Using Visual Basic to Communicate with Sentinel I-21/B-21/F-21

APPLICATION BULLETIN #112A **June 6, 2002**

The Sentinel I-21/B-21/F-21 instruments incorporate both an RS232 and RS485 serial port that may be used in a variety of ways. In some situations, a person may want to write custom software on an IBM PC to gather test data, program the instrument, or perform other tasks. This document reviews a segment of Visual Basic code written for data collection and reporting with the Sentinel I-21/B-21/F-21 instruments. This code is responsible for the actual data gathering procedure. Please refer to Application Bulletin #105A for an explanation of the message formats. Sentinel software version 57 or later must be used for RS232 communications. Sentinel software version 73 or later must be used with RS485 communications.

This document provides details about serial communications with a Sentinel I-21/B-21/F-21 leak test instrument. THE INFORMATION PROVIDED BELOW IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. IN NO EVENT SHALL CINCINNATI TEST SYSTEMS OR ITS REPRESENTATIVES BE LIABLE FOR DAMAGES WHATSOEVER INCLUDING DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, LOSS OF PROFITS, OR SPECIAL DAMAGES UNDER ANY CIRCUMSTANCES. THE USER OF THIS INFORMATION IS SOLELY RESPONSIBLE FOR PROPERLY ENGINEERING AND IMPLEMENTING ALL CONNECTIONS TO THE SENTINEL TEST EQUIPMENT.

Discussion

The Sentinel serial ports are passive. They send information only upon request. They do not normally "stream" data[1]. In a typical serial port conversation, the IBM PC computer will transmit a message: "I want the *Total Tests Since New* value from instrument 3" (^A03^BRDAT,8^C). The Sentinel instrument will respond with: "The *Total Tests Since New* value is 1883" (^A03^BRDAT,8,1883^C). Conversations continue in this manner as required. Since the instrument software is multitasking, the user may use the serial ports at any time without affecting the leak test.

Cincinnati Test Systems, Inc.
Member of *TASI* - A Total Automated Solutions Inc. Company
5555 Dry Fork Road, Village of Cleves, OH 45002 • Tel. 513-367-6699 • Fax 513-367-5426
Website: www.cincinnati-test.com • E-mail: sales@cincinnati-test.com

For ”Streaming Data” there is of course an exception. The instrument can be configured to transmit test data automatically at the end of each test. This setup is normally used to send test results to a printer or other **dedicated** receiver (including a computer). This approach can be used, but you can only receive test data, and only when the Sentinel instrument is ready to send it, not necessarily when your computer is ready to receive it. For more details, please request Application Bulletin #104A.

In the first example code, an IBM PC style computer is connected to a group of Sentinel B-21/I-21/F-21 instruments through an RS485 network. The computer has an RS232 to RS485 converter connected to its RS232 port. In this example, each instrument is running independently, so the computer must determine the test type (a dual test produces two sets of results for each instrument cycle) and the number of tests run since data was last collected (the instrument has the most recent 1000 test values in memory, but in most situations only the last few are “new” since our last retrieval). The sample Visual Basic code performs the following functions:

1. Configure the IBM PC RS232 port to talk to a Sentinel I-21/B-21/F-21 Instrument.
2. Begin a loop that
    - determines the pneumatic circuit using the subroutine Get_Pneum( )
    - determines the number of tests that were run since the last retrieval
    - limits the number of new data values to be collected from 1 and 100
    - resets the instrument data pointer using the function ResetPointer( )
    - retrieves the data using the subroutine Get_Result( )
    - parses the data string into file friendly segments using the subroutine Peel( )

There are other error trapping details in the code, but in most cases they deal with potential noise or problems with the RS485 network, not the Sentinel instrument.

In the second example, the entire code associated with the Get_Counter subroutine is shown. Note that more than half of the code is dedicated to transmission error detection and correction.

AB112A.doc
06Ju02

* PART OF FUNCTION "SCAN"
that coordinates data retrieval

```
'*************************
'Set up the communications port
'*************************

Comm1.CommPort = SerialPort
Comm1.RTSEnable = True      'Allows communication
Comm1.Settings = "9600,N,8,1"
Comm1.InputLen = 0          'Allow any length input string
Comm1.InBufferCount = 0     'Clear input buffer
Comm1.PortOpen = True       'Open the port

'*************************
'Start the retrieval process...
'1) Get the manifold (0,1,2,3) for the instrument
'2) Reset the I-21 data pointer
'3) Get the data
'4) Parse the data into segments
'5) Store all this into an array
'*************************

For Instrument = 1 To Quan_of_Instrs
    Manif = Get_Pneum(Instrument)
    If Manif = "Err" Then
        Bad_Units = Bad_Units + 1
        ComErrors(Instrument) = ComErrors(Instrument) + 1
        DisabledInstrument(Instrument) = 1
        If Bad_Units = Quan_of_Instrs Then GoTo Comm_Error
        GoTo Skip
    End If
    If Manif = "0" Or Manif = "1" Then Size(Instrument) = 5
    If Manif = "2" Or Manif = "3" Then Size(Instrument) = 9


    '*************************************************************
    'Finally, we start getting the data.  This data will be stored
    'in the "TestData" matrix for eventual use.  Note: the result data comes in
    'one long string.  This string must be fragmented into its
    'components (Part #, Press Loss, Zero Shift, Leak Rate, and Result)
    'using "Peel" before it can be placed in TestData()
    'Note:
    '     Test_Results_Reqd() tells us how many leak tests were
    '          run in instrument "I" since the last scan
    '     Get_Counter() is a function that gets the "runs since new" register
    '     OldCounter() stores the "runs since new" counter from the
    '          previous scan
    '     Value contains an entire data packet (Part Num, Pres Loss, Zero Shft,...,A/R2)
    '          and must be broken down to its components using Peel()
    '     ResultPacketNum increments from 1 to the Test_Results_Reqd for that instrument
    '     ResultComponentNum = 1 (Part Num), 2 (Pres Loss), 3 (ZS), etc.
    '     TestData(Instrument1  ,  Result Packet #1 ,  Part Num   )
    '                             (                    Pres Loss  )
    '                             (                    Zero Shft  )
    '                             (                    Leak Rate  )
    '                             (                    A/R        )
    '                             (       ,  Result Packet #2 ,  Part Num   )
    '                             (                    Pres Loss  )
    '                             (                    Zero Shft  )
    '                             (                    Leak Rate  )
    '                             (                    A/R        )
    '                   (Instrument2  ,  Result Packet #1 ,  Part Num   )
    '                             (                    Pres Loss  )
    '                             (                    Zero Shft  )
    '                             (                    Leak Rate  )
    '                             (                    A/R        )
    'Note:  NewData_Avail tells all interested timers that new test data is available.
    '*************************************************************
```

Note: if the instrument fails to respond (turned off, etc), then Get_Pneum returns "Err"

```
    TempData = Get_Counter(Instrument, 8)
    If TempData = "Err" Then
        ComErrors(Instrument) = ComErrors(Instrument) + 1
        GoTo Skip
    End If
    Test_Results_Reqd(Instrument) = TempData - RunsSinceLastScanned(Instrument)

    If Test_Results_Reqd(Instrument) < 0 Then
        RunsSinceLastScanned(Instrument) = TempData
        Test_Results_Reqd(Instrument) = 0
        GoTo Skip
    End If
    If Test_Results_Reqd(Instrument) > 100 Then Test_Results_Reqd(Instrument) = 100
    If Test_Results_Reqd(Instrument) = 0 Then
        NoCyclesOnInst(Instrument) = 1
        GoTo Skip
    End If
    NewData_Avail = "Yes"
    ResetPointer (Instrument)
    For ResultPacketNum = 1 To Test_Results_Reqd(Instrument)
        Value = Get_Result(Instrument, Manif)
        If Value = "Err" Then
            ComErrors(Instrument) = ComErrors(Instrument) + 1
            Test_Results_Reqd(Instrument) = ResultPacketNum - 1
            RunsSinceLastScanned(Instrument) = RunsSinceLastScanned(Instrument) + _
             Test_Results_Reqd(Instrument)
            GoTo Skip
        End If
        If Value = "Done" Then
            RunsSinceLastScanned(Instrument) = TempData
            Test_Results_Reqd(Instrument) = ResultPacketNum - 1
            GoTo Skip
        End If
        If Size(Instrument) = 9 And TestForIncompleteData(Value) = "bad" Then
            ResultPacketNum = ResultPacketNum - 1
            GoTo Hop
        End If
        For ResultComponentNum = 1 To Size(Instrument)
            TestData(Instrument, ResultPacketNum, ResultComponentNum) = Peel(Value, _
             ResultComponentNum)
        Next ResultComponentNum
        Hop:
    Next ResultPacketNum
    RunsSinceLastScanned(Instrument) = TempData
    Skip:
Next Instrument

'*********************************
'Now store the newly acquired data in the TestData
'matrix into files on the hard drive
'  a) Open the Database
'  b) Check for valid data
'  c) Store valid data into DB
'*********************************
```

a value we have stored at the end of the last retrieval process

the actual data is collected by Get_Result()

```
Function Get_Counter (address which,
'*****************************************
'This function receives the instrument number and
'returns the "Total Cycles Since New" value for
'the instrument (or "Err" if poor or no communic).
'*****************************************

Dim Count, Result1$, Result2$, Msg, Duration, I, M, Timeouts

Count = 0
Timeouts = 0

If address = "1" Then address = "01"
If address = "2" Then address = "02"
If address = "3" Then address = "03"
If address = "4" Then address = "04"
If address = "5" Then address = "05"
If address = "6" Then address = "06"
If address = "7" Then address = "07"
If address = "8" Then address = "08"
If address = "9" Then address = "09"

'*****************************************
'Setup and send the "request value" message.
'Then read the result.  Note: I-21 requires a
'two character address.
'*****************************************

Repeat:
Msg = Chr$(1) & address & Chr$(2) & "RDAT," & which & Chr$(3)
Result1$ = ""
Comm1.InBufferCount = 0
Comm1.Output = Msg
Duration = Timer + .3
Do Until InStr(Result1$, Chr$(3)) > 0 Or Timer > Duration
    Result1$ = Result1$ + Comm1.Input
Loop
If Timer > Duration Then
    Duration = Timer + .3
    Do Until Timer > Duration
    Loop
    Comm1.InBufferCount = 0
    Timeouts = Timeouts + 1
    If Timeouts > 5 Then
        Get_Counter = "Err"
        Exit Function
    End If
    GoTo Repeat
End If
Do Until Right$(Result1$, 1) = Chr$(3)
    I = Len(Result1$)
    Result1$ = Left$(Result1$, I - 1)
Loop
If InStr(Result1$, "RDAT") = 0 Then GoTo ErrorTrap
Do Until Left$(Result1$, 4) = "RDAT"
    I = Len(Result1$)
    Result1$ = Right$(Result1$, I - 1)
Loop

'*****************************************
'Repeat the read process (for verify purposes)
'Note: Delay is to allow full transmission before input
'*****************************************
```

e.g. which counter (#1 thru #8) value we want

← wait until we get a ^C or time out

'If we get a reasonable message, strip any "noise" characters from the front & back of the message string

```
ReptRes2:
Result2$ = ""
Comm1.InBufferCount = 0
Comm1.Output = Msg
Duration = Timer + .3
Do Until InStr(Result2$, Chr$(3)) > 0 Or Timer > Duration
    Result2$ = Result2$ + Comm1.Input
Loop
If Timer > Duration Then
    Duration = Timer + .3
    Do Until Timer > Duration
    Loop
    Comm1.InBufferCount = 0
    Timeouts = Timeouts + 1
    If Timeouts > 5 Then
        Get_Counter = "Err"
        Exit Function
    End If
    GoTo ReptRes2
End If
Do Until Right$(Result2$, 1) = Chr$(3)
    I = Len(Result2$)
    Result2$ = Left$(Result2$, I - 1)
Loop
If InStr(Result2$, "RDAT") = 0 Then GoTo ErrorTrap
Do Until Left$(Result2$, 4) = "RDAT"
    I = Len(Result2$)
    Result2$ = Right$(Result2$, I - 1)
Loop

'*************************************************
'Compare to see if received data is correct.  If so,
'strip the communication tailer(s) and set
'Get_Pneum to result. If read error, set Get_Pneum to Err.
'Note: Function attempts to get the data 3 times before fault.
'*************************************************

If Result1$ = Result2$ Then
    Do Until IsNumeric(Right$(Result1$, 1))
        I = Len(Result1$)
        Result1$ = Left$(Result1$, I - 1)
    Loop
    Do Until IsNumeric(Left$(Result1$, 1))
        I = Len(Result1$)
        Result1$ = Right$(Result1$, I - 1)
    Loop
    Do Until InStr(Result1$, ",") = 0
        I = Len(Result1$)
        Result1$ = Right$(Result1$, I - 1)
    Loop
    Get_Counter = Result1$
    Exit Function
End If

'*************************************************
'ErrorTrap catches strings that dont have "RDAT" in
'them after the tailer has been stripped.
'*************************************************
```

```
ErrorTrap:
If Count <= 10 Then
    Count = Count + 1
    GoTo Repeat
Else
    Get_Counter = "Err"
End If

End Function
```