

APPLICATION NOTE

Interfacing Serial Sensors with Campbell Scientific Dataloggers



Interfacing Serial Sensors with Campbell Scientific Dataloggers

With the introduction of our CR1000, CR3000, and CR800-series dataloggers, interfacing a variety of serial sensors became possible. Some sensors offer serial output as the only interface. Some smart sensors have analog output as an option, but provide better precision or additional outputs when using their serial output. This document is intended to help the reader interface their serial sensor, whatever it may be, to a Campbell Scientific datalogger.

1. General Information

1.1 Standard Protocols, Physical Specifications, and SDI-12

To have a sensor successfully interfaced with a datalogger, the communication protocol and physical specification have to match in both devices. In the computer world, a communication protocol is the language used by two devices to speak to each other. Morse code is the oldest digital communication protocol invented. There are a number of standard communication protocols today including Modbus and DNP3 among others. Modbus and DNP3 have special instructions in the datalogger, and these instructions should be used if the sensor supports it.

Many sensor manufacturers use their own protocol. If the protocol is known, the datalogger can be programmed to emulate it. However, writing your program can be time consuming. This document will help with implementing nonstandard serial protocols.

The physical communication specification is how the message gets from point A to point B. An example of a physical communication specification is RS-232 9600 8N1. Notice the 9600 8N1. If a sensor is RS-232 9600 8N1 and the datalogger port is set to RS-232 9600 7N1, they will not communicate. There are dozens of physical serial specifications. This is discussed in more detail later in this document.

SDI-12 was developed as a standard protocol and physical specification. Interfacing an SDI-12 sensor to a datalogger is documented in your datalogger's manual. This document does not address the use of the SDI-12 protocol.

1.2 Important Questions You Need Answered

There are several important questions you need answered about your serial sensor to be able to interface to your sensor. The answers to these questions are most likely found in the manual for the sensor. If you do not find the answers in the manual, you will need to contact the sensor manufacturer.

- What are the signaling level, baud rate, parity, stop bits, and data bits? (Needed for SerialOpen.)
- Does the sensor have to be polled (commands issued to sensor before it sends data)?
- What is the polling command?
- What is the default output string?
- Do I need to change the sensor configuration?
- How do I change the sensor configuration?
- What will the output string be after I change the configuration?

2. Programming Instructions

Following are brief descriptions of the CRBasic instructions used for serial sensors. Detailed descriptions on the usage of these instructions are in the help file of the CRBasic Editor. The brief descriptions in this document are meant to help the reader decide which instructions to add to their program.

2.1 SerialOpen (ComPort, BaudRate, Format, TXDelay, BufferSize)

The first instruction you need to put in your datalogger program is SerialOpen. The SerialOpen instruction sets up the serial port for communication.



This document will not go into detail on all the parameters of CRBasic instructions. Those are documented well in the help file of the CRBasic Editor.

If your sensor uses RS-422 or RS4-85, an interface such as the SDM-SIO1 will be needed. Refer to the SDM-SIO1 manual for additional format options beyond those specified in the CRBasic help.

The RS-232 port on the datalogger provides RS-232 voltage levels and logic. The control port pairs provide TTL voltage levels and either RS-232 or TTL logic. Most RS-232 sensors will work with the control ports. If an RS-232 sensor does not work on the control ports, it should be tried on the RS-232 port.

BaudRate and Format must be selected correctly based on the sensor specifications. TXDelay can usually be left at zero. BufferSize should be set to at least the length of the output string + 1. Too small a buffer will cut your data short. The buffers are ring memory, which means that the newest data overwrites the oldest data.

The BufferSize should be greater than the total number of bytes received during a scan interval. Allow for an extra record from the sensor to account for clock differences.

Example: A sensor outputs 39 bytes at a rate of 2 Hz

The datalogger scan rate is 10 seconds

$39 \text{ bytes/output} \times 2 \text{ outputs/sec} \times 10 \text{ seconds/scan} = 780 \text{ bytes received/scan}$

$780 \text{ bytes} + 39 \text{ bytes/output} = 819 \text{ bytes}$

BufferSize should be at least $820(819+1)$ bytes.

The datalogger performs two checks before reading data from the buffer (see Figure 1). The first check is if the position of the write pointer has changed since the last execution of `SerialIn`, `SerialInBlock`, or `SerialInRecord`. The second check is if the number of bytes between the read pointer and the write pointer is greater than or equal to the number of bytes expected.

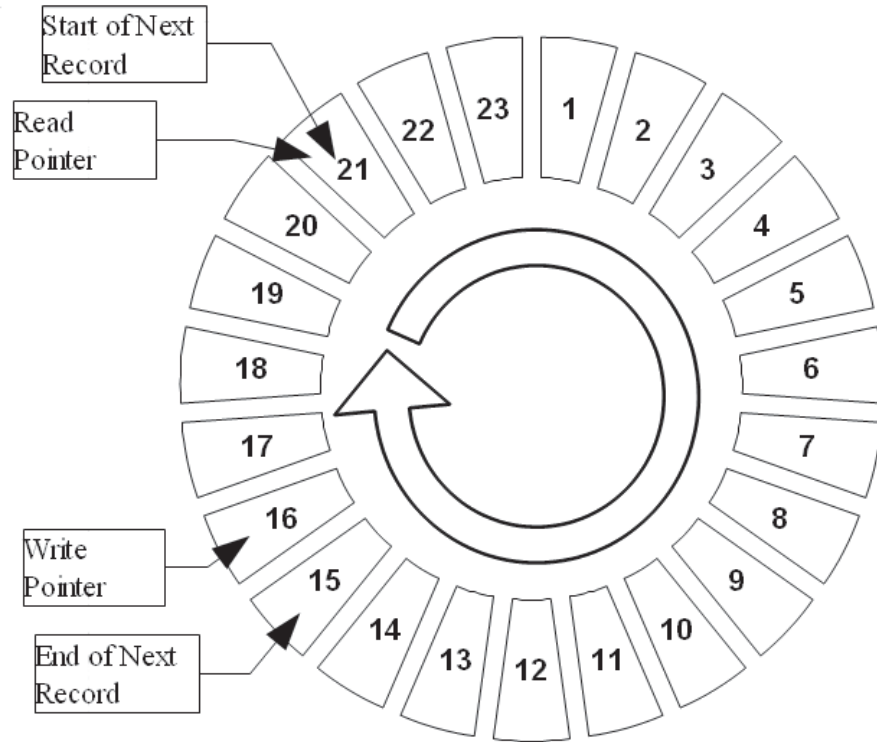


Figure 1. Representation of a 23 byte ring buffer that has received an 18 byte record since last being read

2.2 SerialClose(ComPort)

`SerialClose` is the opposite of `SerialOpen`. It stops communication on a port and frees up memory allocated for the buffer. Most programs have no need for it, but it is available.

2.3 SerialFlush(ComPort)

`SerialFlush` effectively erases the data in the buffer for a `ComPort`. The write pointer and the read pointer are both placed at the beginning of the buffer. Unlike `SerialClose`, `SerialFlush` leaves the port active and the buffer remains allocated for incoming data.

2.4 ComPortIsActive(ComPort)

This function checks for activity on a port.

2.5 SerialInChk(Comport)

The SerialInChk function reports the number of bytes in a buffer between the read and write pointers.

2.6 SerialInBlock(ComPort, Dest, MaxNumberBytes)

SerialInBlock will directly copy data up to the MaxNumberBytes setting from a port's buffer into a destination variable. This instruction is for advanced users only. Data containing null characters can only be read from a buffer with this instruction.

2.7 SerialIn(Dest, ComPort, TimeOut, TerminationChar, MaxNumChars)

SerialIn will read data out of a buffer and wait for more data if needed. A TimeOut of 0 is not recommended. It could make your datalogger program wait indefinitely if your serial sensor is not connected. SerialIn will read data up to the point of the termination character or the maximum number of characters has been received.

2.8 SerialInRecord(ComPort, Dest, BeginWord, NBytes, EndWord, NBytesReturned, LoadNaN)

SerialInRecord has replaced the SerialIn instruction for most applications. One benefit of the SerialInRecord instruction is that with some parameters set constant, it will run in the datalogger measurement task. This allows better timing and faster execution in some programs. The second benefit of the SerialInRecord instruction is it can recognize when output from a sensor is a complete message. The BeginWord and EndWord parameters are integers. Hexadecimal notation is the easiest manner of entering 2 characters for either one.

2.9 SerialOut(ComPort, OutString, WaitString, NumberTries, TimeOut)

SerialOut is used for outputting a string on one of the datalogger's ports.

2.10 SerialOutBlock(ComPort, Expression, NumberBytes)

SerialOutBlock is used if you need to send binary data out a port. Under proper conditions, this instruction will run in the measurement task of the datalogger in pipeline mode.

2.11 SplitStr(SplitResult, SearchString, FilterString, NumSplit, SplitOption)

The SplitStr instruction is used to divide data from a long string into smaller strings. The original string remains intact. SplitStr should generally be used instead of MoveBytes. Most serial sensor programs will use SplitStr with either the NUMERIC or FOOTERFILTER option.

2.12 CheckSum(ChkSumString, ChkSumType, ChkSumSize)

The CheckSum function is used for calculating the checksum of a string. Check sums are used for checking serial messages for transmission errors. The use of this instruction is recommended for advanced programmers only.

2.13 MoveBytes(Destination, DestOffset, Source, SourceOffset, NumBytes)

The MoveBytes instruction is used for moving binary data between memory locations. Manipulating binary data is an advanced topic and not covered in this document. Strings containing null characters must be split with MoveBytes instead of SplitStr.

3. Different Levels of Protocol Complexity

The simplest serial protocol scenario is a continuous output serial sensor connected to a datalogger (see Figure 2). The sensor initiates the communication and sends data to the datalogger. The datalogger has to be programmed only to receive the data and process it.

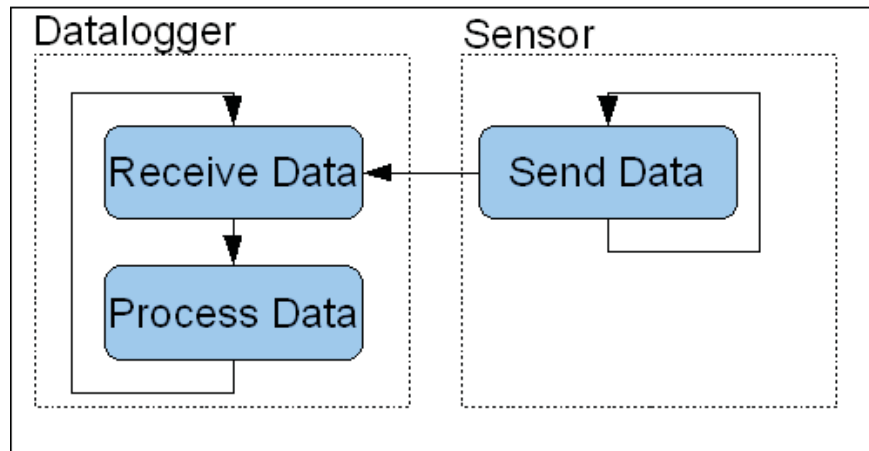


Figure 2. Flowchart of a continuous output serial sensor communicating with a datalogger.

The next level of complexity is that of a polled sensor (see Figure 3). The datalogger must send a command to the sensor, then receive the answer from the sensor.

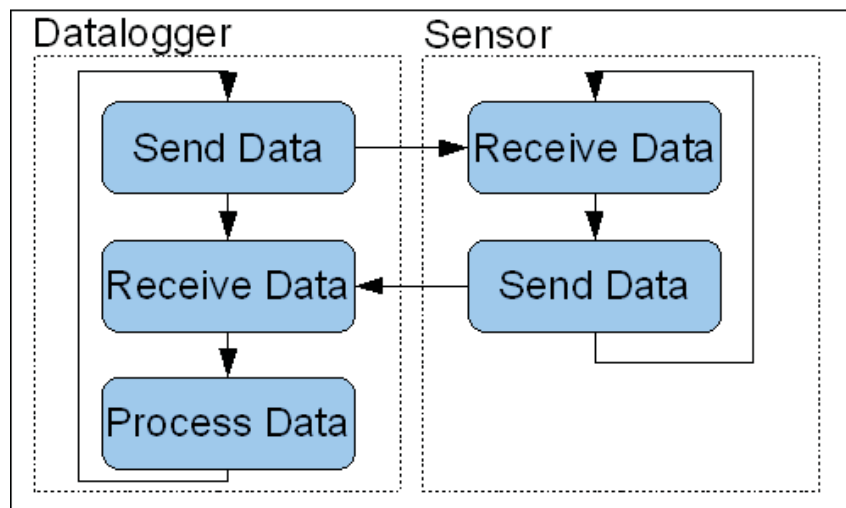


Figure 3. Flowchart of communications between a polled sensor and a datalogger.

An even higher level of complexity is when the datalogger must change the commands it sends to the other device based on received messages (see Figure 4). The other device could be a host device such as a PC or datalogger.

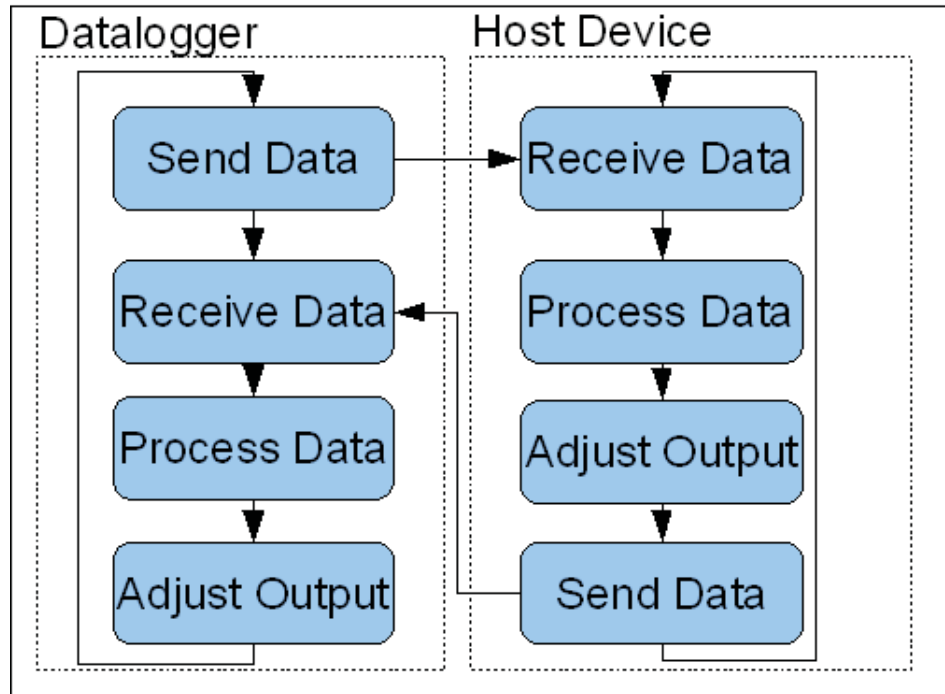


Figure 4. Flowchart of communications between a datalogger and host device.

4. Step-by-Step Programming Examples

For the programming examples that follow a CR1000 is used. The examples can be easily modified to work on a CR800, CR850, or CR3000 also. The equipment required for these examples is a CR1000, a power supply, and two short pieces of wire. Connect a piece of wire from C1 to C4 on the CR1000, and the other wire from C2 to C3. We will simulate the serial sensor on the datalogger itself at COM2. However, simulating a sensor on the same datalogger does not allow testing a programs behavior during communication errors or time asynchronization.

4.1 Continuous Output Sensor

With a continuous output serial sensor, data is continually sent to the datalogger. The datalogger needs to recognize the data it wants and pull those pieces from the buffer.

- Declare variables
- Set up communication port
- Read from buffer
- Split read data.
- Process split data.

For our example, we are going to use an imaginary sensor of these characteristics:

- TTL, Even Parity, one stop bit, 8 data bits, 4800 baud
- 1 Hz output rate
- Values output are; a floating point number, and integer, and a string.
- Example output: <STX>1234E+04,1234,acbd<CR><ETX>

The first step in writing a program for a serial sensor is declare variables. You will need a long string to hold the raw data from the sensor. You also need variables to hold processed data and program status.

Here are the variable declarations for our example:

- Public RawString as String * 30, SplitStrings(3) as String * 16, FloatValue, IntegerValue as Long
- Public NBytesReturned

RawString is our variable to hold raw data from the sensor. SplitStrings(3) are three shorter strings to hold pieces of the raw data. FloatValue and IntegerValue are variables to store the processed data in the format we want. NbytesReturned is a variable to hold a result from the SerialInRecord instruction.

The second step is to set up the communication port. The communication port is setup with the SerialOpen instruction. It is critical that the correct baud rate and format are chosen to match the sensor. According to the specifications of our sensor, we should choose Format option 18. Our example message is 21 bytes long. Expressions such as '<STX>' represent a single unprintable character, they count as one byte. The datalogger will be running at a 1-Hz scan rate which is the same as the output interval of the sensor. Referring to the equation for minimum buffer size for SerialOpen, the buffer needs to be at least 43 bytes. We will round up to a 50 byte buffer size. Below is our instructions inserted into the basic CR1000 program template. At this point, you should load the program into the datalogger and verify the sensor output with the 'W' terminal mode as described in Appendix B.

```
Public PTemp, batt_volt
Public RawString As String * 30, SplitStrings(3) As String * 16, FloatValue, IntegerValue
As Long
Public NBytesReturned
```

```
DataTable (Test,1,1000)
  DataInterval (0,15,Sec,10)
  Minimum (1,batt_volt,FP2,0,False)
  Sample (1,PTemp,FP2)
EndTable
```

```
BeginProg
  SerialOpen (Com1,4800,18,0,50)
  SerialOpen (Com2,4800,18,0,50)
  Scan (1,Sec,0,0)
  PanelTemp (PTemp,250)
  Battery (batt_volt)
  CallTable Test
NextScan
```

```
SlowSequence
Scan (1,Sec,3,0)
  SerialOut (Com2,CHR(2)&"1234E+04,1234,abcd"&CHR(13)&CHR(3),"",0,100)
NextScan
EndProg
```



The second SerialOpen instruction and the SlowSequence scan are added to emulate the sensor.

The next step is to read in the data from the buffer. This will be done with SerialInRecord.

```
SerialInRecord (Com1,RawString,&h02,0,&h0D03,NBytesReturned,01)
```

Note the usage of hexadecimal notation, this is the easiest manner to use two characters as the EndWord. With Nbytes of zero, SerialInRecord will place in RawString the data that occurs between BeginWord and EndWord. We can send the program to the datalogger and verify RawString is now updated with data.

```
Public PTemp, batt_volt
Public RawString As String * 30, SplitStrings(3) As String * 16, FloatValue, IntegerValue
As Long
Public NBytesReturned
```

```
DataTable (Test,1,1000)
  DataInterval (0,15,Sec,10)
  Minimum (1,batt_volt,FP2,0,False)
  Sample (1,PTemp,FP2)
EndTable
```

```
BeginProg
  SerialOpen (Com1,4800,18,0,50)
  SerialOpen (Com2,4800,18,0,50)
  Scan (1,Sec,0,0)
  PanelTemp (PTemp,250)
  Battery (batt_volt)
  SerialInRecord (Com1,RawString,&h02,0,&h0D03,NBytesReturned,01)
```

```
  CallTable Test
NextScan
SlowSequence
Scan (1,Sec,3,0)
  SerialOut (Com2,CHR(2)&"1234E+04,1234,abcd"&CHR(13)&CHR(3),"",0,100)
NextScan
EndProg
```

The next step is to divide the RawString into pieces with SplitStr. Looking at the example string, we see that the data is separated by commas. We want the value before the first comma and all values that follow commas. That corresponds to the FooterFilter option of SplitStr. The maximum number of strings is 3 and the destination is our array SplitStrings().

```
SplitStr (SplitStrings(),RawString,",",3,5)
```

The final step is to process our data. We will use the automatic data type conversion of CRBasic.

```
FloatValue=SplitStrings(1)
```

```
IntegerValue=SplitStrings(2)
```

Merely copying the values from strings into floats or longs will change the data type. These values can now be used in calculations in the same manner as values from non-serial sensors. You should create data tables for the output processing you desire. We now have a complete program to read our serial sensor as follows.

```
Public PTemp, batt_volt
```

```
Public RawString As String * 30, SplitStrings(3) As String * 16, FloatValue, IntegerValue  
As Long
```

```
Public NBytesReturned
```

```
DataTable (Test,1,1000)
```

```
  DataInterval (0,15,Sec,10)
```

```
  Minimum (1,batt_volt,FP2,0,False)
```

```
  Sample (1,PTemp,FP2)
```

```
  Sample (1,FloatValue,IEEE4)
```

```
  Sample (1,IntegerValue,Long)
```

```
  Sample (1,SplitStrings(3),String)
```

```
EndTable
```

```
BeginProg
```

```
  SerialOpen (Com1,4800,18,0,50)
```

```
  SerialOpen (Com2,4800,18,0,50)
```

```
  Scan (1,Sec,0,0)
```

```
    PanelTemp (PTemp,250)
```

```
    Battery (batt_volt)
```

```
    SerialInRecord (Com1,RawString,&h02,0,&h0D03,NBytesReturned,01)
```

```
    SplitStr (SplitStrings(),RawString,",",3,5)
```

```
    FloatValue=SplitStrings(1)
```

```
    IntegerValue=SplitStrings(2)
```

```
  CallTable Test
```

```
  NextScan
```

```
  SlowSequence
```

```
  Scan (100,mSec,3,0)
```

```
    SerialOut (Com2,CHR(2)&"1234E+04,1234,abcd"&CHR(13)&CHR(3),"",0,100)
```

```
  NextScan
```

```
EndProg
```

4.2 Polled Sensor

For our example polled sensor, we will use the same simulated sensor as before except that it now must receive "PollData" before it outputs data. The program only needs two additional instructions for reading the polled sensor, a serial output and a delay before reading from the buffer. For the serial output, we will use SerialOutBlock. It can be placed before the other measurement instructions, so they can execute when you would otherwise delay. The amount of time needed for a polled sensor to respond is often not specified in the manual. It can easily be verified with the 'W' terminal mode of the datalogger.

Below is the program to read our polled sensor. Notice the position of SerialOutBlock and Delay in relation to SerialInRecord.

```
Public PTemp, batt_volt
Public RawString As String * 30, SplitStrings(3) As String * 16, FloatValue, IntegerValue As Long
Public NBytesReturned, SensorInput As String * 16
```

```
DataTable (Test,1,1000)
  DataInterval (0,15,Sec,10)
  Minimum (1,batt_volt,FP2,0,False)
  Sample (1,PTemp,FP2)
  Sample (1,FloatValue,IEEE4)
  Sample (1,IntegerValue,Long)
  Sample (1,SplitStrings(3),String)
EndTable
```

```
BeginProg
  SerialOpen (Com1,4800,18,0,50)
  SerialOpen (Com2,4800,18,0,50)
  Scan (1,Sec,0,0)
  SerialOutBlock (Com1,"PollData",8)
  PanelTemp (PTemp,250)
  Battery (batt_volt)
  Delay (0,100,mSec)
  SerialInRecord (Com1,RawString,&h02,0,&h0D03,NBytesReturned,01)
  SplitStr (SplitStrings(),RawString,"",3,5)
  FloatValue=SplitStrings(1)
  IntegerValue=SplitStrings(2)
```

```
  CallTable Test
  NextScan
  SlowSequence
  Scan (100,mSec,3,0)
  SerialIn (SensorInput,Com2,10,13,16)
  If InStr (1,SensorInput,"PollData",2) Then
    SerialOut (Com2,CHR(2)&"1234E+04,1234,abcd"&CHR(13)&CHR(3),"",0,100)
  EndIf
  NextScan
EndProg
```

4.3 SplitStr Options

The most common split options used with SplitStr are Numeric and FooterFilter. The following simple program can be used to see the behavior of different options. The variables rawString and filterString can be changed in real time by the user.

```
'CR1000 Series Datalogger
Public rawString As String * 50
Public filterString As String * 16
Public option0(5)      'Numeric split will output floating point numbers
Public option1(5) As String * 16
Public option2(5) As String * 16
Public option3(5) As String * 16
Public option4(5) As String * 16
Public option5(5) As String * 16
Public option6(5) As String * 16
Public option7(5) As String * 16

Public option8(5) As String * 16

BeginProg
  rawString = "123e+123a123E123b123e-123"
  Scan (1,Sec,0,0)
    SplitStr (option0(),rawString,filterString,5,0)
    SplitStr (option1(),rawString,filterString,5,1)
    SplitStr (option2(),rawString,filterString,5,2)
    SplitStr (option3(),rawString,filterString,5,3)
    SplitStr (option4(),rawString,filterString,5,4)
    SplitStr (option5(),rawString,filterString,5,5)
    SplitStr (option6(),rawString,filterString,5,6)
    SplitStr (option7(),rawString,filterString,5,7)
    SplitStr (option8(),rawString,filterString,5,8)
  NextScan
EndProg
```

Appendix A. Troubleshooting Tips

Use the latest version of the datalogger operating system. If you are using an older operating system, update it before trying any other troubleshooting.

Record diagnostic codes from the sensor, result codes of instructions, and checksum errors. These will narrow problems when troubleshooting. These values can also be used as data quality indicators for processed data.

Try switching the transmit and receive wires.

Try using the RS232 port, SDM-SIO1, or SC105. The control port pairs, while they work with most devices, are not true RS232 voltage level.

If the device is polled, double check the command you are sending to the device. Sometimes polled commands need to be terminated with a carriage return, even though the manual does not specify.

Your buffer size may be too small. Try increasing the buffer size to 10,000 bytes while troubleshooting your program.

If you are interfacing a GPS, use the dedicated GPS instruction.

A.1 Why do I only receive part of the message from the sensor?

Your string variables may be dimensioned too short. The NbytesReturned from SerialInRecord will be a good indicator.

A.2 My sensor sends binary data, but the datalogger doesn't seem to receive it. Why do I not see data in my string in table monitor?

Binary data often uses unprintable ASCII characters. If no printable characters exist in the string before a Null character is encountered, no data will be displayed. This does not mean no data is in the string. The CRBasic command ASCII() can be used to check the values of characters in strings which cannot be printed. Data containing Null characters can not be treated as a string.

A.3 Data won't even appear in the 'W' terminal mode?

Your string variables may be dimensioned too short. The NbytesReturned from SerialInRecord will be a good indicator.

Appendix B. Terminal Mode W

The CRBasic dataloggers have a special mode available through their terminal commands for troubleshooting serial connections. At the datalogger prompt in the terminal, type 'W' then press 'Enter'. A list of available communication ports is provided (see Figure 5).

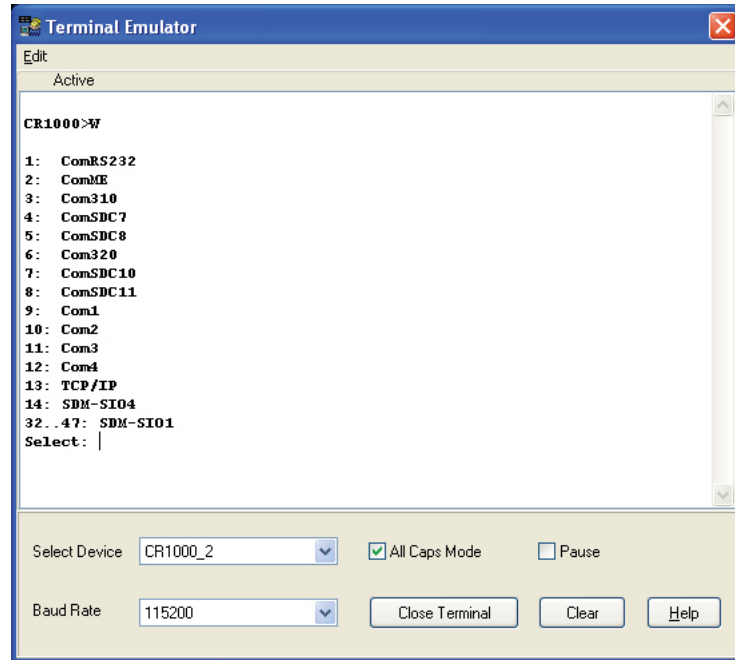


Figure 5. This terminal emulator screen lists the communication ports that are available.

In this case, an electronic compass is attached to COM3. COM3 is selected by typing '11' then 'Enter' (see Figure 6).

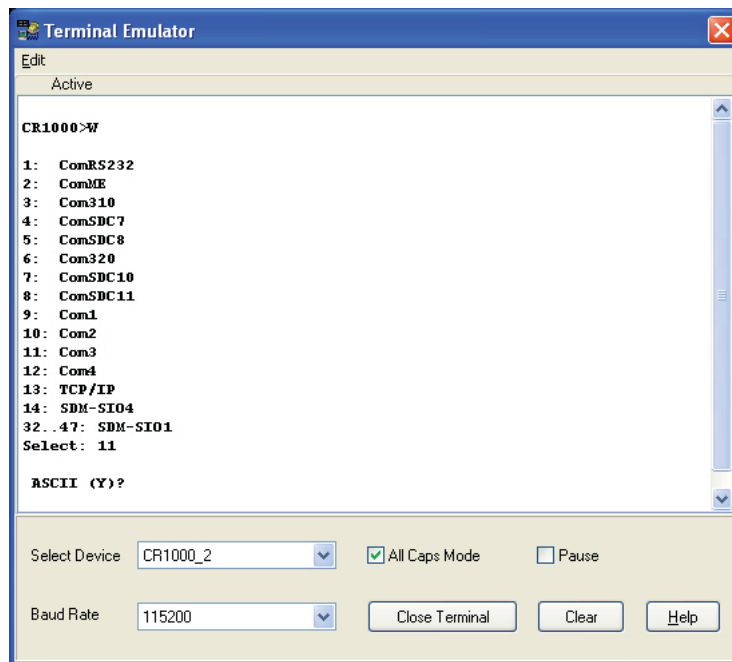


Figure 6. COM3 is chosen by typing 11 and pressing the enter key.

A prompt asks the user if the data is ASCII. Answering 'N' will display the data in hexadecimal format. Answering 'Y' will display the data as ASCII characters (see Figure 7). The datalogger needs to have the communication port configured correctly with SerialOpen for data to appear in Mode W. If using the SDM-SIO1, a serial input instruction such as SerialInRecord also needs to be in the program before data will appear in the terminal.

The electronic compass in this example is configured for continuous output. The data appearing in the terminal confirms the sensor is outputting data twice a second

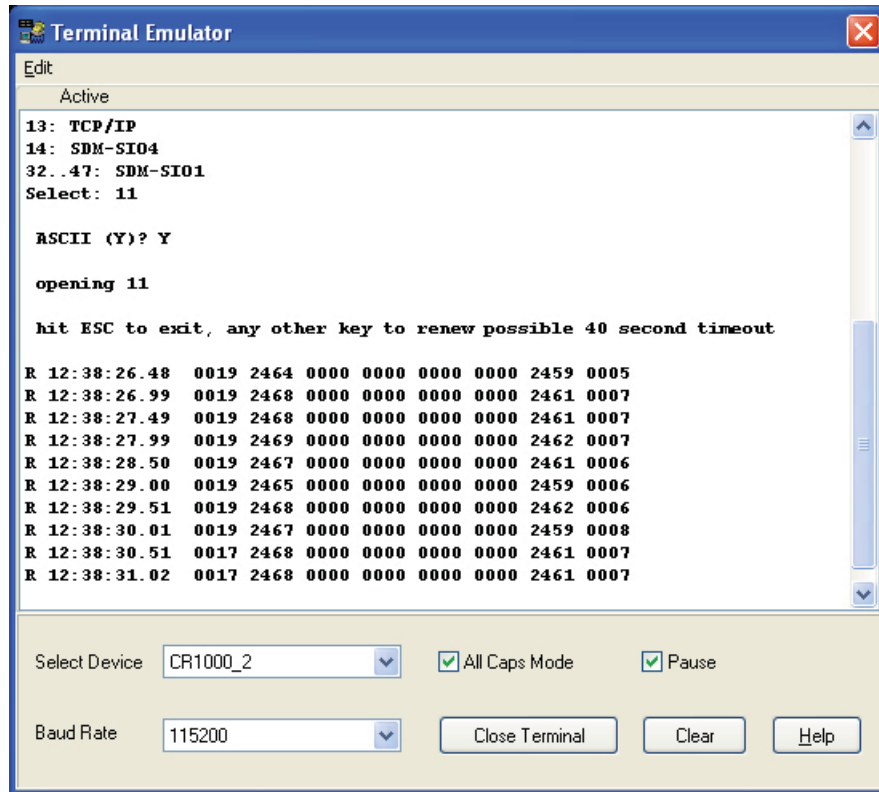


Figure 7. The data from the electronic compass is displayed as ASCII characters.