = MSComm1.Input
    strLen = Asc(Mid$(rxBuf, 1))
    ' resend command to start upload
    MSComm1.Output = Chr$(CmdRxS)
    Call FlushTxBuf
    ' wait for string
    Do
      DoEvents
    Loop Until (MSComm1.InBufferCount = strLen)
    ' show it
    txtStringData.Text = MSComm1.Input
  End If

  ' allow transfers
  Call EnableButtons
End Sub

Private Sub Form_Initialize()
  ' initialize controls
  Call mnuSettingsComm1_Click
  optDataType(0).Value = True
  optDataType(1).Value = False
  Call scrValue_Change
  txtStringData.Text = ""

  With MSComm1
    .Settings = "9600,N,8,1"
    .DTREnable = False
    .Handshaking = comRTS                   ' use hardware flow control
    .RTSEnable = True
    .InputMode = comInputModeText
  End With
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode As Integer)
  ' don't quit in middle of a transfer
  Cancel = Not (mnuFileExit.Enabled)
End Sub

Private Sub Form_Unload(Cancel As Integer)
  If MSComm1.PortOpen Then
    MSComm1.PortOpen = False
  End If
End Sub

Private Sub mnuFileExit_Click()
  Unload Me
End Sub

Private Sub mnuSettingsConnect_Click()
  If mnuSettingsConnect.Caption = "&Connect" Then
    ' trap connection problems
    On Error GoTo NoConnect
    MSComm1.PortOpen = True
    frmMain.Caption = "PC-to-Stamp Comm [Connected]"
    ' clear error trapping
    On Error GoTo 0
    mnuSettingsConnect.Caption = "&Disconnect"
    ' disable port selection
    mnuSettingsComm1.Enabled = False
    mnuSettingsComm2.Enabled = False
    ' enable transfer buttons
    Call EnableButtons
  Else
    MSComm1.PortOpen = False
    frmMain.Caption = "PC-to-Stamp Comm"
    mnuSettingsConnect.Caption = "&Connect"
    ' enable port selection
    mnuSettingsComm1.Enabled = True
    mnuSettingsComm2.Enabled = True
    ' disable transfer buttons
    Call DisableButtons
  End If
  Exit Sub

NoConnect:
  Dim response
  ' display port connection problem
  response = MsgBox("Error: Could not connect.", _
                    vbExclamation + vbOKOnly, _
                    "PC-to-Stamp Comm", "", 0)
  On Error GoTo 0
End Sub

Private Sub mnuSettingsComm1_Click()
  mnuSettingsComm1.Checked = True
  mnuSettingsComm2.Checked = False
  MSComm1.CommPort = 1
End Sub

Private Sub mnuSettingsComm2_Click()
  mnuSettingsComm1.Checked = False
  mnuSettingsComm2.Checked = True
  MSComm1.CommPort = 2
End Sub

Private Sub scrValue_Change()
  ' show value
  lblValue.Caption = Str(scrValue.Value)
End Sub

Private Sub scrValue_Scroll()
  Call scrValue_Change
End Sub

Private Sub FlushTxBuf()
  ' flush transmit buffer
  Do
    DoEvents
  Loop Until (MSComm1.OutBufferCount = 0)
End Sub

Private Sub DisableButtons()
  cmdDnload.Enabled = False
  cmdUpload.Enabled = False
  mnuFileExit.Enabled = False
```

```
End Sub

Private Sub EnableButtons()
  cmdDnload.Enabled = True
  cmdUpload.Enabled = True
  mnuFileExit.Enabled = True
End Sub
```

```
' =========================================================================
'
'   Program Listing 89.2
'   File...... StampToPC.BS2
'   Purpose... Serial Comm with PC using Flow Control
'   Author.... Jon Williams
'   E-mail.... jwilliams@parallaxinc.com
'   Started... 01 AUG 2002
'   Updated... 02 AUG 2002
'
'   {$STAMP BS2}
'
' =========================================================================


' -------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------

' This program (and its companion VB program) demonstrate serial communications
' with a PC using a MAX232 level shifter and flow control.
'
' Using flow control allows the Stamp to complete a task without missing data
' from the PC.
'
' Note: If you have two serial connections on your computer, you can use
'       DEBUG to view the Stamp side of the communications process


' -------------------------------------------------------------------------
' Revision History
' -------------------------------------------------------------------------


' -------------------------------------------------------------------------
' I/O Definitions
' -------------------------------------------------------------------------

RX              CON     0                       ' receive (from PC)
TX              CON     1                       ' transmit (to PC)
RTS             CON     2                       ' Request To Send (from PC)
CTS             CON     3                       ' Clear To Send (to PC)

LEDs            VAR     OutD                    ' task indicator


' -------------------------------------------------------------------------
' Constants
' -------------------------------------------------------------------------
```

```
Baud            CON     84                        ' 9600-N-8-1

RxNum           CON     $C0                       ' rx number from PC
TxNum           CON     $C1                       ' tx number to PC
RxStr           CON     $C2                       ' rx string from PC
TxStr           CON     $C3                       ' tx string to PC

NumTasks        CON     4
Off             CON     %1111                     ' all LEDs off


' -------------------------------------------------------------------------------
' Variables
' -------------------------------------------------------------------------------

task            VAR     Nib                       ' current Stamp task

cmd             VAR     Byte                      ' command from PC
aNumber         VAR     Word                      ' numeric value from PC

idx             VAR     Byte                      ' EEPROM address pointer
dByte           VAR     Byte                      ' data byte from PC


' -------------------------------------------------------------------------------
' EEPROM Data
' -------------------------------------------------------------------------------

StrLen          DATA    21
Msg             DATA    "Nuts & Volts Magazine", 0


' -------------------------------------------------------------------------------
' Initialization
' -------------------------------------------------------------------------------

Initialize:
  LEDs = Off                                      ' all LEDs off
  DirD = %1111                                    ' make port D outputs

  HIGH CTS                                        ' not ready to receive yet


' -------------------------------------------------------------------------------
' Program Code
' -------------------------------------------------------------------------------

Main:
  SERIN RX\CTS, Baud, 5, Do_Task, [cmd]
  ' DEBUG HEX ?cmd
  LOOKDOWN cmd, [RxNum, TxNum, RxStr, TxStr], cmd
  BRANCH cmd, [RX_Number, TX_Number, RX_String, TX_String]


Do_Task:
  BRANCH task, [Task_0, Task_1, Task_2, Task_3]
  task = 0                                       ' fix bad task spec
  GOTO Main


Task_0:
  LEDs = %1110                                   ' show current task
  PAUSE 250                                      ' take some time doing it
  LEDs = Off
  task = task + 1 // NumTasks                    ' point to next task
  GOTO Main


Task_1:
  LEDs = %1101
  PAUSE 250
  LEDs = Off
  task = task + 1 // NumTasks
  GOTO Main


Task_2:
  LEDs = %1011
  PAUSE 250
  LEDs = Off
  task = task + 1 // NumTasks
  GOTO Main


Task_3:
  LEDs = %0111
  PAUSE 250
  LEDs = Off
  task = task + 1 // NumTasks
  GOTO Main


' -------------------------------------------------------------------------------
' Subroutines
' -------------------------------------------------------------------------------

RX_Number:
  SERIN RX\CTS, Baud, [aNumber.LowByte, aNumber.HighByte]
  ' DEBUG DEC ?aNumber
  GOTO Do_Task
```

```
TX_Number:
  ' DEBUG "Sending: ", DEC aNumber, CR
  SEROUT TX\RTS, Baud, [aNumber.LowByte, aNumber.HighByte]
  GOTO Do_Task


RX_String:
  idx = 0                                         ' reset address pointer

RX_Char:
  SERIN RX\CTS, Baud, [dByte]                     ' receive char from PC
  WRITE (Msg + idx), dByte                        ' save it to EEPROM
  IF (dByte = 0) THEN RX_Str_Done                 ' wait for another if not 0
  idx = idx + 1                                   ' update address pointer
  GOTO RX_Char

RX_Str_Done
  WRITE StrLen, idx                               ' save string length
  GOTO Do_Task


TX_String:
  READ StrLen, idx                                ' get string length
  SEROUT TX\RTS, Baud, [idx]                      ' send string length
  SERIN RX\CTS, Baud, 100, Do_Task, [cmd]         ' wait for restart
  IF (cmd <> TxStr) THEN Do_Task                  ' abort if bad command
  idx = Msg

TX_Char:
  READ idx, dByte                                 ' get char from EEPROM
  IF (dByte = 0) THEN TX_Str_Done                 ' check for end
  SEROUT TX\RTS, Baud, [dByte]                    ' send char
  idx = idx + 1                                   ' point to next
  GOTO TX_Char

TX_Str_Done:
  GOTO Do_Task
```