Implementing Serial I/O with RobotBASIC

RobotBASIC has commands that can utilize any serial com port (real or virtual) on a PC to do serial I/O with any device that can communicate using Asynchronous Serial I/O. These ports could be real physical RS232 ports or they can be Bluetooth virtual ports, or a USB to serial converter virtual port. As far as RB is concerned there is no difference between a virtual port and a real port. All RB needs is the Com port number assigned to the port. Additionally, RB has a receive buffer. So a program can receive data while it is busy doing other tasks.

The Scope Of This Article:

This article will show how to send/receive data to and from a Basic Stamp (BS2) as particular example of serial data communications. However, the principles can be applied with ***any*** other microcontroller or other communications device or even with instrumentation devices that can be controlled through the use of the RS232 standard. See the document RobotBASIC_To_PropellerChip_Comms.PDF for demo programs that are similar to the ones given here but that work with the Propeller Chip from Parallax Inc. and uses the Spin language.

The article will not show how to setup a BS2 to do the serial communications. It will only deal with the programming of the BS2 not how it ought to be connected the PC's RS232 port. This com link could even be a wireless one through a Bluetooth link. For details on how to arrange the setup, read **RobotBASIC_To_BS2_Comms.pdf**.

Serial Communications:

By its nature serial communication occurs on a byte by byte basis. Actually, it is on a bit by bit basis, but the UART (the electronic hardware that carries out the actual data I/O) is given a byte (8-bits) and it takes care of shifting out/in the entire byte. Therefore, as far as programming is concerned the trnasfer occurs byte by byte.

In RobotBASIC there is a receive buffer which can be thought of as a one-dimensional array of bytes (4095 bytes long). There is also a send buffer used when RB sends serial data but this buffer gets sent

out immediately if there is a device on the other end. If the device does not have a receive buffer of its own the data will be lost.

When a device sends serial data to the serial port assigned to RB, the bytes will be appended to the end of the receive buffer. This buffer will hold up to 4095 bytes before it starts to write again from the beginning. So if RB is busy doing other things and more than 4095 bytes are received before RB can read them, the buffer will start filling again from the beginning and thus overwriting the previously received bytes. This will of course cause a problem since you will be reading wrong data when you do read the buffer. So an RB program will have to monitor the buffer and make sure to read the data out of the buffer before it becomes full. The reading action will also clear the buffer and start filling from the start again.

RobotBASIC can do three types of serial communications. It can periodically check for the number of bytes in the buffer while it is doing other tasks, or it can wait with a time out period for a certain number of bytes to arrive. The third way is a little more complex since it uses event handling (interrupts). RB can trigger an event handler whenever bytes arrive in the receive buffer. The handler can then do any action necessary like reading the bytes into another buffer and then doing something only when the required number or set of bytes have been accumulated. We will not consider this third method in this article since it requires more advanced programming principles.

RobotBASIC's Serial Communications Commands:

To accomplish serial communications with RB the program has to be able to:

1- Indicate the serial port to be used and the Com parameters like the baud rate and so forth (SetCommPort command).
2- Clear the receive buffer (ClearSerBuffer command)
3- Check the amount of data in the receive buffer (CheckSerBuffer command)
4- Get bytes from the receive buffer (SerIn and SerBytesIn commands)
5- Put bytes into the send buffer (SerOut and SerialOut commands).
6- Change the timeout period (SetTimeOut and GetTimeOut commands).

The above 9 commands are all that you need to send and receive serial data. However, there are also a few functions that are needed to manipulate the data to be sent and received. We shall see these later.

Sending Text Data (ASCII data):

To send text or numbers converted to text you use the SerOut command. This command will take parameters that can be either strings or numbers. If a parameter is a number it will be converted to its string equivalent. So the number 234 will be sent as three separate bytes with each byte containing the

numbers 50, 51 and 52. Why these values? The answer is that these are the ASCII codes of the alphanumeric digits 2, 3 and 4. If you want to send the number 234 as a binary one byte value of 0xEA see the next section.

So let's say you want to send the string "hello" then the numbers 45 and 0xAD (=173) as strings. You would do this:

```
SerOut "hello",45,0xAD
```

What will be sent over the serial line is 10 bytes with the following values:

104	101	108	108	111	52	53	49	55	51
h	e	l	l	o	4	5	1	7	3

To send text data or numbers converted to their text equivalent use the SerOut command.

Sending Binary Byte Data:

In the above section when we sent the number 173 (0xAD) we used 3 bytes to send it. Depending on the application this might be necessary. However, there is a more efficient way of sending a number. We can send the number as a binary value. An integer like 32128765 would require 8 bytes to send as text. It becomes even worse when we consider a float. π for instance would require 16 bytes if sent as a text. Conversely, if we send integers as binary values we would need 4 bytes for even the largest integer supported by RB, and floating point numbers can be sent as 8 bytes.

Sending binary data bigger than one byte over the serial line can be achieved in many ways. Which method you use depends on what format the receiver expects the data to be in. In this section we will consider sending numbers that are not any bigger than 255 (0xFF), and thus they are one byte values and can be sent as one byte. In the next section we will have a look at how to send numbers that are bigger than one byte.

As in the previous section we want to send "hello" then 45 then 173 (0xAD). We do this:

```
SerialOut Left("hello there",5),32+13,0xAD
```

SerialOut as opposed to SerOut will treat numeric parameters as one byte integers. Even if the number is bigger than one byte it will be truncated to one byte. So if you have a number like 345 which is two bytes 0x0159, only one byte will be sent (the LSByte i.e. the first byte from the right, or the least significant byte). So only 0x59 will be sent; ***therefore, be careful with this***. If you want to send numbers bigger than one byte see the next section.

In the above statement, instead of 45 we used 32+13 to illustrate that any parameter is actually an expression and you can either use a literal number (or string), or a calculated number (or string) and the calculation can of course use any functions or operators and variables etc. [See the RobotBASIC help file, the Language Overview section for a definition of an expression].

The statement above will cause 7 bytes to be sent over the serial line (notice the last two bytes are the actual numbers):

104	101	108	108	111	45	173
h	e	l	l	o		

To send data consisting of text and numbers not bigger than a byte (0 to 255) use the SerialOut command.

Sending Binary Multiple-Byte-Number Data:

The method you use to send numbers bigger than one byte depends on the receiving device. This can become a complex situation with different processors expecting numbers like integers to be in formats that are not the same as other processors. For example on the PC (Intel) a 32-bit number (4 bytes) is stored in the Little-endian format, while on Motorola processors they are stored using the Big-endian format. This becomes even a bigger mess when you consider floating point numbers and negative numbers and so forth.

We won't go into all this since it is not necessary for most work, like sending data to a BS2. Nonetheless, RobotBASIC has functions that facilitate manipulating a ***String Buffer*** as an ***array of bytes*** to insert integers, floats, bytes and text into it and read from it. See BuffWrite(), BuffWriteB(), BuffRead(), BuffReadI(), BuffReadB(), BuffReadF(), toByte(), char(), PutStrByte(), GetStrByte(), ToNumber(), ToString() and much more. Once you have formed a buffer with all the necessary binary data (and text if needed) you can send it using SerOut or SerialOut as a one string parameter.

For a more in depth information about this subject with examples see Section 3 in the document **RobotBASIC_Networking.pdf** also see the document RobotBASIC_To_PropellerChip_Comms.PDF for another practical example of sending and receiving binary data.

In this section we will consider sending a two-byte-integer. We will be using the Most Significant Byte First order (MSBF). That means that an integer value like 0xA3CD will be sent as two bytes 0xA3 then 0xCD. The 0xA3 will be sent first.

RobotBASIC integers are 4 bytes long. If you know that the value in the integer is no bigger than 0xFFFF (65535) then you can treat the integer as a two-byte integer. There is one other point of import

that should be considered. If you want to use negative numbers you must consider an appropriate format. One ubiquitous format is the 2's Compliment format. But this is another topic that is too complex to consider here. We will just stick to unsigned numbers.

RobotBASIC has various bit-manipulation functions and operators. Operators like >> and << and &, @, ~ and | are able to manipulate bits in a number. Also functions like MakeBit(), MakeByte(), GetBit() and GetByte() can be used to set, reset or get particular bits or bytes from an integer.

So for example to obtain the second byte (from right to left and first byte is byte number 0) from the number 4567 stored in the variable n, then you would use `GetByte(n,1)`.

So to send the string "hello" and the number 4567 over a serial line with the format discussed above we do:

```
SerialOut "hello",GetByte(4567,1),GetByte(4567,0)
```

Or

```
SerialOut "hello", (4567 >> 8), 4567
```

Note: SerialOut is much more convenient to use in this case but you can still use SerOut if you wish. Here is how:

```
SerOut "hello", char((4567 >> 8)), char(4567)
```

Notice that the char() function is used to make sure that the resulting integer will be made into one byte. You can also use toByte() instead. Although for sending binary bytes, it is much more convenient to use SerialOut.

Receiving Text Data (ASCII data) Or Binary data:

RobotBASIC has a receive buffer that can hold up to 4095 bytes. So an RB program can perform actions while data is being received from the serial port. However, you should make sure that any data in the buffer is read and stored somewhere before the buffer becomes full. It is a circular buffer so data that comes in after the buffer is full will start to write over the old data from the beginning again. ***Reading the buffer will clear it***.

There are three ways you can wait for the data to arrive into the buffer:

1- ***Polling.*** With this method the RB program can be doing other actions and periodically check the buffer to see if the required number of bytes have arrived and then act accordingly.

   To do this use the CheckSerBuff command to get a count of the number of bytes in the buffer. Then when the time comes to read the buffer use SerIn to read the entire buffer into a string (array of bytes).

2- ***Timeout Waiting***. In this method the program stops and waits for the right number of bytes to arrive. If the bytes arrive they would be read and returned into a string (array of bytes). If the right number of bytes do not arrive before a specific time period has elapsed then the characters that have arrived will be returned into a string and the count of how many into a variable and then the program flow goes on.

   Use SerBytesIn to get the required bytes and use SetTimeOut to set the required time if you need to change the default which is 5000 milliseconds.

3- ***Interrupt***. This method utilizes event driven programming. Using OnSerial you can signal RB to branch to a particular subroutine (event handler) whenever the serial buffer gets a byte or more. The program will then be able to do whatever other processing is required, but whenever a byte comes in on the serial line the program flow will branch to the event handler routine. In the handler you read the bytes already there and store them and if more actions need to be taken they are carried out, but afterwards the program will go back to where it was before branching to the handler.

   This method is too complex for the scope of this article and will not be discussed here.

Whether you use the first or second method you will eventually end up with a string that has the bytes that were received. The first byte in the string is the first byte to have arrived and so on.

You can treat this string as a normal string. So if you know all the bytes in it are ASCII characters that represent text or numbers that were converted to their textual representations then you can use this string for all intents and purposes as any other string in RB. If you want to convert string numbers to actual numbers you can use ToNumber() to convert them to a numeric (integer or float depending on its value).

If the data contains binary numbers whether single bytes or multi-byte numbers then you must treat the string as an ***array of bytes (buffer).***

There are various functions in RB that help in obtaining the value a particular byte from an array of bytes (string). BuffReadB() treats the buffer as 0 indexed array (1st character is character number 0 and so forth) and GetStrByte() treats the buffer as a string and thus the 1st character is character 1 and so forth. However, both functions will return a numerical result which is actually an integer but with only the LSByte (first from right) filled with a value which is the value of the byte from the desired position in the buffer.

You can also use any of the string functions like SubString() and so on. Additionally, see the functions BuffRead(), BuffReadI(), BuffReadF(), BuffWrite(), BuffWriteB(), and PutStrByte().

So if you have a string buffer that has arrived and you know that the first 5 bytes are a text and the next byte is an 8 bit number and the next 2 bytes are a 16 bit number with the MSBF order then you can do this (assuming the variable **s** holds the bytes):

```
Print Left(s,5)  //prints the text (5 bytes)
Print GetStrByte(s,6) //prints the 8-bit number which is one
                      //byte and is the 6th character.
Print BuffReadB(s,5)  //prints the 8-bit number which is one byte
                      //and is the 6th character but is therefore
                      //byte number 5 in a zero indexed basis.
Print (GetStrByte(s,7)<<8)+GetStrByte(8)  //creates the 16-bit number
                                          //from the 7th and 8th
                                          //characters.
Print (BuffReadB(s,6)<<8)+BuffReadB(s,7)  //does the same as above…
                                          //notice indexing.
```

BuffReadI(), BuffReadF() are other functions that can also help with numbers like floats and 32-bit integers but you must read the RobotBASIC help about these functions carefully.

A Practical Example:

The following two demos do the exact same action. They allow you to enter two numbers on the PC using RB and then, at the push of a button, these two numbers are sent to the BS2 where they are added and the result is sent back to the PC where it is displayed. You can repeat the actions as often as desired. Initially the RB program will wait for you to reset the BS2 which will send a string "Rdy" to indicate that it is ready to receive the number and is up and running. This also serves as a way of synchronization.

The first pair of programs is the RobotBASIC and PBasic programs of the first demo that sends and receives the numbers as Text. The second pair sends and receives the numbers as 2-Byte binary numbers with the MSBF order.

The demos assume that the BS2 will ***receive*** data through its I/O pin P0 and will send data through its I/O pin P1. It is also assumed that the BS2 is linked to the Com2 port on the PC, so the RobotBASIC program will use this port. If you have a different setup all you need to do is change the appropriate assignments in the programs.

See the document RobotBASIC_To_PropellerChip_Comms.PDF for demo programs that are similar to the ones given here but that work with the Propeller Chip from Parallax Inc. and use the Spin language.

RobotBASIC_Demo1.BAS (Send/Receive Text numbers):

```
CommPort = 2  //change this to reflect your setup
setcommport CommPort,br9600,8,pNone,sbOne,fcNone
ClearSerBuffer
xystring 10,10,"Reset the BS2 now...."
while true
   repeat  //wait for the "Rdy"
     checkserbuffer n
   until n == 3
   SerIn s \ if s == "Rdy" then break
wend //repeat until the "Rdy" string is received
//add edit boxes and buttons and instructions
addedit "v1",50,50,100,0,70
addedit "v2",170,50,100,0,48
addedit "re",310,50,100,0,"0"  \ readonlyedit "re"
addbutton "Calc",80,100,100
xystring 10,10,"OK the BS2 is ready...."
xystring 10,25,"Enter numbers in the edit boxes then press the 'Calc' button"
xytext 155,45,"+","",15
xytext 290,45,"=","",15
while true  //repeat for ever
  repeat  //wait for the calc button to be pressed
  until LastButton() =="Calc"
  //get the numbers from the edit boxes
  v1 = tonumber(getedit("v1"),0) \ v2 = tonumber(getedit("v2"),0)
  //send the two numbers as text one after the other
  //the delay is to give the BS2 time to be ready
  serout v1," " \ delay 10 \  serout v2," "
  repeat //wait until there is atleast one byte in the buffer
    checkserbuffer n
  until n>=1
  delay 100 //give the the BS2 time to finish
            //sending the text number there is no way to know
            //how many characters but should not take longer
            //then 100 ms to send all of them
  serin s   //read the buffer
  //assign the number (text) to the result edit box
  setedit "re",s
wend
```

BS2_Demo1.Bs2 (Send/Receive Text numbers):

```
' {$STAMP BS2}
' {$PBASIC 2.5}
RcvPin PIN 0
SndPin PIN 1
v1     VAR Word
v2     VAR Word
SEROUT SndPin,84,["Rdy"]
DO
```

```
   SERIN RcvPin,84,[DEC v1]
   SERIN RcvPin,84,[DEC v2]
   DEBUG DEC v1,"-", DEC v2,CR
   v1 = v1+v2
   SEROUT SndPin,84,[DEC v1]
LOOP
```

RobotBASIC_Demo2.BAS (Send/Receive 2-byte binary numbers):

```
CommPort = 2  //change this to reflect your setup
setcommport CommPort,br9600,8,pNone,sbOne,fcNone
ClearSerBuffer
xystring 10,10,"Reset the BS2 now...."
while true
   repeat  //wait for the "Rdy"
     checkserbuffer n
   until n == 3
   SerIn s \ if s == "Rdy" then break
wend //repeat until the "Rdy" string is received
//add edit boxes and buttons and instructions
addedit "v1",50,50,100,0,70
addedit "v2",170,50,100,0,48
addedit "re",310,50,100,0,"0"  \ readonlyedit "re"
addbutton "Calc",80,100,100
xystring 10,10,"OK the BS2 is ready...."
xystring 10,25,"Enter numbers in the edit boxes then press the 'Calc' button"
xytext 155,45,"+","",15
xytext 290,45,"=","",15
while true  //repeat for ever
  repeat  //wait for the calc button to be pressed
  until LastButton() =="Calc"
  //get the numbers from the edit boxes
  v1 = tonumber(getedit("v1"),0) \ v2 = tonumber(getedit("v2"),0)
  //send the two numbers as two-byte binaries
  //----this is one way
  serialout (v1>>8),v1,(v2>>8),v2
  //----the following lines are other ways to do the same thing...commented out
  //serialout GetByte(v1,1),GetByte(v1,0),GetByte(v2,1),GetByte(v2,0)
  //serout char((v1>>8)),char(v1),char((v2>>8)),char(v2)
  //serout toByte((v1>>8)),toByte(v1),toByte((v2>>8)),toByte(v2)
  repeat //this time we know to expect two bytes
    checkserbuffer n
  until n==2
  serin s  //read them
  //now construct the number from the 1st byte and the 2nd byte
  a = (getstrbyte(s,1)<<8)+getstrbyte(s,2)
  //assign the number to the result edit box
  setedit "re",a
wend
```

BS2_Demo2.Bs2 (Send/Receive 2-byte binary numbers):

```
' {$STAMP BS2}
' {$PBASIC 2.5}
RcvPin PIN 0
SndPin PIN 1
v1     VAR Word
v2     VAR Word
v1H    VAR v1.HIGHBYTE
v1L    VAR v1.LOWBYTE
v2H    VAR v2.HIGHBYTE
v2L    VAR v2.LOWBYTE
SEROUT SndPin,84,["Rdy"]
DO
   SERIN RcvPin,84,[v1H,v1L,v2H,v2L]
   DEBUG DEC v1,"-", DEC v2,CR
   v1 = v1+v2
   SEROUT SndPin,84,[v1H,v1L]
LOOP
```