4: Programming for Data and XBee Configurations

Data between systems can take several forms depending on need. It may be raw bytes, which can represent a variety of information such as characters, input/output bits, or simply a binary value. While the XBee modules can help ensure the data arrives correctly to the controller, it is up to our code to ensure it is accepted properly and used in the manner desired. Just because our data makes it out of the XBee it doesn't necessarily mean it was accepted properly by the controller.

In some cases a controller may send a value repeatedly, such as a sensor value. It may not be important that some data is missed by the controller as long as a recent sample is obtained. In other cases it may be more critical that each transmission is accepted and used by the controller. Sometimes data can simply be a single byte. Other times it may be a collection of values that needs to be accepted in the proper order.

This section will explore data reception and transmission by the BASIC Stamp and Propeller microcontrollers and explore some ways of handling different tasks for data reception. It will also explore configuring the XBee from the controllers using the AT Command Codes. A base XBee will be used connected to the PC to communicate to a remote XBee interfaced to a microcontroller as shown in Figure 4-1.

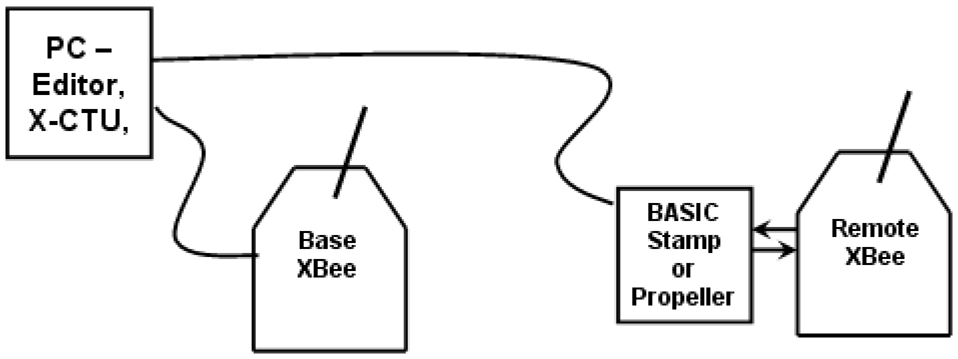


Figure 4-1: Base and Remote XBee General Setup

BASIC Stamp

Hardware & Software

For these tests a single remote BASIC Stamp will be used for communications with a base XBee connected to the PC using USB and X-CTU software. Data will be manually entered to be transmitted to the BASIC Stamp to investigate data reception.

Hardware:

- BASIC Stamp microcontroller and development board
- 2 XBee modules (Ensure they are restored to default configuration)
- A 5V/3.3V XBee Adapter Board, or SIP Adapter
- XBee USB Adapter Board & cable
- (2) 220 Ω resistors and (2) LEDs (optional)

Connect the BASIC Stamp to an XBee Adapter Board as shown in Figure 4-2, depending on the style of board used. LEDs and resistors are optional but are useful. Note that the DOUT and DIN pins are not jumpered together.

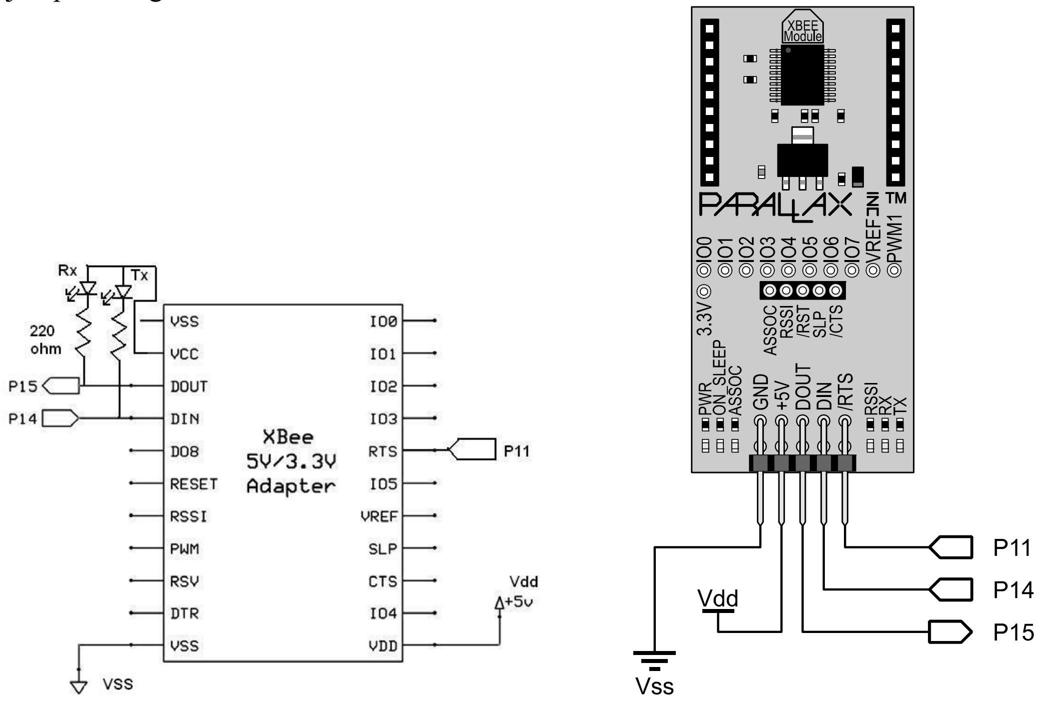


Figure 4-2: XBee 5V/3.3V Adapter Connection to BASIC Stamp (DIP and SIP version)

XBee 5V/3.3V Adapter Rev B:

The XBee 5V/3.3V Adapter Board Rev B has onboard Tx and RX LEDs, so you will not need to add the LED circuits that are shown in the schematic above.

Software:

- BASIC Stamp Editor (www.parallax.com/basicstampsoftware)
- Digi’s X-CTU Software (www.parallax.com/go/xbee)

Simple DEBUG to Remote Terminal

Being able to monitor a remote unit is very beneficial, such as for robotics or monitoring remote sensors. This first code example will display text strings and values back from the remote unit to the base for monitoring.

```
' ********************************************************
' Simple_Debug.bs2
' Sends data to remote terminal for monitoring
' ********************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

' *************** Constants & PIN Declarations ***********
```

```
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT
Baud            CON     T9600

Rx              PIN     15   ' XBee DOUT - Not used in this example
Tx              PIN     14   ' XBee DIN
RTS             PIN     11   ' XBee RTS - Not used in this example

' ************** Variable Declarations ******************
Counter         VAR     Byte

' ************** Main LOOP ******************************
PAUSE 500                       ' 1/2 second pause to stabilize comms
SEROUT Tx, Baud, [CLS,"Program Running...",CR]

PAUSE 2000                      ' Pause before counting

FOR Counter = 1 TO 20           ' Count and display remotely
  SEROUT Tx, Baud, ["Counter = ", DEC Counter, CR]
  PAUSE 100
NEXT

SEROUT Tx, Baud, ["Loop complete.",CR]
END
```

Code discussion:

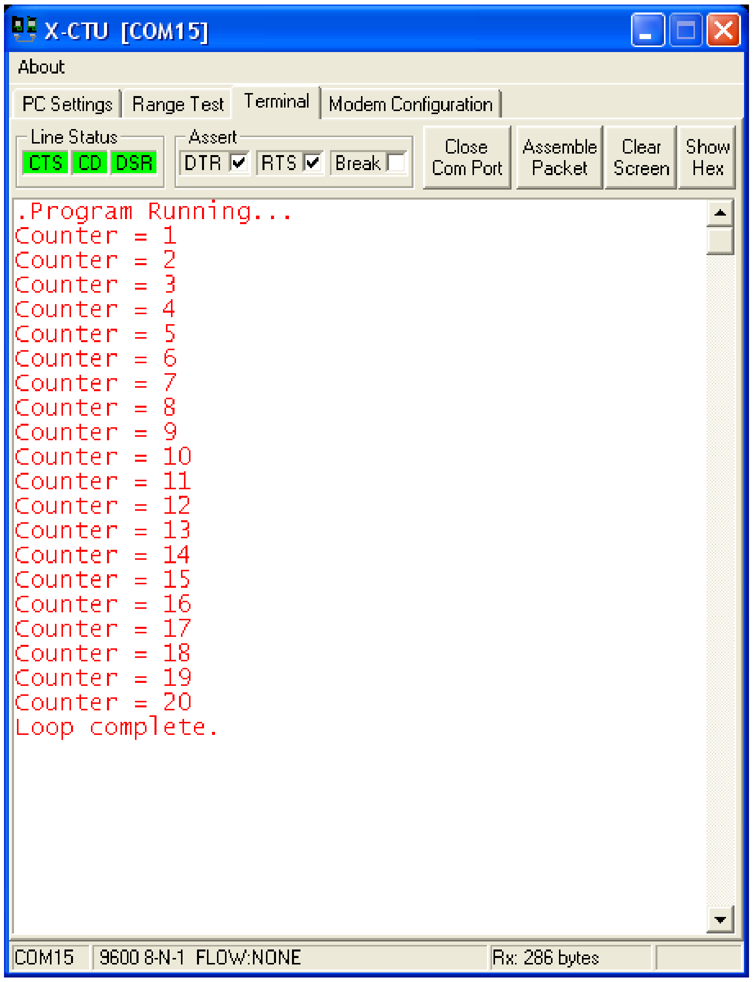
- The **`SELECT-CASE`** structure is used to ensure a baud rate of 9600 no matter the model of BASIC Stamp you may be using.
- DOUT and RTS are not used in this example but will be used in later examples.
- It displays a message, the value of **`Counter`** from 1 to 20, and a final message.

Terminals

While we use the X-CTU in our example, you may also a BASIC Stamp Editor Debug Terminal selected to the correct COM port. The **`CLS`** (clear screen) only shows a dot in X-CTU, but will clear the Debug Terminal.

Test the code:

✓ Connect and monitor the base XBee using X-CTU or other terminal program.
✓ Download **Simple_Debug.bs2** to the BASIC Stamp connected to the remote XBee unit.
✓ Monitor the debugged data in the base terminal window as shown in Figure 4-3.



```
.Program Running...
Counter = 1
Counter = 2
Counter = 3
Counter = 4
Counter = 5
Counter = 6
Counter = 7
Counter = 8
Counter = 9
Counter = 10
Counter = 11
Counter = 12
Counter = 13
Counter = 14
Counter = 15
Counter = 16
Counter = 17
Counter = 18
Counter = 19
Counter = 20
Loop complete.
```

COM15 | 9600 8-N-1 FLOW:NONE | Rx: 286 bytes

Figure 4-3: Simple Debug to Base Testing

Accepting and Displaying Single Bytes

In this next code example, we will use the X-CTU terminal to transmit bytes to the BASIC Stamp to accept, display locally and transmit back to the base XBee for display in X-CTU (echoed).

```
' ********************************************************
' Simple Byte Receive.bs2
' Accepts, displays and echoes back incoming bytes
' ********************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

' *************** Constants & PIN Declarations ***********
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600        CON     84
  #CASE BS2SX, BS2P
    T9600        CON     240
  #CASE BS2PX
    T9600        CON     396
#ENDSELECT
```

```
Baud            CON     T9600

Rx              PIN     15    ' XBee DIN
Tx              PIN     14    ' XBee DOUT
RTS             PIN     11    ' XBee RTS - Not used in this example

' ************** Variable Declarations ******************
DataIn          VAR     Byte

' ************** Main LOOP ******************************
PAUSE 500                    ' 1/2 second pause to stabilize comms
DEBUG "Awaiting Byte Data...",CR

DO
  SERIN Rx, Baud, [DataIn]   ' Accept incoming byte
  SEROUT Tx, Baud, [DataIn]  ' Echo byte back
  DEBUG DataIn               ' Display byte as character
LOOP
```

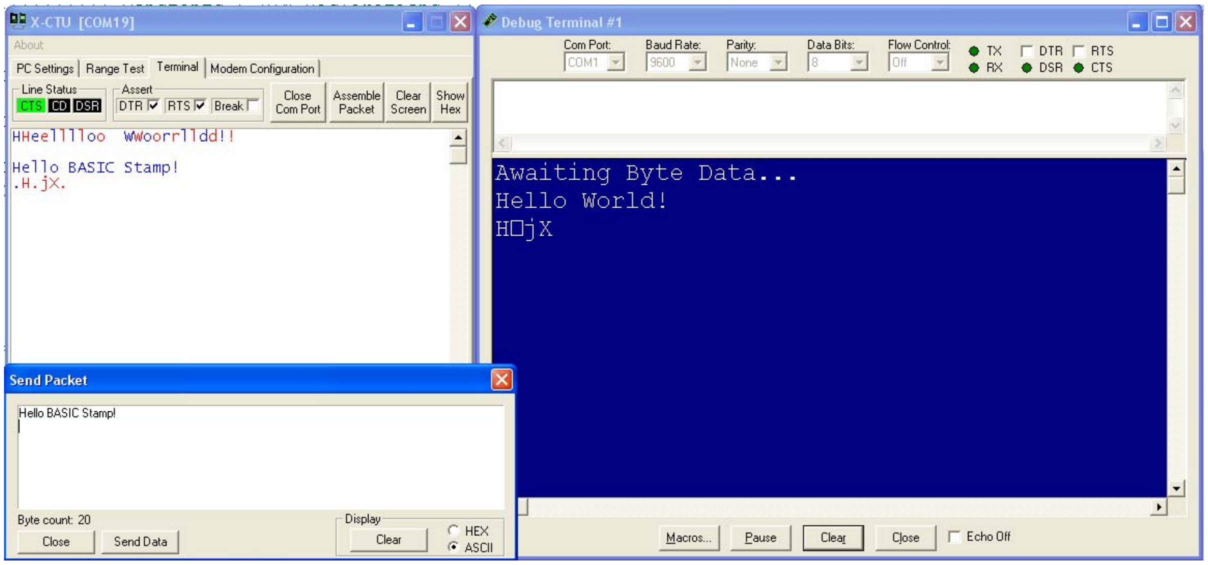
Code discussion:

- ✓ Within the **`DO-LOOP`**, **`SERIN`** waits until it receives a byte from the XBee, sends it back to the XBee to be transmitted, and sends the character to the Debug Terminal of the BASIC Stamp Editor.

Test the code:

- ✓ Download **Simple_Byte_Receive.bs2** to the BASIC Stamp.
- ✓ From the X-CTU terminal window, manually type **Hello World**. Note that it displays in the BASIC Stamp Debug Terminal and the X-CTU terminal window correctly.
- ✓ Use the **Assemble Packet** feature of X-CTU to send "**Hello BASIC Stamp**!" as a single packet. Note that the first character may be correct, but the remaining data is mostly garbage.

Figure 4-4 is an image of our testing. The first byte was accepted by the BASIC Stamp correctly since it was sitting idle waiting for it. Once the first byte was accepted, the program continued on. In the mean time, data continued to be sent from the XBee even though the BASIC Stamp was not accepting it any longer. When the BASIC Stamp looped back, it caught a byte mid-position and continued collection from there getting garbage. When we typed the data directly in to the terminal window, we were typing too slowly to get the data out in a single transmission and the BASIC Stamp was fast enough to loopback for our next character.



```
HHeelllloo  WWoorrlldd!!

Hello BASIC Stamp!
.H.jX.
```

```
Hello BASIC Stamp!
```

```
Awaiting Byte Data...
Hello World!
H□jX
```

Figure 4-4: Receiving Byte Test Results

Sending and receiving raw bytes is generally more difficult in ensuring data is being accepted properly since it could be any byte value, 0 to 255 in binary. Using decimal values can aid in ensuring data is proper.

Accepting and Displaying Decimal Values

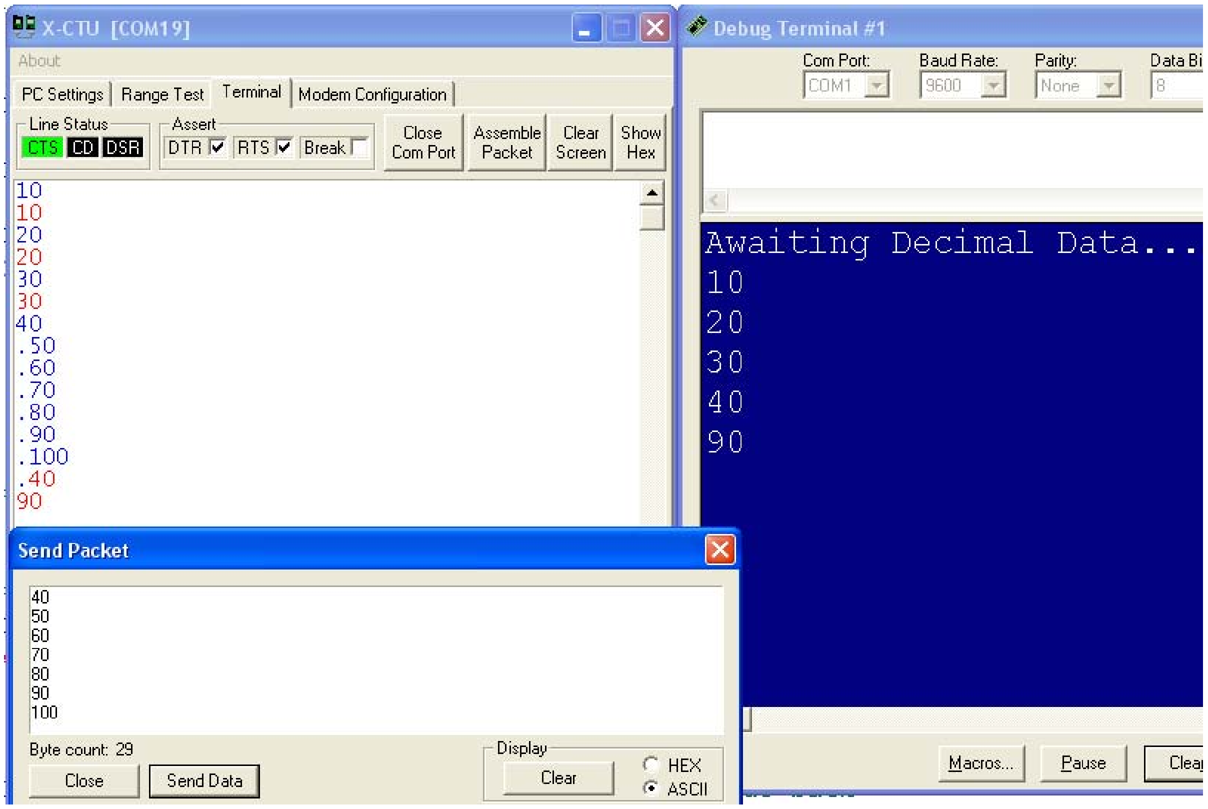
When data is sent or received as decimal values, using PBASIC's **DEC** modifier, values are sent as characters representing the value. Using bytes, a value such as 123 is sent as a single byte with a value of 123. When 123 is sent and accepted using **DEC**, the value is an ASCII character string as '1', '2' and '3'. The BASIC Stamp will ignore or use as the 'end of string' any non-numeric character limiting potential garbage and helping to ensure received data is good. Next, **Simple_Decimal_Receive.BS2** demonstrates accepting, displaying and transmitting decimal values.

```
' ***********************************************************
' Simple Decimal Receive.bs2
' Accepts, displays and echoes back incoming Decimal Value
' ***********************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

' *************** Constants & PIN Declarations ***********
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT
Baud            CON     T9600

Rx              PIN     15     ' XBee DOUT
Tx              PIN     14     ' XBee DIN
RTS             PIN     11     ' XBee RTS - Not used in this example
```

```
' ************** Variable Declarations ******************
DataIn          VAR     Word

' ************** Main LOOP ******************************
PAUSE 500                     ' 1/2 second pause to stabilize comms
DEBUG "Awaiting Decimal Data...",CR

DO
  SERIN Rx, Baud, [DEC DataIn]      ' Accept incoming Decimal Value
  SEROUT Tx, Baud, [DEC DataIn,CR] ' Echo decimal value back
  DEBUG DEC DataIn,CR              ' Display value
LOOP
```

For testing, the X-CTU terminal is again used.

- ✓ Download **Simple_Decimal_Receive.bs2**.
- ✓ In the terminal window of X-CTU, type in a series of numbers, each followed by **Enter** (Carriage Return), such as 10 20 30.
- ✓ Notice that only after **Enter** (or a non-numeric value) data is processed, displayed and sent back. The BASIC Stamp collects data until the character is no longer numeric.
- ✓ Click **Assemble Packet**. Using the Send Packet window, enter a series of numbers to be sent at once, again each followed by **Enter**, such as 40 50 60, etc.
- ✓ Send the packet and monitor the results. The first value is used and displayed, several are skipped, and then another is accepted and used as show in Figure 4-5.



```
10
10
20
20
30
30
40
.50
.60
.70
.80
.90
.100
.40
90
```

Com Port: COM1 Baud Rate: 9600 Parity: None Data Bi 8

```
Awaiting Decimal Data...
10
20
30
40
90
```

```
40
50
60
70
80
90
100
```

Byte count: 29

Display: ( ) HEX (•) ASCII

Figure 4-5: Simple Decimal Receive Testing

The XBee again received your data correctly and sent it to the BASIC Stamp, but after getting the first value, the BASIC Stamp was not available to capture much of the remaining data. Since the BASIC Stamp does not buffer incoming serial data, that data is lost. By using flow control, such as RTS, we can help prevent missed data.

?

Range & Negative Values?

The BASIC Stamp, using Word sized variables, can accept values up to 65535 (16-bit value). To use negative values, use the **SDEC** modifier instead for a range of -32768 to +32767.

Configuring the XBee for using RTS Flow Control & Timeouts

Using the RTS (Ready-to-Send) line, we can limit lost data by having the XBee send data only when the BASIC Stamp is ready to receive it by using the RTS line and function of the **SERIN** command. Additionally, we can use the *Timeout* feature of **SERIN** so the BASIC Stamp can process other code while waiting for data to arrive. To enable RTS on the XBee, AT Command codes will be sent by the BASIC Stamp.

The program **Using_RTS_Flow_Control_for_Bytes.bs2** demonstrates using RTS to collect buffered data from the XBee.

```
' ******************************************************
' Using_RTS_Flow_Control_for_Bytes.bs2
' Configures XBee to Use RTS flow control to
' prevent missed DATA
' ******************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

' *************** Constants & PIN Declarations ***********
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT
Baud            CON     T9600

Rx              PIN     15
Tx              PIN     14
RTS             PIN     11

' ************** Variable Declarations ******************
DataIn          VAR     Word

' ************** XBee Configuration ********************
PAUSE 500
DEBUG "Configuring XBee...",CR
PAUSE 2000                          ' Guard Time
SEROUT Tx,Baud,["+++"]              ' Command mode sequence
PAUSE 2000                          ' Guard Time
```

```
SEROUT Tx,Baud,["ATD6 1,CN",CR]      ' RTS enable (D6 1)
                                     ' Exit Command Mode (CN)'
************** Main LOOP ******************************
PAUSE 500
DEBUG "Awaiting Multiple Byte Data...",CR

DO
  SERIN Rx\RTS,Baud,1000,TimeOut,[DataIn]  ' Use Timeout to wait for byte
  SEROUT Tx, Baud, [DataIn]                ' Send back to PC
  DEBUG DataIn                             ' Display data
  GOTO Done                                ' Jump to done

  Timeout:                                 ' If no data, display
    DEBUG "."

  Done:

LOOP
```

Analyzing the code:

- In the XBee Configuration, the identical action we took using the terminal window is used to configure the XBee: A short delay, sending +++, seeing "OK", another delay, sending AT Commands and using the **CN** command to exit Command Mode. The Command Mode will also terminate automatically after a period of inaction.
- The **`SERIN`** instruction uses the **`\RTS`** modifier to signal the XBee when it is available to accept new data. The **`1000`** and **`Timeout`** label is used to branch if data in not received within 1 second, allowing processing to continue. (See the Timeout units note in the box below.)
- If data is not received within 1 second, the code at the **`Timeout:`** label will cause a dot to be displayed in the Debug Terminal to indicate that and to show processing by the BASIC Stamp is being performed.

Syntax Refresher for SEROUT and SERIN

SEROUT *Tpin* { *Fpin* }, *Baudmode*, { *Pace*, } { *Timeout*, *Tlabel*, } [*OutputData*]

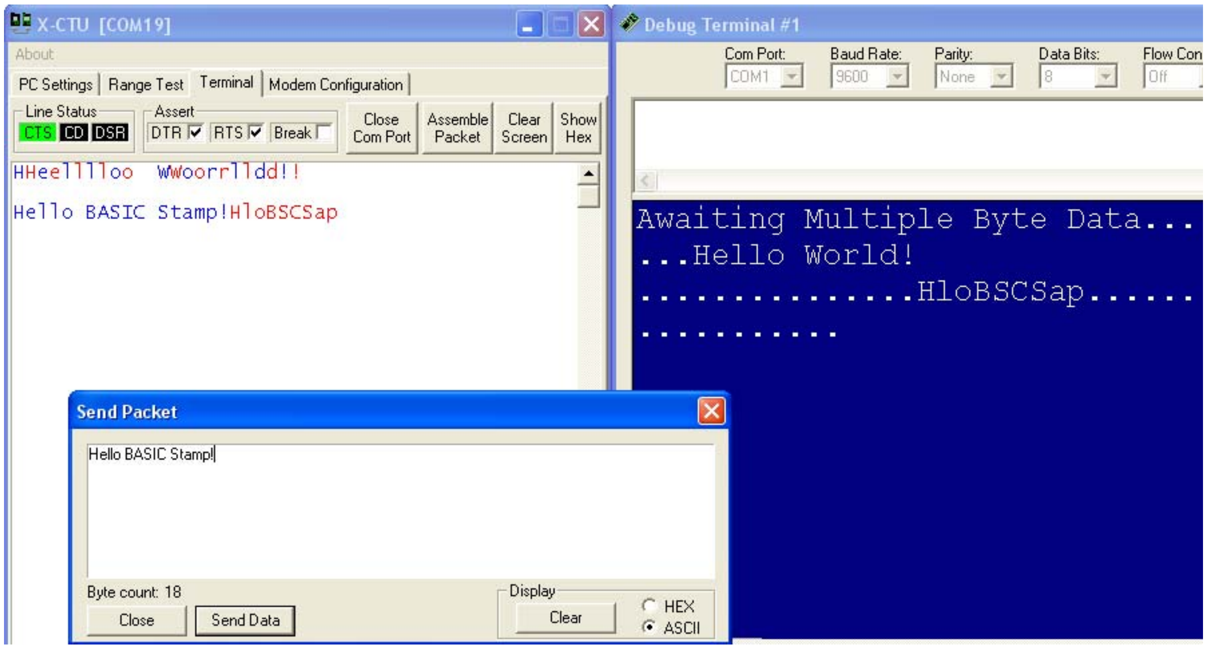
SERIN *Rpin* { *Fpin* }, *Baudmode*, { *Plabel*, } { *Timeout*, *Tlabel*, } [*InputData*]

For full details on these and any other PBASIC commands or keywords, see the BASIC Stamp Editor Help.

Timeout units note: The units in *Timeout* are 1 ms for the BS2, BS2e, and BS2pe, hence a *Timeout* argument of 1000 would be equivalent to 1 second. If you are using a BS2sx, BS2p, or BS2px, the *Timeout* units are only 0.4 ms, so the same value of 1000 would be equivalent to 400 ms—something to keep in mind while you read the code explanations throughout the book.

Testing:

✓ Download **Using_RTS_Flow_Control_for_Byte.bs2.**
✓ Using X-CTU, send a string in both the terminal window and using the Send Packet window.
✓ Notice, as in Figure 4-6, that when sending data as a packet, only every other byte is accepted and processed.



```
HHeellllloo  WWoorrlldd!!

Hello BASIC Stamp!HloBSCSap
```

```
Awaiting Multiple Byte Data...
...Hello World!
...............HloBSCSap......
..........
```

```
Hello BASIC Stamp!
```

Figure 4-6: Using RTS Flow Control for Bytes Testing

The XBee is buffering data allowing the BASIC Stamp to accept more without garbage, but every other byte is missed. The reason for this is that after collecting one byte, the BASIC Stamp de-asserts the RTS line to stop the XBee from sending more data, but it is too slow. The next byte has left the XBee before it gets notification to stop sending data. This happens every cycle causing every other byte to be missed.

Monitor LEDs

When the code starts and goes through the configuration sequence, watching the Tx & Rx LEDs blink back and forth is a great indicator that it the XBee is accepting your command mode functions.

Using RTS and String Delimiter with Decimal Values

Using **DEC** for decimal values can help ensure data is received properly. One issue with receiving multiple values is ensuring what order they are received in and where to start. For example, let's say we want to send two values and send data several times to be accepted and used, such as:

10 & 20
10 & 30
10 & 50

On reception, the BASIC Stamp gets out of sequence so that the data it receives is:

20 & 10
30 & 10

Because it perhaps missed one, the sequence of how it is accepting data does not match the sequence we intended. Through the use of a start-of-string identified or delimiter we can help ensure the data is collect starting at the correct value in the buffer. The same could be done for bytes but it can cause issues since a byte value can be ANY value typically, 0 to 255. A unique value may not be able to be identified for our communications. Using **DEC** values, only 0-9 are valid characters so anything outside of that range would be unique from our data. **Using_RTS_Flow_Control_for_Decimals.bs2** demonstrates using RTS and a string start delimiter.

```
' ********************************************************
' Using_RTS_Flow_Control_for_Decimals.bs2
' Configures XBee to Use RTS flow control to
' prevent missed data using start delimiter
' ********************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}

' *************** Constants & PIN Declarations ***********
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT
Baud            CON     T9600

Rx              PIN     15
Tx              PIN     14
RTS             PIN     11

' ************** Variable Declarations ******************
DataIn          VAR     Byte
Val1            VAR     Word
Val2            VAR     Word

' ************** XBee Configuration ********************
PAUSE 500
DEBUG "Configuring XBee...",CR
PAUSE 2000                          ' Guard Time
SEROUT Tx,Baud,["+++"]              ' Command mode sequence
PAUSE 2000                          ' Guard Time
SEROUT Tx,Baud,["ATD6 1,CN",CR]     ' RTS enable (D6 1)
                                    ' Exit Command Mode (CN)
' ************** Main LOOP *****************************
DEBUG "Awaiting Delimiter and Multiple Decimal Data...",CR

DO
  SERIN Rx\RTS,Baud,500,TimeOut,[DataIn]  ' Briefly wait for delimiter

  IF DataIn = "!" THEN                    ' If delimiter, get data
    SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val1]  ' Accept first value
    SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val2]  ' Accept next value

                                          ' Display remotely and locally
    SEROUT Tx, Baud, [CR,"Values = ", DEC Val1," ", DEC Val2,CR]
    DEBUG CR,"Values = ", DEC Val1," ", DEC Val2,CR
  ENDIF
  GOTO Done                                     ' Jump to Done
  Timeout:                                      ' If no data, display dots
    DEBUG "."
  Done:
LOOP
```

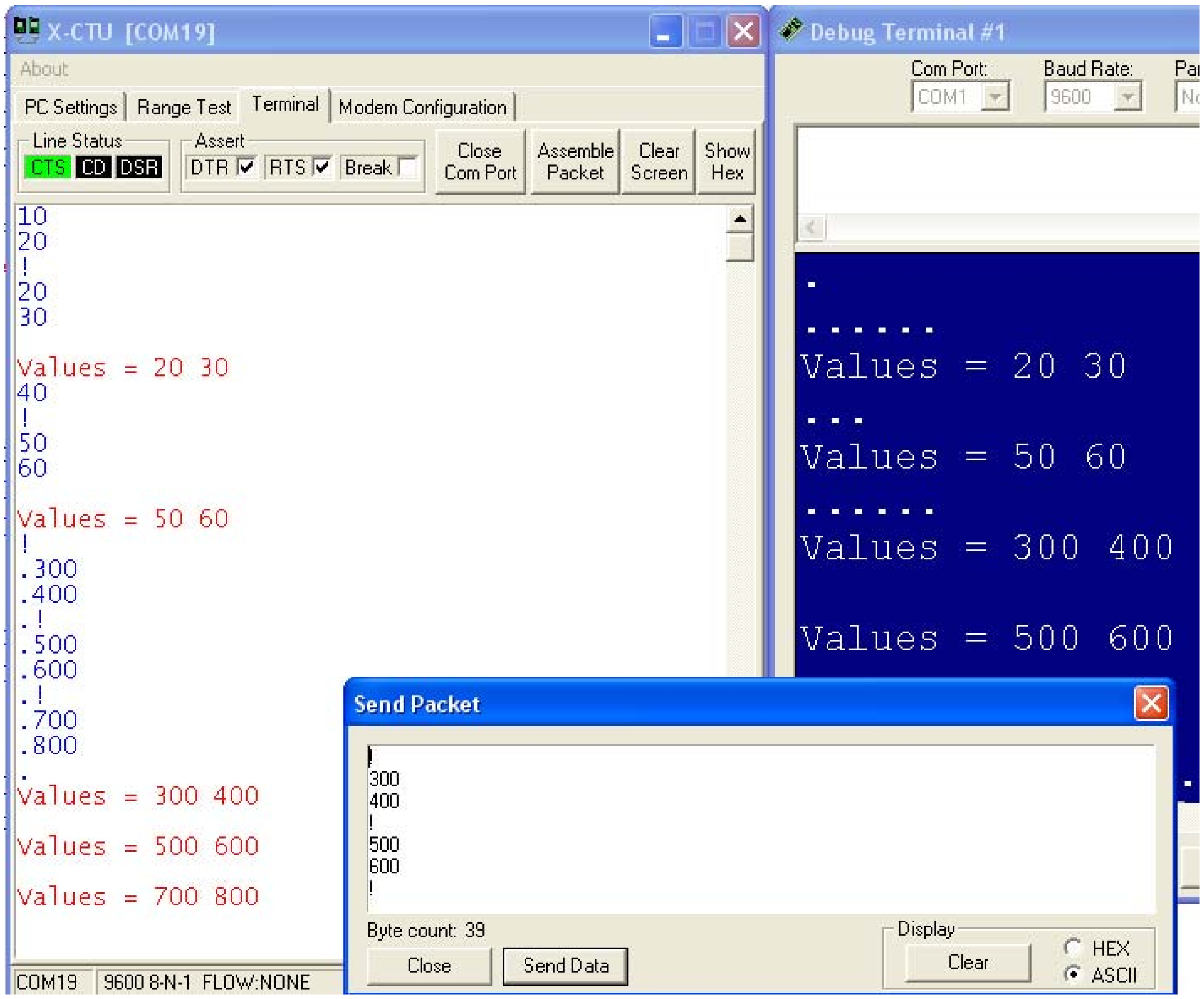
Analyzing the code:

- Again, RTS flow configured for the XBee using AT Commands.
- A **SERIN** with a 500 ms timeout starts the loop. If no data within this time, the timeout dot is shown.
- Next, the code checks to see if the character was the exclamation point "!" and if not, the code is done.
- If it was a !, two decimal values are collected again using timeouts to ensure the code does not lock up waiting.
- The data values are displayed locally and remotely.

Testing the code:

✓ Download **Using_RTS_Flow_Control_for_Decimals.bs2**.
✓ Enter several values, the ! with an **Enter**, then several more values. After entering !, the next two values should be accepted and displayed.
✓ Use the Send Packet window to make a series of numbers and !'s to be sent and test. Notice in Figure 4-7 each set of data following the delimiter was correctly processed.

The decimal values were adequately buffered so that no data was lost. What about those missed bytes when RTS de-asserts? Our Enter or carriage return was at the end of each string of characters so that the lost bytes corresponded to those Enter/carriage return characters.



```
10
20
!
20
30

Values = 20 30
40
!
50
60

Values = 50 60
!
.300
.400
.!
.500
.600
.!
.700
.800
.
Values = 300 400

Values = 500 600

Values = 700 800
```

```
.
......
Values = 20 30
...
Values = 50 60
......
Values = 300 400

Values = 500 600
```

```
300
400
!
500
600
!
```

Figure 4-7: Using RTS for Decimal Values Testing

Reading Data from the XBee – Displaying RSSI Level

The BASIC Stamp can read settings and values from the XBee as well as configuring it. Just as we had done through the X-CTU terminal window in Chapter 3, values can be requested, though it can cause a few issues. Going into Command Mode and performing various AT commands, an "OK" along with data was returned. This "OK" can sometimes cause problems since it will also be sent to the BASIC Stamp. Another problem is to ensure the buffer is empty of other data, especially when using RTS, so that when the value is returned, it is accepted correctly. Finally, when going into AT Command Mode the guard times make this a slow process. By lowering the guard times we can greatly speed up the process of entering Command Mode.

Getting_dB_Level.bs2 is optimized to accept values, to request the RSSI dBm level, and display data.

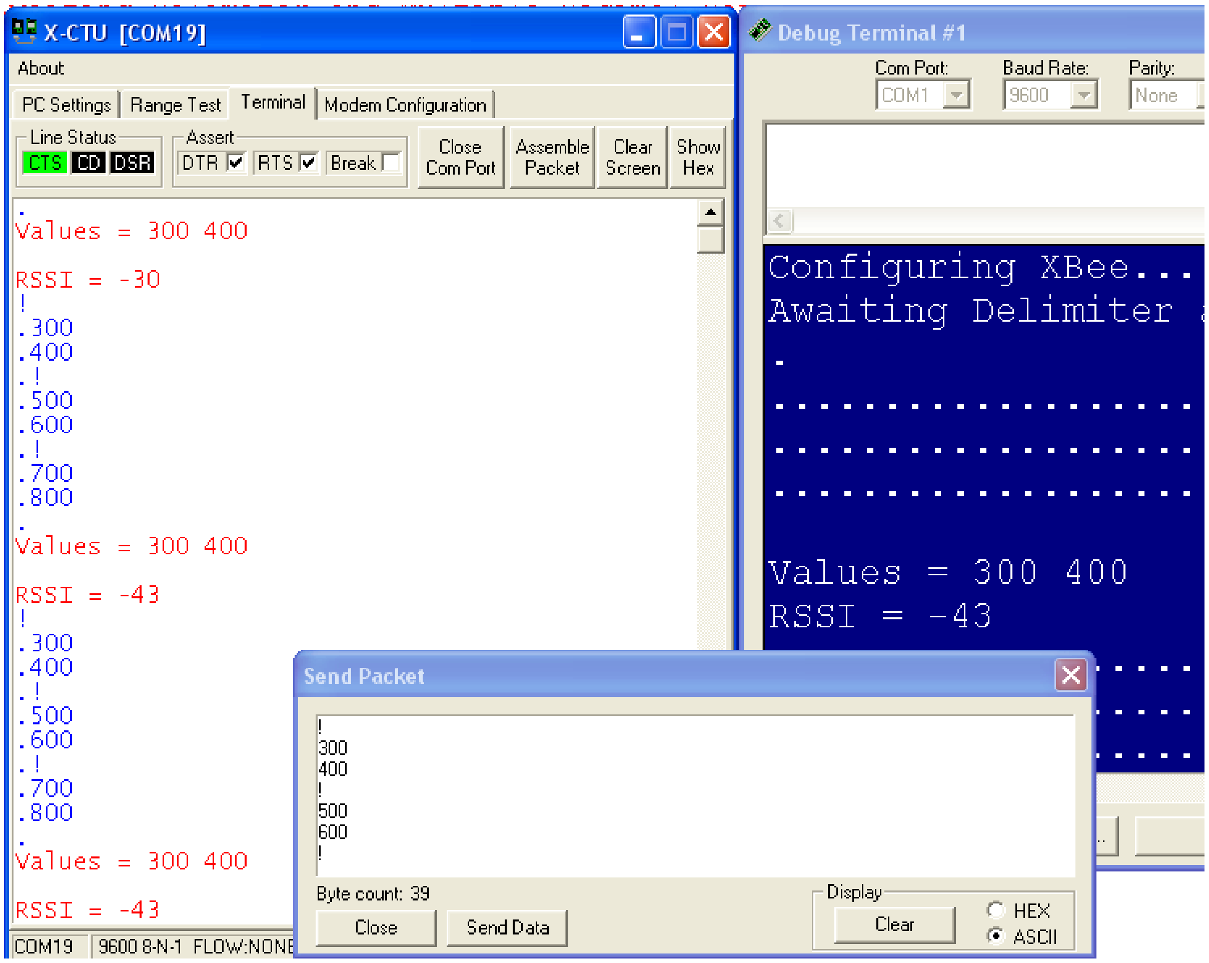
```
' ***********************************************************
' Getting_dB_Level.bs2
' Receive multiple decimal data and report dBm level
' ***********************************************************

' {$STAMP BS2}
' {$PBASIC 2.5}
' *************** Constants & PIN Declarations ***********
#SELECT $STAMP
  #CASE BS2, BS2E, BS2PE
    T9600       CON     84
  #CASE BS2SX, BS2P
    T9600       CON     240
  #CASE BS2PX
    T9600       CON     396
#ENDSELECT
Baud            CON     T9600

Rx              PIN     15
Tx              PIN     14
RTS             PIN     11

' ************** Variable Declarations ******************
DataIn          VAR     Byte
Val1            VAR     Word
Val2            VAR     Word
' ************** XBee Configuration *********************
PAUSE 500
DEBUG "Configuring XBee...",CR
PAUSE 2000                         ' Guard Time
SEROUT Tx,Baud,["+++"]             ' Command mode sequence
PAUSE 2000                         ' Guard Time
SEROUT Tx,Baud,["ATD6 1,GT3,CN",CR] ' RTS enable (D6 1)
                                   ' Very low Guard Time (GT 3)
                                   ' Exit Command Mode (CN)
' ************** Main LOOP ******************************
PAUSE 500
DEBUG "Awaiting Delimiter and Multiple Decimal Data...",CR

DO

  SERIN Rx\RTS,Baud,5,Timeout,[DataIn]    ' Briefly wait for delimiter

  IF DataIn = "!" THEN                    ' If delimiter, get data
    SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val1]  ' Accept first value
    SERIN Rx\RTS,Baud,3000,Timeout,[DEC Val2]  ' Accept next value
                                          ' Display remotely and locally
    SEROUT Tx, Baud, [CR,"Values = ", DEC Val1," ", DEC Val2,CR]
    DEBUG CR,"Values = ", DEC Val1," ", DEC Val2,CR

    GOSUB Get_dBm                         ' Retrieve RSSI level
  ENDIF

  GOTO Done                               ' Jump to done
  Timeout:
    DEBUG "."                             ' If no data, display dots
  Done:
LOOP

' ************** Subroutines ***************************
Get_dBm:
  GOSUB Empty_Buffer            ' Ensure no data left in XBee
  PAUSE 5                       ' Short guard time
  SEROUT Tx,Baud,["+++"]        ' Command Mode
  PAUSE 5                       ' Short guard time
  SEROUT Tx,Baud,["ATDB,CN",CR]  ' Request dBm Level (ATDB)& Exit
                                ' Accept returning HEX data with timeout
  SERIN Rx\RTS,Baud,50,TimeOut_dB,[HEX DataIn]
  SEROUT Tx,Baud,[CR,"RSSI = -",DEC DataIn,CR]  ' Display remotely
  DEBUG "RSSI = -",DEC DataIn,CR                ' Display locally
  TimeOut_dB:
RETURN

Empty_Buffer:                        ' Loop until no more data
  SERIN Rx\RTS,Baud,5,Empty,[DataIn]  ' Accept, when no data exit
  GOTO Empty_Buffer                  ' Keep accepting until empty
  Empty:
RETURN
```

Code analysis:

- In the XBee Configuration, **GT 3** (ATGT 3) is used to lower the guard time down to 3 ms.
- After accepting data, the **`Get_dBm`** subroutine is executed, which branches to **`Empty_Buffer`** to loop accepting data until the buffer is empty.
- **`Get_dBm`** then sends the +++ command sequence with very short delays.
- **ATDB** is sent to request the RSSI level, which the XBee returns and is accepted using **`SERIN`** as a hexadecimal value. It is then displayed locally and remotely.
- Note that any buffered data will be lost.
- The use of the string delimiter, !, will act also to filter out any unwanted data from being processed as well.

Testing:

✓ Download **Getting_dB_Level.bs2**.
✓ Enter several values, the ! with **Enter**, then several more values. After entering !, the next two values should be accepted and displayed along with the RSSI value as shown in Figure 4-8.
✓ Use the X-CTU Send Packet window to make a series of numbers and !'s to be sent and test. Notice that data is lost in this instance.



```
.
Values = 300 400

RSSI = -30
!
.300
.400
.!
.500
.600
.!
.700
.800
.
Values = 300 400

RSSI = -43
!
.300
.400
.!
.500
.600
.!
.700
.800
.
Values = 300 400

RSSI = -43
```

```
Configuring XBee...
Awaiting Delimiter
.
....................
....................
....................

Values = 300 400
RSSI = -43
```

```
!
300
400
!
500
600
!
```

Figure 4-8: Getting dB Level Test

General BASIC Stamp Notes of Interest

- The X-CTU terminal window is useful for sending data both as individual characters and as an assembled packet, but the BASIC Stamp Editor may also be used to interface to the XBee via USB. Simply open a new Debug Terminal and select the correct COM port, though it only supports individual character entry.
- Most any BASIC Stamp program that uses the Debug Terminal, for display or interaction, may be modified to use a remote Debug Terminal across the XBee transparent link. Replace any **DEBUG**'s with **SEROUT** structures and **DEBUGIN**'s with **SERIN** structures.
- A second BASIC Stamp and XBee may be used to send the data without worrying about receiving the data for display since the remote XBee/BASIC Stamp will display in data in the Debug Terminal. Use the code structure from the sample for baud rate and pins and send data with code such as:

```
DO
  SEROUT Tx,Baud,["!",CR,          ' Send string of data
         DEC 100,CR,
         DEC 200,CR]
  PAUSE 1000                       ' Short delay
LOOP
```

- Receiving serial data can be tricky and may require unique 'tweaks' to get the data received properly. An option may be to slow down the data by using a slower data rate from the XBee. For example, using **ATDB 0** will set the baud rate to 1200 allowing the BASIC Stamp more time to process incoming data. The best means is to use X-CTU to manually change the baud rate of the XBee and modify your code accordingly. If done solely in code you will need to start at 9600, change the XBee configuration to the new baud rate, then use the slower baud rate from then on. Also, see the next note…
- While configuring the XBee in code is nice, it can lead to issues. Let's say you add code to change the baud rate and it begins to communicate at 1200 after configuration code. When you download new code, the XBee will still be at 1200 baud and your configuration information at 9600 will not be understood (especially an issue if you changed the configuration!). Either manually cycling power on the board to reset or using the XBee Reset line (brought **LOW** to reset before configuring) to return to default configurations may be needed. While the BASIC Stamp resets with a code download, the XBee does not!
- The **SERIN** instruction has a **WAIT** modifier to idle until a character is received and allows multiple values to be accepted, such as:

  ```
  SERIN Rx\RTS,BAUD [WAIT ("!"), DEC Val1, DEC val2].
  ```

  We found that complex structure was worse at collecting data correctly than our 3-line method. Also, timeouts cannot be used with the **WAIT** modified. The **SERIN** instruction has many options. Please see the BASIC Stamp Help files for more information on **SERIN**.

Propeller Chip

The Propeller Chip is an excellent microcontroller for handling communications. Between the Propeller's speed, multi-core processing, and available objects, serial data communications is nearly effortless.

Hardware & Software

For these tests a single Propeller chip will be used for communications with an XBee connected to the PC using USB and X-CTU software as the base node. Data will be manually entered to be transmitted to the controller to investigate data reception.

Hardware:

- Propeller chip and development board, using a 5 MHz external crystal oscillator
- 2 XBee modules (Ensure they are restored to default configuration)
- An XBee Adapter Board
- XBee USB Adapter Board & Cable
- (2) 220 Ω resistors and (2) LEDs (optional)

Alternative to USB Adapter

Instead of an XBee on a USB Adapter, a Propeller & XBee using Serial_Pass_Through.spin discussed in this section or the PropPlug as discussed in Chapter 2 may be used for X-CTU communications and configuration. These configurations cannot be used for firmware updates for the XBee.

✓ Connect the Propeller to an XBee Adapter Board as shown in Figure 4-9. LEDs and resistors are optional but are useful.

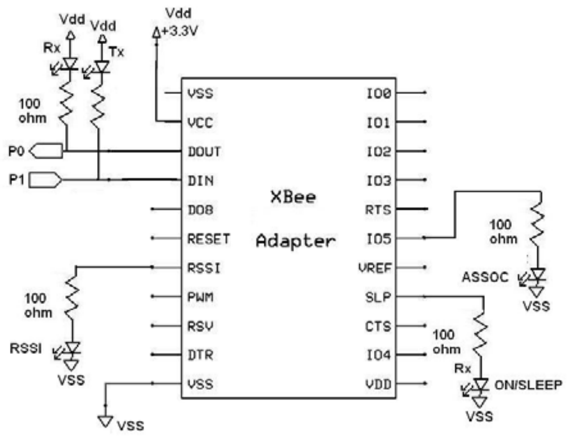


Figure 4-9: Propeller to XBee Adapter Board Interface with Optional LEDs

Software

- Propeller Tool Software (from the Downloads link at www.parallax.com/propeller)
- Digi's X-CTU software (www.parallax.com/go/xbee)

Receiving Byte Values & Chat

The ability to use multiple cogs in multitasking allows the Propeller to accept and process data in one cog while performing operations in another cog. In this example, the Propeller is being used for passing serial data between the PC and XBee. Data from the XBee is accepted and passed to the PC in one cog, and accepted from the PC and passed to the XBee in another cog as illustrated in Figure 4-10.

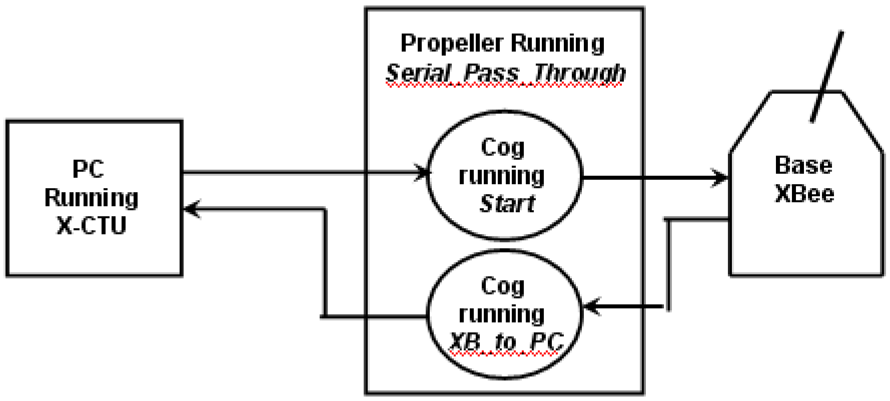


Figure 4-10: Cogs Passing Serial Data

```
{{
   ***************************************
   * Serial_Pass_Through                 *
   ***************************************
   *  See end of file for terms of use.  *
   ***************************************

   Provides serial pass through for XBee (or other devices)
   from the PC to the device via the Propeller. Baud rate
   may differ between units though FullDuplexSerial can
   buffer only 16 bytes.

}}

CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

  ' Set pins and Baud rate for XBee comms
  XB_Rx     = 0    ' XBee DOUT
  XB_Tx     = 1    ' XBee DIN
  XB_Baud   = 9600

  ' Set pins and baud rate for PC comms
  PC_Rx     = 31
  PC_Tx     = 30
  PC_Baud   = 9600

Var
  long stack[50]                ' stack space for second cog

OBJ
  PC    : "FullDuplexSerial"
  XB    : "FullDuplexSerial"

Pub Start

  PC.start(PC_Rx, PC_Tx, 0, PC_Baud) ' Initialize comms for PC
  XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee
  cognew(XB_to_PC,@stack)       ' Start cog for XBee--> PC comms

  PC.rxFlush                    ' Empty buffer for data from PC
  repeat
    XB.tx(PC.rx)                ' Accept data from PC and send to XBee

Pub XB_to_PC

  XB.rxFlush                    ' Empty buffer for data from XB
  repeat
    PC.tx(XB.rx)                ' Accept data from XBee and send to PC
```

Analyzing the code:

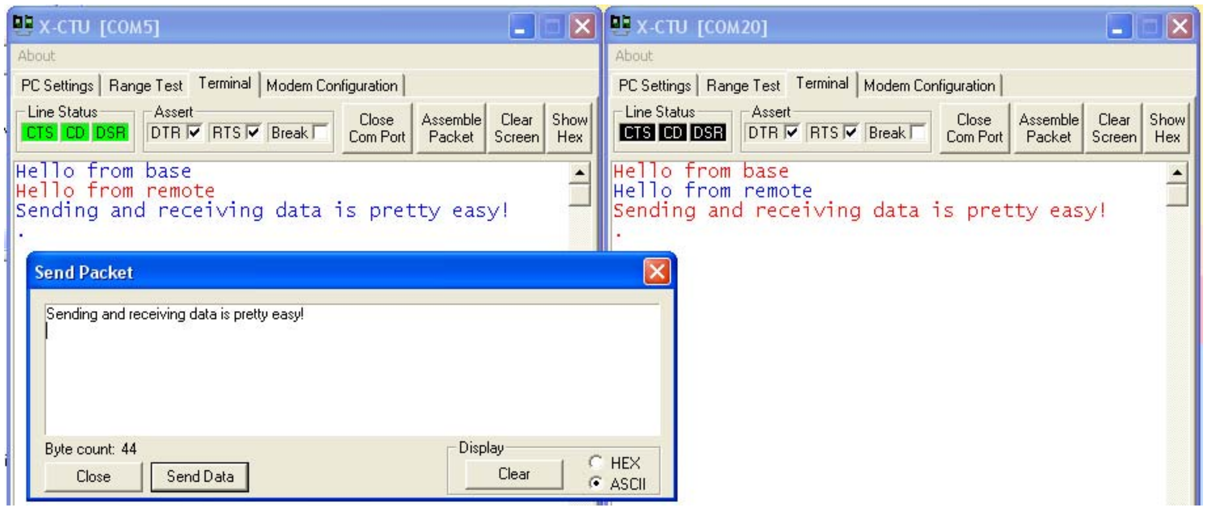
- As mentioned, two cogs are used with one running the `Start` method, and one running the `XB_to_PC` method.
- The **FullDuplexSerial.spin** object is used for serial communications. This object buffers 16 bytes allowing it to receive data before being passed to the code when requested with the `.rx` method. Using this method, execution will wait until a byte is available and returned before continuing.
- In `Start`, a byte from the PC is requested and passed to the XBee using `XB.tx(PC.rx)`. This is equivalent to accepting a byte to a variable, then sending the byte:

  ```
  DataIn := PC.Rx
  XB.Tx(DataIn)
  ```

  The benefit in not storing it before transmission is the lower processing time needed increasing the execution speed.
- In the `XB_to_PC` method, data is accepted from the XBee and passed to the PC, allowing data to flow in both directions at the same time using two cogs—full duplex communications.

Testing:

- ✓ You may use:
  - o One XBee on a USB Adapter and one connected to the Propeller running **Serial_Pass_Through.spin**.
    —OR—
  - o 2 Propellers connected to XBee modules running **Serial_Pass_Through.spin**.
- ✓ Download the code to controller(s).
- ✓ Open two instances of X-CTU; one using the COM port of the base unit, and one connected to the remote XBee (separate PC's may be used).
- ✓ Type in each window to send data to the other.
- ✓ Test using the Send Packet window as well as shown in Figure 4-11.



```
Hello from base
Hello from remote
Sending and receiving data is pretty easy!
```

Sending and receiving data is pretty easy!

Figure 4-11: Serial Pass Through Chatting

- ✓ Using a Propeller connected XBee, open the **Modem Configuration** tab and verify that you can load and save configuration settings via the Propeller.

COM Port Use: Only one device can use the Com Port at any time. When programming the Propeller, the X-CTU software must be disconnected by using Close Com Port. The Propeller Serial Terminal (PST) program may also be used.

Propeller Resets: When using the terminal window to interface to the Propeller, it is recommended that you use F11 to download instead of F10. This will ensure your code is in memory if the Propeller cycles on terminal window connections.

Note that the serial speed of the XBee and PC do not need to be the same. Different speeds may be used between each side of the Propeller. Communications with the XBee can be handled by the code up to its maximum speed.

Of course, in this example we are simply passing bytes, though code may be written to use the byte values, or as data for processing based on its value:

```
DataIn := XB.rx
If DataIn == "p"
   ' Code to be processed
```

Debugging Back to Base Unit

Using the XBee is a great way to simply monitor your remote unit for robotics or sensor data. In this example we will use the `.str` and `.dec` methods of the FullDuplexSerial object in **Simple_Debug.spin** to send information back to the base for monitoring.

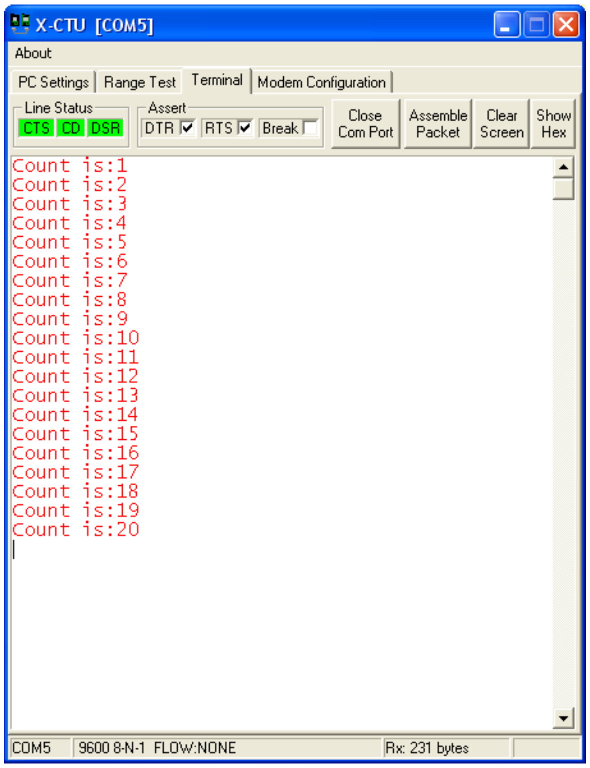
```
{{
   ***************************************
   * Simple_Debug                        *
   ***************************************
   *  See end of file for terms of use.  *
   ***************************************

   Demonstrates debugging to remote terminal

}}

CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

  ' Set pins and Baud rate for XBee comms
  XB_Rx     = 0    ' XBee DOUT
  XB_Tx     = 1    ' XBee DIN
  XB_Baud   = 9600
  CR        = 13   ' Carriage Return value

Var
  word stack[50]

OBJ
   XB    : "FullDuplexSerial"

Pub  Start | Counter
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee

Delay (1000)                       ' one second delay
repeat Counter from 1 to 20        ' count up to 20

  ' Send to Base
  XB.str(string("Count is:"))      ' send string
  XB.dec(Counter)                  ' send decimal value
  XB.Tx(CR)                        ' send Carriage Return

  Delay(250)                       ' Short delay

Pub Delay(ms)       ' Delay in milliseconds
   waitcnt(clkfreq / 1000 * ms + cnt)
```

Analyzing the code:

- The FullDuplexSerial object is once again used to interface to the XBee, but in this case the local interfacing to the PC in not performed.
- The value of `Counter` is incremented from 1 to 20, a string is sent to the base using `XB.str` method, and the `Counter` value is sent using `XB.dec` method.
- The line is terminated by sending the byte value of 13 for a carriage return (`CR`).

Testing:

- ✓ Connect the base XBee to the PC using USB Adapter board, or a Propeller with **Serial_Pass_Through.spin,** and X-CTU terminal window.
- ✓ Set up the remote XBee with a Propeller running **Simple_Debug.spin**.
- ✓ Download and run the program, monitor the base terminal window as shown in Figure 4-12.



```
Count is:1
Count is:2
Count is:3
Count is:4
Count is:5
Count is:6
Count is:7
Count is:8
Count is:9
Count is:10
Count is:11
Count is:12
Count is:13
Count is:14
Count is:15
Count is:16
Count is:17
Count is:18
Count is:19
Count is:20
```

COM5 | 9600 8-N-1 FLOW:NONE | Rx: 231 bytes

Figure 4-12: Simple Debug Monitoring

Receiving Simple Decimal Values

The FullDuplexSerial object can also receive decimal values (as well as hexadecimal) that are terminated with a carriage return. In this section we will look at code to accept and return decimal values using **Simple_Decimal_Receieve.spin**. This code requires the **XBee_Object.spin** object by Martin Hebel which is available from the Propeller Object Exchange (http://obex.parallax.com) or in the distributed files for this tutorial. The **XBee_Object** uses **FullDuplexSerial**, but greatly extends it with functions, many specific to XBee interfacing.

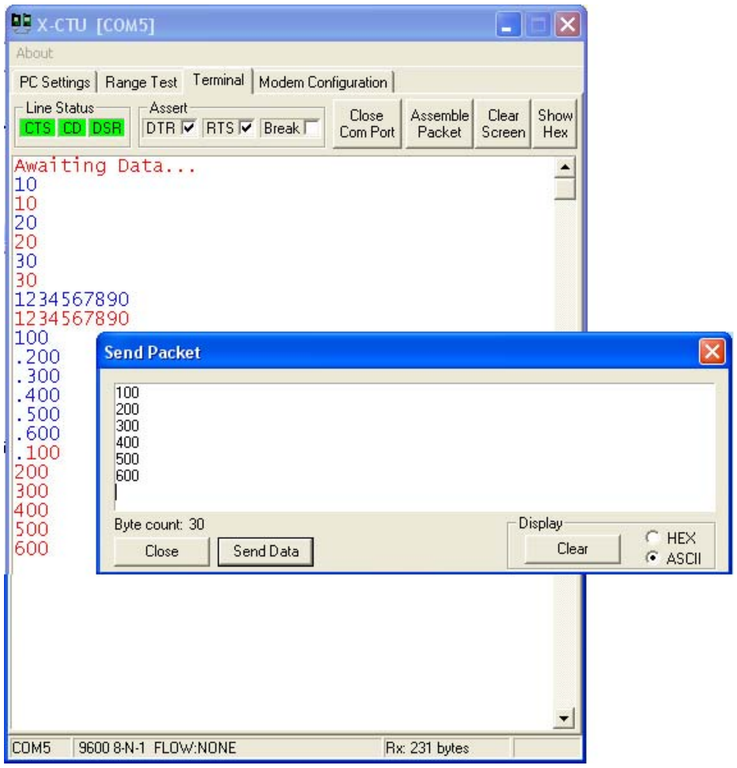
```
{{
   ***************************************
   * Simple_Decimal_Receive              *
   ***************************************
   *  See end of file for terms of use.  *
   ***************************************

   Demonstrates receiving and echoing decimal value

}}

CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000
  ' Set pins and Baud rate for XBee comms
  XB_Rx     = 0    ' XBee DOUT
  XB_Tx     = 1    ' XBee DIN
  XB_Baud   = 9600


OBJ
   XB    : "XBee_Object"

Pub Start | Value
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee

XB.Delay(1000)                     ' One second delay

XB.str(string("Awaiting Data...")) ' Notify base
XB.CR

Repeat
  Value := XB.RxDec                ' Wait for and accept decimal value
  XB.Dec(Value)                    ' Send value back to base
  XB.CR                            ' Send carriage return
```

Analyzing the Code:

- In this example, only one terminal window is needed – the one with the base XBee using a USB Adapter or Propeller running **Serial_Pass_Through.spin** for PC communications.
- The XBee_Object is used for XBee communications and interfacing providing additional functionality.
- In the `Start` method, the base XBee is informed the remote is awaiting data and waits for a decimal value to arrive using the `XB.RxDec` method. The decimal value needs to be terminated with a carriage return (enter key) or by a comma to separate values.
- Once a decimal value is received and stored in `Value`, it is sent back to the base XBee as a decimal value with `XB.Dec(Value)` along with a carriage return.
- Note that the **XBee_Object** has methods for both `.Delay` and `.CR` to minimize coding for normal needs.

Testing:

- ✓ Base XBee connected to PC using USB adapter or Propeller with **Serial_Pass_Through.spin** and X-CTU terminal Window.
- ✓ Remote XBee with Propeller running **Simple_Byte_Receive.spin**.
- ✓ Test sending decimal value in the X-CTU terminal window and the Send Packet window using carriage returns between values as shown in Figure 4-13. Note that no data is lost due to the speed of the Propeller and the buffering of the drivers.



```
Awaiting Data...
10
10
20
20
30
30
1234567890
1234567890
100
.200
.300
.400
.500
.600
.100
200
300
400
500
600
```

```
100
200
300
400
500
600
```

Figure 4-13: Receiving Simple Decimal Values

Receiving Multiple Decimal Values with Delimiter

In many instances you may need to send multiple values to the Propeller for various operations, such as the speed of both motors on a robot. While the Propeller is great at buffering and receiving the data, it may require some extra code to ensure the data is sent in the correct sequence, just as with the BASIC Stamp. If we send 2 values for `val1` and `val2`, such as:

10 & 20
10 & 30
10 & 50

On reception, the Propeller gets out of sequence so that the data it receives is:

20 & 10
30 & 10

Because it perhaps missed one, the sequence of how it is accepting data does not match the sequence we intended. Through the use of a start-of-string identifier or delimiter we can help ensure the data is collected starting at the correct value in the buffer. The same could be done for bytes but it can cause issues since a byte value can be ANY value typically, 0 to 255. A unique value may not be able to be identified for our communications. Using `DEC` values, only 0–9 are valid characters so anything outside of that range would be unique from our data. The program **Multiple_Data_Receive.spin** illustrates this as well as some other features.

```
{{
   ****************************************
   * Mulitple_Decimal_Receive             *
   ****************************************
   *  See end of file for terms of use.   *
   ***************************************

   Demonstrates receiving multiple decimal
   value with start delimiter

}}

CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

  ' Set pins and Baud rate for XBee comms
  XB_Rx     = 0    ' XBee DOUT
  XB_Tx     = 1    ' XBee DIN
  XB_Baud   = 9600

  ' Carriage return value
  CR = 13

OBJ
   XB    : "XBee_Object"

Pub  Start | DataIn, Val1,Val2
XB.start(XB_Rx, XB_Tx, 0, XB_Baud) ' Initialize comms for XBee

XB.Delay(1000)                     ' One second delay

XB.str(string("Awaiting Data...")) ' Notify base
XB.CR

Repeat
  DataIn := XB.RxTime(100)         ' Wait for byte with timeout
  If DataIn == "!"                 ' Check if delimiter
    Val1 := XB.RxDecTime(3000)     ' Wait for 1st decimal value with timeout
    Val2 := XB.RxDecTime(3000)     ' Wait for next decimal value with timeout
    If Val2 <> -1                  ' If value not received value is -1
      XB.CR
      XB.Str(string(CR,"Value 1 = ")) ' Display remotely with string
      XB.Dec(Val1)                    ' Decimal value
      XB.Str(string(CR,"Value 2 = ")) ' Display remotely
      XB.Dec(Val2)                    ' Decimal value
      XB.CR
  Else
    XB.Tx(".")                     ' Send dot to show actively waiting
```

Analyzing the code:

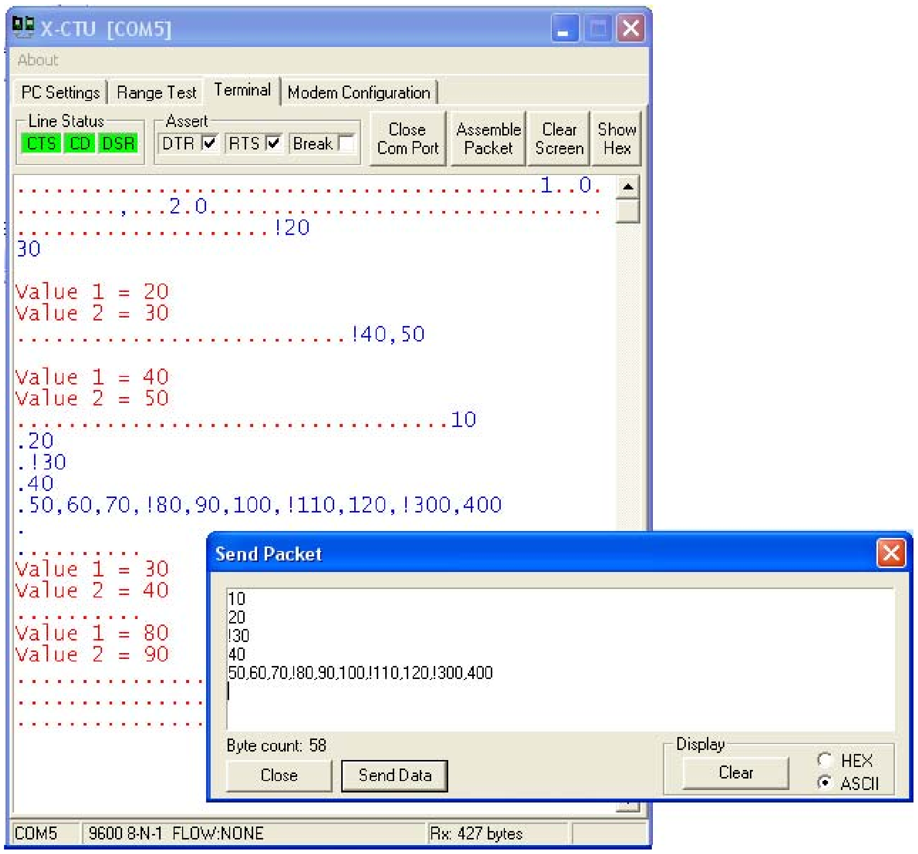
- The XBee_Object is used for communications and interfacing.
- In this example**,** `DataIn := XB.RxTime(100)` reads a byte from the buffer, but only waits 100 ms for data arrive before continuing processing – a receive with timeout. If no data is received, the value in **DataIn** will be -1.
- Upon reception or timeout, the value of **DataIn** is check to see if it the string delimiter, !. If it is, 2 decimal values, with 3 second timeouts, are accepted into **Val1** and **Val2**.
- Identifying strings and decimal values are sent back to the base unit for display.
- Upon timeout with a value of -1, a dot is sent to the remote terminal to indicate processing is continuing.

Value Delimiter

The XBee_Object can use carriage returns or commas as delimiters between values, so data may be sent as 30,40 instead of using the enter key between each.

Testing:

- ✓ Base XBee connected to PC using USB adapter or Propeller with **Serial_Pass_Through.spin** and X-CTU terminal window.
- ✓ Remote XBee with Propeller running **Multiple_Decimal_Receive.spin**.
- ✓ Test by sending decimal value in the terminal window by entering some values, using !, then entering a few more values—DO not press Enter after !. Test using commas as well.
- ✓ Use the Send Packet window to build a packet of values and using the ! delimiter. Again, test the use of carriage returns and commas between values as shown in Figure 4-14.



```
........................................1..0.
........,...2.0.............................
...................!20
30

Value 1 = 20
Value 2 = 30
.........................!40,50

Value 1 = 40
Value 2 = 50
.................................10
.20
.!30
.40
.50,60,70,!80,90,100,!110,120,!300,400
.
.........
Value 1 = 30
Value 2 = 40
.........
Value 1 = 80
Value 2 = 90
..............
..............
..............
```

```
10
20
!30
40
50,60,70,!80,90,100,!110,120,!300,400
```

Byte count: 58

COM5 9600 8-N-1 FLOW:NONE Rx: 427 bytes

Figure 4-14: Multiple Decimal Receive Testing

Notice that there was some lost data—the !300, 400 was not displayed. The buffer of the object may have been exceeded while processing. When dealing with serial communications, testing and work-arounds are always needed, but we have a high probability of getting simple packets through successfully at normal update speeds.

Setting and Reading XBee Configurations

Just as we can configure the XBee through the serial terminal, the Propeller can send and receive data for configuration changes and reading values in the same fashion. **Config_Getting_dB_Level.spin** demonstrates a simple configuration of the XBee and requesting and accepting dB level for RSSI. Since we don't need to set RTS with the Propeller and we are not ready for address changes yet, we will use a configuration to turn off the association indicator on the XBee.

```
{{
   ***************************************
   * Config_Getting_dB_Level             *
   ***************************************
   *  See end of file for terms of use.  *
   ***************************************

   Demonstrates receiving multiple decimal
   value with start delimiter

}}

CON
  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

  ' Set pins and Baud rate for XBee comms
  XB_Rx     = 0    ' XBee DOUT
  XB_Tx     = 1    ' XBee DIN
  XB_Baud   = 9600

  ' Carriage return value
  CR = 13

OBJ
   XB    : "XBee_Object"

Pub  Start | DataIn, Val1,Val2
XB.start(XB_Rx, XB_Tx, 0, XB_Baud)      ' Initialize comms for XBee
XB.Delay(1000)                          ' One second delay

' Configure XBee module
XB.Str(String("Configuring XBee...",13))
XB.AT_Init                             ' Configure for fast AT Command mode

XB.AT_Config(string("ATD5 4"))         ' Send AT command turn off Association LED

XB.str(string("Awaiting Data..."))     ' Notify base
XB.CR

Repeat
  DataIn := XB.RxTime(100)             ' Wait for byte with timeout
  If DataIn == "!"                     ' Check if delimiter
    Val1 := XB.RxDecTime(3000)         ' Wait for 1st value with timeout
    Val2 := XB.RxDecTime(3000)         ' Wait for next value with timeout
    If Val2 <> -1                      ' If value not received value is -1
      XB.CR
      XB.Str(string(CR,"Value 1 = "))  ' Display remotely with string
      XB.Dec(Val1)                     ' Decimal value
      XB.Str(string(CR,"Value 2 = "))  ' Display remotely
      XB.Dec(Val2)                     ' Decimal value

      XB.RxFlush                       ' Clear buffer
      XB.AT_Config(string("ATDB"))     ' Request dB Level
      DataIn := XB.RxHexTime(200)      ' Accept returning hex value
      XB.Str(string(13,"dB level = ")) ' Display remotely
      XB.Dec(-DataIn)                  ' Value as negative decimal
      XB.CR
  Else
    XB.Tx(".")                         ' Send dot to show actively waiting
```

Code Analysis:

- Just as when we manually configure the XBee, it requires a 2-second delay since last data sent, the +++ sequence, and a 2-second delay. This delay may be reduced by setting the guard time lower (ATGT). The XBee object has a method called `.AT_Init` which will perform this sequence while lowering guard time to allow fast configurations. Looking at the **XBee_Object** code, we can see the actions taken. The `.rxFlush` method is used to clear data from the object buffer along with the OK's that are returned.

```
Pub AT_Init
{{
    Configure for low guard time for AT mode.
    Requires 5 seconds.  Required if AT_Config used.
}}

    delay(3000)
    str(string("+++"))
    delay(2000)
    rxflush
    str(string("ATGT 3,CN"))
    tx(13)
    delay(500)
    rxFlush
```

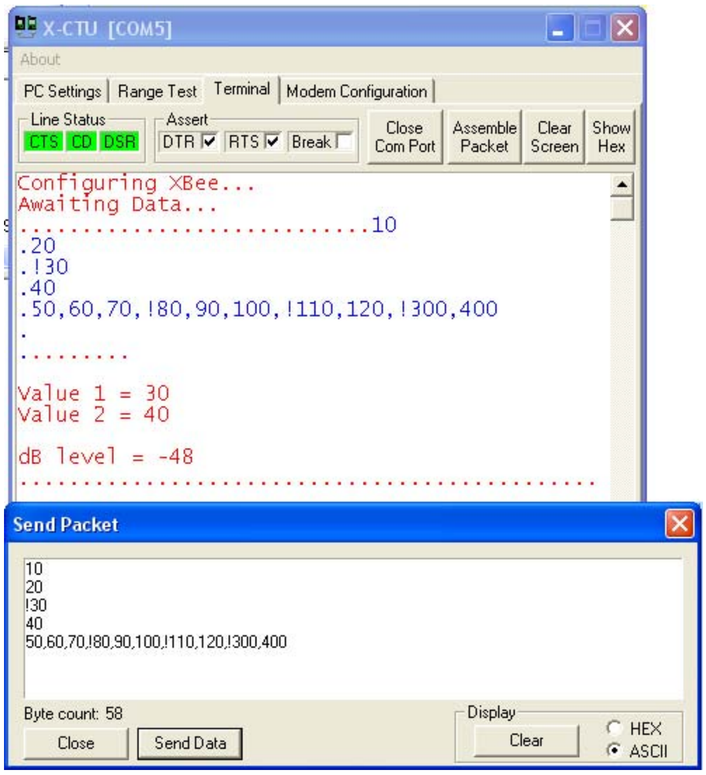
- After `AT_Config`, the ASSOC indicator LED on the XBee Adapter (if connected) is disabled using `XB.AT_Config(string("ATD5 4"))`. Using fast guard times, the configuration of the XBee is quickly updated for D5 to be 4, setting the DIO5 output low.
- After receiving the delimiter and 2 decimal values, the code requests and shows the dB level. `XB.AT_Config(string("ATDB"))` is used to place the XBee into Configuration Mode and sendthe ATDB command (it also sends the CN to exit Command Mode). The returned hexadecimal value is returned and saved using

```
DataIn := XB.RxHexTime(200)
```

- The dBm level is displayed by sending it back to the base unit.
- Note that `RxFlush` is used to clear out the buffer when using the command mode to prevent buffered data from interfering. This means that not all of our burst data will be processed.

Testing:

- ✓ Base XBee connected to PC using USB adapter or Propeller with **Serial_Pass_Through.spin** and X-CTU terminal Window.
- ✓ Remote XBee with Propeller running **Config_Getting_dB_Level.spin**.
- ✓ Test using the ! delimiter and values. Note that the dBm level is returned after a good set of data and remaining data is lost as shown in



```
Configuring XBee...
Awaiting Data...
............................10
.20
.!30
.40
.50,60,70,!80,90,100,!110,120,!300,400
.
.........

Value 1 = 30
Value 2 = 40

dB level = -48
..............................................
```

```
10
20
!30
40
50,60,70,!80,90,100,!110,120,!300,400
```

Byte count: 58

DTR [✓] RTS [✓] Break [ ]

Display: HEX [✓ ASCII]

Figure 4-15: Configuring and Reading dBm Level Test

Config with Variables:

Using `AT_Config`, a complete string is required. If a variable value is needed, use `AT_ConfigVal` method where a string and value are passed: `XB.AT_ConfigVal(string("ATDL"), DL_Val)`.

Monitor LEDs:

When the code starts and goes through the configuration sequence, watching the Tx & Rx LEDs blink back and forth is a great indicator that it the XBee is accepting your command mode functions.

While configuring the XBee in code is nice, it can lead to issues. Let's say you add code to change the baud rate and it begins to communicate at 1200 after configuration code. When you download new code, the XBee will still be at 1200 baud and your configuration information at 9600 will not be understood (especially an issue if you changed the configuration!). Either manually cycling power on the board to reset or using the XBee Reset line (brought LOW to reset before configuring) to return to default configurations may be needed. While the Propeller resets with a code download, the XBee does not!

Summary

The BASIC Stamp and Propeller can both be programmed to receive data, bytes or decimal values. With the BASIC Stamp, the use of RTS can help ensure data is not missed. With both controllers, XBee configuration may be performed by both controllers using the XBee's AT Command Mode to change settings or to request data such as dBm level for RSSI. While an XBee and terminal window were used, in upcoming chapters two controllers will be used to interact with one another using XBees.