dataTaker

dataTaker General Code Examples

Updated 13/7/2020

Thermo Fisher SCIENTIFIC

Table of Contents

Introduction

Example Programs are working examples produced with dataTaker loggers that can be downloaded and utilized as a starting point for your own projects or applications.

Hardware: DT80, 800 and 500 Series of logger

Starting and stopping the DataTaker at a set time

The DataTaker has several internal timers and in combination with the alarm functions can be used to start and stop the DataTaker.

This program starts schedule B running at 9 AM and stops schedule B at 10 AM.

DT80 range V9 firmware and latter

Version 9 firmware for the DT8x range introduced CRON based schedule triggering. This new schedule trigger greatly simplifies this and similar tasks.

```
BEGIN

RB[*:*:9-10]
   1V
   LOGON
END
```

DT80 range Pre V9 firmware and DT800

Note: 1ST is a seconds timer. 2ST is a minutes timer. 3ST is an hours timer. 4ST is day of week.

```
BEGIN

RA1H
   ALARM1(3ST>9)”Start”{[GB]}
   ALARM2(3ST>10)”Stop”{[HB]}

RB1S
   1V
   LOGON
END
```

DT50/500/600 range code

Note: 1ST is a seconds timer. 2ST is a minutes timer. 3ST is an hours timer. 4ST is day of week.

```
BEGIN

RA1S
   1V

RZ1H
   ALARM1(3ST>9)”Start”{[GA]}
   ALARM2(3ST>10)”Stop”{[HA]}
   LOGON
END
```

Zeroing readings

There are a number of readings that require offsets to be zeroed out at the start of data acquisition.

This can be achieved in several ways.

1/ Manually read the sensor and then in the DataTaker program subtract the zero reading from the actual reading. This method is recommended for long term testing where there is a chance the DataTaker might be reset. If the offset is hard coded in the program, then it will not be lost is the dataTaker is reset due to power loss.

For example;

```
BEGIN

1CV(W)=123  ‘Read initial value

RA1S
   1BGI(=2CV,W) ‘Take reading
   3CV=2CV-1CV ‘Remove Zero reading
END
```

2/ Read the sensor in the immediate schedule and assign the value to a CV. Then subtract the CV from the actual reading in the DataTaker program.

For example;

```
BEGIN

1BGI(=1CV,W)  ‘Read initial value

RA1S
   1BGI(=2CV,W) ‘Take reading
   3CV=2CV-1CV ‘Remove Zero reading
END
```

3/ Read the sensor in the X schedule, and assign the value to a CV. Then subtract the CV from the actual reading in the DataTaker program. The readings can be zeroed at any time by sending an X character to the DataTaker

For example;

```
BEGIN

RA1S
   1BGI(=2CV,W) ‘Take reading
   3CV=2CV-1CV ‘Remove Zero reading

RX
   1BGI(=1CV,W)  ‘Read initial value
END
```

4/ Read the sensor in a function and assign the value to a CV. Then subtract the CV from the actual reading in the DataTaker program. The readings can be zeroed at any time by selecting the ‘Tare’ function (DT80 series loggers only)

For example;

```
BEGIN
   FUNCTION1=”Tare”{1BGI(=1CV,W)}      ‘Read initial value
RA1S
   1BGI(=2CV,W) ‘Take reading
   3CV=2CV-1CV ‘Remove Zero reading
END
```

Accumulating a measurement such as flow

Application description: A 4–20 mA flow meter connected to channel 1. The scaling of the flow meter is 0–30 litres per hour. We want to measure the flow every minute and report the total volume every hour and every day. If we read the meter each minute, then 30 litres per hour translates to a scaling of 0.5 litres per minute.

Code

```
BEGIN
 S1=0,0.5,0,100”Litres”
 RA1M
  1#L(S1,”Litres per min”,+=1CV,W)
 RB1H
  1CV(“Litres per hour”,+=2CV,R)
 RC1D
  2CV(“Litres per day”,R)
END
```

Extracting time components from time channel

This code extracts milli seconds, seconds, minutes, or hours from the time. For DT5xx series, the milli second part can be removed and schedule rate changed to seconds etc

```
BEGIN
RA200T
        T(=1CV)                 ‘Read the time
        1CV                     ‘Display raw time data (in seconds)
        2CV=(1CV*1000)%1000     ‘Display Milli Second time part (DT8x/800 only)
        3CV=1CV%60              ‘Display Seconds time part
        4CV=(1CV/60)%60         ‘Display Minute time part
        5CV=(1CV/3600)%24‘Display Hour time part
END
```

Statistics

In addition to using the built in STATISTICAL functions and schedule (RS), it is possible to return statistical data by direct calculation as shown in these examples

Co-Variance

Covariance is a method of measuring how strongly variable are related to each other. For further details, please refer to TN-0026.

```
BEGIN
'
'Program for calculate the covariance of two variables.
'
RA1S 'Sample variables every 1 second
   1V(=1CV,W) 'Read the A and assign to 1CV
   2V(=2CV,W) 'Read the B and assign to 2CV
    3CV(W)=3CV+1CV '3CV = Sum of A
   4CV(W)=4CV+2CV '4CV = Sum of B
   5CV(W)=5CV+(1CV*2CV) '5CV = Sum A*B
   6CV(W)=6CV+1 'Number of samples
RB1M 'Report Covariance every 1 minute
   7CV=5CV/6CV-(3CV/6CV)*(4CV/6CV) 'Calculate Covar(A,B) and save
   1..7CV(W)=0 'Reset the variables and start again
LOGONB 'Turn on logging for schedule B
END
```

Linear regression

Linear regression is a method to calculate the variables for a straight line fit through a data set. In this case, the DataTaker will measure two inputs and then return the two co-efficients

```
BEGIN"LinR"
RA1S
  1V("X",=1CV)
  2V("Y",=2CV)
  3CV("n")=3CV+1 'Used in M and B
  4CV("Sum X")=4CV+1CV 'Used in M and B
  5CV("Sum Y")=5CV+2CV 'Used in M and B
  6CV("Sum X*Y")=6CV+(1CV*2CV) 'Used in M
  7CV("Sum(X^2)")=7CV+1CV^2 'Used in M

RB1M
 8CV("M")=(6CV-(4CV*5CV))/((3CV*4CV)-4CV^2)
 9CV("B")=(5CV-(8CV*4CV))/3CV

END
```

Running Average

```
Begin"Run_AV"
RA1S
   'Keep last 10 readings in a shift register.
   'These CV's are used to calculate;
   '    Running average
   10cv(w)=9cv
   9cv(w)=8cv
   8cv(w)=7cv
   7cv(w)=6cv
   6cv(w)=5cv
   5cv(w)=4cv
   4cv(w)=3cv
   3cv(w)=2cv
   2cv(w)=1cv
   'Read current temperature.
   1tk(=1cv,"Current reading ~DegC")
   'Calculate running average.
   11cv(w)=(1cv+2cv+3cv+4cv+5cv+6cv+7cv+8cv+9cv+10cv)/10
   11CV("Run_AV")
END
```

Running minimum

```
BEGIN"Run_MN"
RA1S
   'Keep last 10 readings in a shift register.
   'These CV's are used to calculate;
   '    Running minimum.
   10cv(w)=9cv
   9cv(w)=8cv
   8cv(w)=7cv
   7cv(w)=6cv
   6cv(w)=5cv
   5cv(w)=4cv
   4cv(w)=3cv
   3cv(w)=2cv
   2cv(w)=1cv
   'Read current temperature.
   1tk(=1cv,"Current reading ~DegC")
   12cv(w)=1cv   'Seed the running minimum with current temperature.
   12cv(w)=12cv*(2cv>=12cv)+2cv*(2cv<12cv) 'If n-1 is less than current running minimum then update.
   12cv(w)=12cv*(3cv>=12cv)+3cv*(3cv<12cv) 'If n-2 is less than current running minimum then update.
   12cv(w)=12cv*(4cv>=12cv)+4cv*(4cv<12cv)
   12cv(w)=12cv*(5cv>=12cv)+5cv*(5cv<12cv)
   12cv(w)=12cv*(6cv>=12cv)+6cv*(6cv<12cv)
   12cv(w)=12cv*(7cv>=12cv)+7cv*(7cv<12cv)
   12cv(w)=12cv*(8cv>=12cv)+8cv*(8cv<12cv)
   12cv(w)=12cv*(9cv>=12cv)+9cv*(9cv<12cv)
   12cv(w)=12cv*(10cv>=12cv)+10cv*(10cv<12cv)
   12CV("Minima")
END
```

Running Maxima

```
BEGIN"Run_MX"
RA1S
   'Keep last 10 readings in a shift register.
   'These CV's are used to calculate;
   '    Running maximum.
   10cv(w)=9cv
   9cv(w)=8cv
   8cv(w)=7cv
   7cv(w)=6cv
   6cv(w)=5cv
   5cv(w)=4cv
   4cv(w)=3cv
   3cv(w)=2cv
   2cv(w)=1cv
   'Read current temperature.
   1tk(=1cv,"Current reading ~DegC")
   'Calculate the running maximum.
   13cv(w)=1cv   'Seed the running maximum with current temperature.
   13cv(w)=13cv*(2cv<=13cv)+2cv*(2cv>13cv) 'If n-1 is less than current running maximum then update.
   13cv(w)=13cv*(3cv<=13cv)+3cv*(3cv>13cv) 'If n-2 is less than current running maximum then update.
   13cv(w)=13cv*(4cv<=13cv)+4cv*(4cv>13cv)
   13cv(w)=13cv*(5cv<=13cv)+5cv*(5cv>13cv)
   13cv(w)=13cv*(6cv<=13cv)+6cv*(6cv>13cv)
   13cv(w)=13cv*(7cv<=13cv)+7cv*(7cv>13cv)
   13cv(w)=13cv*(8cv<=13cv)+8cv*(8cv>13cv)
   13cv(w)=13cv*(9cv<=13cv)+9cv*(9cv>13cv)
   13cv(w)=13cv*(10cv<=13cv)+10cv*(10cv>13cv)
   13CV("Run_MX")
END
```

Line of Best Fit

For a more complete explanation, please see the Tech Note: TN-0027. This uses the same mathematical technique as Excel to establish a line of best fit using the logger. This technique can be used to establish an ‘End Point’ for a dataset, for example predicting when a tank will be emptied, etc.

```
BEGIN
 Y1=0,1”Sec”
 T(=6CV,W) ‘Read the start time and store in 6CV.
 4CV(“Y0”,W)=0 ‘Tank empty value.
 RA1S
   T(=1CV,W) ‘Read Time now.
   1CV(“X”)=1CV-6CV ‘Calculate elapsed time.
   1R(“Y”,4W,=2CV) ‘Read tank level.
   10CV(“N”,W)=10CV+1 ‘Number of reading taken.
   11CV(“SumX”,W)=11CV+1CV ‘Total of X values.
   12CV(“SumY”,W)=12CV+2CV ‘Total of Y values.
   13CV(“SUMX*Y”,W)=13CV+(1CV*2CV) ‘Total of X*Y values.
   14CV(“SUMX^2”,W)=14CV+(1CV*1CV) ‘Total of X squared.
   ‘Calculate the Slope.
   15CV(“m”)=((10CV*13CV)-(11CV*12CV))/((10CV*14CV)-(11CV^2))
   ‘Calculate the Intercept.
   16CV(“c”)=((12CV*14CV)-(11CV*13CV))/((10CV*14CV)-(11CV^2))
   ‘Calculate the estimated time if more than 6 samples taken.
   3CV(“X0”,Y1)=((4CV-16CV)/15CV)*(10CV>6)
   ‘Calculate remaining time if more than 6 samples taken.
   5CV(“X0-X”,Y1)=(3CV-1CV)*(10CV>6)
 RX
   10..16CV(W)=0 T(=6CV,W) ‘Reset variables and time.
X G LOGON
END
```

Exponential Smoothing

Exponential smoothing is a simple method of taking the average of a stream of data. It is simply the process of adding a ‘part’ of the difference between the current value and the new value to give a ‘new’ average. The ‘part’ of the difference added is called the smoothing factor. When the smoothing factor is one, there is NO smoothing applied, OR if the smoothing factor is zero, the reading does not change. For example, with a smoothing factor of 0.1, the current reading is 10. We receive a new reading of 12. We then apply 0.1 of the difference between the old reading and the current reading. This gives us a new reading of 10.2. Exponential smoothing effectively applies a damping factor.

```
BEGIN”Smooth”
 1CV(“Factor”,W)=0.2
 RA1S
   1TK(=2CV,”Raw”)
   3CV(“Smooth”)=3CV+((2CV-3CV)*1CV)
END
```

This is a VERY simple averaging technique. One needs to experiment with different smoothing factors to obtain the appropriate result. As the smoothing factor is within a CV, the actual factor used can be manipulated while the logger program is running. In this case, the initial factor is set to 0.2. If the command 1CV=1 is sent to the logger either from an external Host connection, or as a result of another part of the logger program, the new factor (in this case ‘1’) will be applied dynamically. Please note that like most averaging and smoothing techniques, the ‘phase’ of the data is shifted (or slightly delayed). The results obtained using this approach are VERY SIMILAR to a running average. The advantage of this technique is that it uses considerably less logger resources.

F_0 food sterilization calculation

The F_0 calculation is used to validate the thermal processing of food products to ensure it has been sterilized correctly.

```
BEGIN
1..16CV(W)=0
   RA15S HA
       ‘Measure food temperature
     1+TT(“Temp 1”,=1CV)
     1-TT(“Temp 2”,=2CV)
     2+TT(“Temp 3”,=3CV)
     2-TT(“Temp 4”,=4CV)
     3+TT(“Temp 5”,=5CV)
     3-TT(“Temp 6”,=6CV)
       ‘Calculate F0 for last 15 seconds
     1CV(W)=(10^((1CV-121.1)/10))*0.25
     2CV(W)=(10^((2CV-121.1)/10))*0.25
     3CV(W)=(10^((3CV-121.1)/10))*0.25
     4CV(W)=(10^((4CV-121.1)/10))*0.25
     5CV(W)=(10^((5CV-121.1)/10))*0.25
     6CV(W)=(10^((6CV-121.1)/10))*0.25
        ‘Sum F0 for each channel
     11CV(“Probe1”)=11CV+1CV
     12CV(“Probe2”)=12CV+2CV
     13CV(“Probe3”)=13CV+3CV
     14CV(“Probe4”)=14CV+4CV
     15CV(“Probe5”)=15CV+5CV
     16CV(“Probe6”)=16CV+6CV
END
```

Phase change detection

Applies to DT8x and DT800
This program is to record the phase change temperature of materials while being heated or cooled.

It works by;
Keeping track of the last 10 readings by using a shift register.

- Calculates the average temperature of the last 10 readings
- Calculates the minimum of the last 10 readings
- Calculates the maximum of the last 10 readings

As a material changes phase the temperature remains stable due to the latent heat capacity of the material under test.

If the Maximum and Minimum of the last 10 readings is within a specified tolerance then the average temperature of the last 10 readings is saved.

The is also a feature that sets the range to check for the phase changes. This removes data being logged as a phase change at the start or end of a test run.

```
begin
‘Initialize channel variables.
1cv=0           ‘Holds the current temperature value.
2..10cv=0       ‘Use to store the last 10 readings as a shift register.
11cv=0  ‘Holds moving average of last 10 readings.
12cv=0  ‘Holds the minimum of the last 10 values.
13cv=0  ‘Holds the maximum of the last 10 values.
14cv=1  ‘Hold tolerance of phase change. (+/- 14cv of temperature)
15cv=0  ‘Holds minimum tolerance of phase change.
16cv=0  ‘Holds maximum tolerance of phase change.
17cv=-20 ‘Hold minimum temperature range for checking phase change.
18cv=120 ‘Hold maximum temperature range for checking pahse change.
‘Schedule speed should be adjusted so 10
‘readings are taken over the phase change.
RA1S
    ‘Keep last 10 readings in a shift register.
    ‘These CV’s are used to calculate;
    ‘   Running average
    ‘   Running minimum.
    ‘   Running maximum.
    10cv(w)=9cv
    9cv(w)=8cv
    8cv(w)=7cv
    7cv(w)=6cv
    6cv(w)=5cv
    5cv(w)=4cv
    4cv(w)=3cv
    3cv(w)=2cv
    2cv(w)=1cv
    ‘Read current temperature.
    1tk(=1cv,”Current reading ~DegC”)
    ‘Calculate running average.
    11cv(w)=(1cv+2cv+3cv+4cv+5cv+6cv+7cv+8cv+9cv+10cv)/10
    ‘Calculate the running minimum.
    12cv(w)=1cv  ‘Seed the running minimum with current temperature.
    12cv(w)=12cv*(2cv>=12cv)+2cv*(2cv<12cv) ‘If n-1 is less than current running minimum then update.
    12cv(w)=12cv*(3cv>=12cv)+3cv*(3cv<12cv) ‘If n-2 is less than current running minimum then update.
    12cv(w)=12cv*(4cv>=12cv)+4cv*(4cv<12cv)
    12cv(w)=12cv*(5cv>=12cv)+5cv*(5cv<12cv)
    12cv(w)=12cv*(6cv>=12cv)+6cv*(6cv<12cv)
    12cv(w)=12cv*(7cv>=12cv)+7cv*(7cv<12cv)
    12cv(w)=12cv*(8cv>=12cv)+8cv*(8cv<12cv)
    12cv(w)=12cv*(9cv>=12cv)+9cv*(9cv<12cv)
    12cv(w)=12cv*(10cv>=12cv)+10cv*(10cv<12cv)
    ‘Calculate the running maximum.
    13cv(w)=1cv  ‘Seed the running maximum with current temperature.
    13cv(w)=13cv*(2cv<=13cv)+2cv*(2cv>13cv) ‘If n-1 is less than current running maximum then update.
    13cv(w)=13cv*(3cv<=13cv)+3cv*(3cv>13cv) ‘If n-2 is less than current running maximum then update.
    13cv(w)=13cv*(4cv<=13cv)+4cv*(4cv>13cv)
    13cv(w)=13cv*(5cv<=13cv)+5cv*(5cv>13cv)
    13cv(w)=13cv*(6cv<=13cv)+6cv*(6cv>13cv)
    13cv(w)=13cv*(7cv<=13cv)+7cv*(7cv>13cv)
    13cv(w)=13cv*(8cv<=13cv)+8cv*(8cv>13cv)
    13cv(w)=13cv*(9cv<=13cv)+9cv*(9cv>13cv)
    13cv(w)=13cv*(10cv<=13cv)+10cv*(10cv>13cv)
    ‘Show current values for running average, running minimum and running average.
    11cv(“Average ~DegC”)
    12cv(“Minimum ~DegC”)
    13cv(“Maximum ~DegC”)
    15cv(w)=11cv-14cv ‘Calculate the minimum temperature tolerance.
    16cv(w)=11cv+14cv ‘Calculate the maximum temperature tolerance.
    ‘If the average temperature is inside the test temperature range
    ‘And If the running minimum is within the temperature tolerance
    ‘And if the running maximum is within the temperature tolerance
    ‘Then we have a phase change so save the average temperature.
    if(11cv><17cv,18cv)and
    if(12cv><15cv,16cv)and
    if(13cv><15cv,16cv){[xb]}
    logona
rbx
    20cv(“Phase temperature ~DegC”)=11cv
    logonb
end
```

Simple program to display data received at the serial sensor port

Often when debugging serial sensor code, we need to see data being sent to the serial sensor port. The following is a simple piece of code to display all ASCII characters received on the serial sensor port. This code assumes the ‘line separator’ is a or 13 character, as this is generally true of ASCII data. If non ASCII characters are being returned, then setting P56=1 will display those characters

```
BEGIN”SSdisp”
PS=9600,N,8,1 ‘ set the appropriate communications rate
/n/c/u
RA1SERIAL””
 1SERIAL(“%s[1$]\\013”,W) 1$
END
```

or if the terminating character is CR LF

```
BEGIN”SSdisp”
PS=9600,N,8,1 ‘ set the appropriate communications rate
/n/c/u
RA1SERIAL””
 1SERIAL(“%s[1$]\\013\\010”,W) 1$
END
```

Another method is to;

- Connect to the DataTaker with DeTransfer
- Click in the receive window to bring it into focus
- Click on Receive > Show NonPrint Characters and select what is to be shown. (Decimal is good as you can refer the ASCII table in the DT80 range user manual)
- Select the DeTransfer “Send” window
- Send P56=1 to the DataTaker
- Send the command 1SERIAL(“\\e) which will dump the contents of the serial buffer then clear it

Using analog inputs to record a state change

The analog inputs on a DataTaker can also be used to measure digital states.

```
BEGIN
RA1S
  21CV(W)=20CV      ‘Shift register to save previous sum of channels (Used later in schedule)
     ‘Read digital states
  1AS(2500,=1CV,W)      ‘Save dig 1 to 1 CV as a working channel
  2AS(2500,=2CV,W)      ‘Save dig 2 to 2 CV as a working channel
  3AS(2500,=3CV,W)      ‘etc,
  4AS(2500,=4CV,W)      ‘etc.
  5AS(2500,=5CV,W)
  6AS(2500,=6CV,W)
  7AS(2500,=7CV,W)

     ‘Multiply CV by a binary number
     ‘If we simply add all the digital input up if we get two changing to opposite states
     ‘then there would be no change. By using a binary number the state change is always unique.
  11CV(W)=1CV*2^0      ‘11CV = 0 or 1
  12CV(W)=2CV*2^1      ‘12CV = 0 or 2
  13CV(W)=3CV*2^2      ‘13CV = 0 or 4
  14CV(W)=4CV*2^3      ‘14CV = 0 or 8
  15CV(W)=5CV*2^4      ‘15CV = 0 or 16
  16CV(W)=6CV*2^5      ‘16CV = 0 or 32
  17CV(W)=7CV*2^6      ‘17CV = 0 or 64

     ‘Sum the state changes
  20CV(W)=11CV+12CV+13CV+14CV+15CV+16CV+17CV
     ‘Subtract the previous sum from the current sum.
     ‘If there was no state change the result is 0
     ‘If there was a state change the 202 is not equal to zero
  22CV(W)=21CV-20CV

RB22+CV      ‘Run schedule B when 202CV changes from zero
  D      ‘Record Date schedule is run
  T      ‘Record Time schedule is run
  1CV(FF0,”Dig1”)      ‘Record digital state 1
  2CV(FF0,”Dig2”)      ‘Record digital state 2
  3CV(FF0,”Dig3”)      ‘etc,
  4CV(FF0,”Dig4”)      ‘etc.
  5CV(FF0,”Dig5”)
  6CV(FF0,”Dig6”)
  7CV(FF0,”Dig7”)
END      ‘End program
```

MAP 450 HMI ASCII Operator Interface

Applies to DT8x and DT800

This code is for use with a Maple Systems MAP450B Mini-Terminal operator interface (alpha-numeric keyboard, LCD display). Code assumes the MAP 450 is configured as follows.

- RS232 Baud= 9600, Parity = NONE, Data = 8, Stop = 1, Handshake = NONE
- Also assumes Operations mode set to BLOCK (Send all on CR)

```
•       BEGIN
•       1SSPWR=1     ‘Turn on MAP 450. Connect the power between the 12V and GD.
•            ‘You may need to put a delay in here to allow for initialization time of MAP.
•       PS=9600,N,8,1,NOFC             ‘Set RS232 parameters.
•       1SERIAL(RS232,”{\\027o\\002}\\e”,W)     ‘Clear MAP (Esc o STX) and SSP I/O buffer
•       1SERIAL(RS232,”{\\027k21020\\002}\\e”,W)     ‘Set up key board. Enabled, Upper case, click on.
•       RA1S
•            ‘Read in a string
•           1SERIAL(RS232,”{\\027w14\\002}”,W)  ‘Create display variable.
•           1SERIAL(RS232,”{\\027vInput Text  :\\002}”,W)        ‘Write display variable.
•           1SERIAL(RS232,”%s[1$]”,30,=1CV,W)     ‘Read string into variable 1$. Note: 30 second time out
•            ‘Read in a number
•           1SERIAL(RS232,”{\\027w11\\002}”,W) ‘Create display variable.
•           1SERIAL(RS232,”{\\027vInput Number:\\002}”,W)     ‘Write display variable.
•           1SERIAL(RS232,”%f[10CV]”,30,=2CV)  ‘Read floating point number into variable 1$.
•            ‘Note: 30 second time out
•            ‘Show time out errors
•           ALARM1(1CV><19.5,20.5)”Text input time out^M^J”
•           ALARM2(2CV><19.5,20.5)”Number input time out^M^J”
•           1$(“String is “)
•           10CV(“Number is “)
•       END
```

Count Down

This program counts down from a time to 0

- 1CV hold the start minutes
- 2CV holds the start seconds

```
•       BEGIN
•       /u/n
•       1CV=5
•       2CV=0
•       $=”:”
•       RA1S
•         1CV(W)=((2CV=0)*(1CV-1))+((2CV>0)*1CV)
•         2CV(W)=(1CV>=0AND2CV<=60AND2CV>0)*(2CV-1)+(1CV<0AND2CV<59AND2CV>=0)*(2CV+1)+(1CV>=0AND2CV=0)*(59)
•         1CV(“Minutes”,FF0)
•         $
•         2CV(“Seconds”,FF0)
•       END
```

Flash a light on/off

If a light (LED, etc.) is connected to a digital output, then you can flash it one second on and one second off.

```
RA1S
 1CV=Not(1CV) ‘If 1 CV=0 then 1CV=1. If 1CV=1 then 1CV=0
 1DSO=1CV ‘Digital output = value of 1 CV
```

Bits, Bytes, and Binary stuff

Extract bits from a byte

This piece of code will extract the bits from a byte, for example, if you need bits 7&8 from an 8-bit piece of data.

```
'1cv is the source data
2CV("bits 1&2")=1CV%4
3CV("bits 3&4")=((1CV%16)/4)%1000
4CV("bits 5&6")=((1CV%64)/16)%1000
5CV("bits 7&8")=(1CV/64)%1000
```

OR

```
' Where 1CV is the data source and 6CV & 7CV are the bit numbers
8CV=((1CV%(2^7CV)))/(2^(6CV-1))%1000
```

Note:This code assumes the bits are placed 87654321 within the byte.

Extracting binary words

This code fragment converts 4 bytes them to a Single Precision Floating point number.

This is DT8x code but could be modified for DT800.

```
'=========================================================
        '
        '       CALCULATION OF FLOATING POINT NUMBER FROM 4 BYTE INPUT.
        '
        '
        '       Sensor outputs 4 HEX bytes per item of data
        '       This needs to be converted from IEEE 754 single precision format.
        '       Refer http://en.wikipedia.org/wiki/Single_precision for details.
        '
        '
        '       Convert from Binary to single precision Floating point number
        '
        '       Top bit if MSB hold sign (Bit 32)
        '       Next 7 bits of MSB hold Exponent. (Bits 23 to 30)
        '
        '       Top bit of MSB-1 (Bit 23) Holds low bit of Exponent and needs to be
        '       added to Exponent (Exponent *2 + Lowest bit) to make
        '       8 bit Exponent.
        '
        '       Exponent is then offset by -127 Bias to get final Exponent.
        '
        '       Last 3 bytes (minus top most bit) make up the rest of the number.
        '       and are represented as a binary fraction in the range of 1 > f  > 2.
        '       Binary fraction  Byte 2 = 1/ (2^7)
        '               Byte 3 = 1/ (2^15)
        '               Byte 4 = 1/ (2^23)
        '       Then calculate sum
        '
        '       Note:  Binary fraction  must be in the range 1 > f  > 2.
        '               If less than 1 Add 1 to fraction.
        ‘
        ‘       Floating point = Sign * 2^Exponent * fraction.
        ‘
        ‘==========================================================

        ‘Check for Sign bit and remove if present (15CV Holds sign)
        15CV(“Sign Bit”,W)=(-1*(10CV>127.5))+(10CV<127.5)            ‘15CV = -1 or 1
        IF(10CV>127.5){10CV(W)=10CV-128}                             ‘Remove Most Significant Bit if set

        ‘Check for Exponent low bit and remove if present (16CV Hold sign bit)
        16CV(“Exp Low Bit”,W)=11CV>127.5                             ‘16CV= 1 or 0
        IF(11CV>127.5){11CV(W)=11CV-128}                             ‘Remove Most Significant Bit if set

        ‘Calculate Exponent and remove bias
        20CV(“Exponent bias”,W)=((10CV*2)+16CV)    ‘If Exponent = 255 then Not a number (Check at end)
        17CV(“Exponent”,W)=((10CV*2)+16CV)-127     ‘Add Exp low bit to Exp*2 then remove 127 bias.

        ‘Calculate binary fraction and check if in range of 1 > f > 2
        18CV(“Bin Frac”,FF7,W)=(11CV/(2^7))+(12CV/(2^15))+(13CV/(2^23))        ‘Sum binary fraction
        IF(18CV<1){18CV(W)=18CV+1}                                   ‘If binary fraction < 1 then Binary fraction +1

        ‘Calculate floating point number. (Sign*2^Exponent*Fraction)
        19CV(“Number”,W)=15CV*(2^17CV)*18CV

       ‘If Not A Number or +/- Infinity. Set 98989 as error
        IF(20CV>254.5)”Response Not a Number^M^J”{19CV(W)=98989}
        19CV(“Number”)
```

Returning integer part of a real number

This code fragment will return an integer from a floating point number.

```
1CV(FF7)=123.456
2CV(ff7)=1CV-(1CV%1)
```

Starlog Precision Water Level Meter

This code and wiring applies to the DT8x. The DT800 & the 50/500/600 series loggers would be very similar but would require different wiring. This device uses the UNIDATA HSIO interface. This code and wiring should work for ALL HSIO devices.

```
Wiring
DT8x 7D   - Red Wire (+5VDC)
DT8x DGND - Green Wire (COMMON)
DT8x 5D   - Blue Wire (Data)
DT8x 6D   - Yellow Wire (Clock)
```

DT8x code

```
BEGIN"6541AEx"
RA1S
  7DSO(W)=1 DELAY(w)=10
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=1CV
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*2
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*4
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*8
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*16
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*32
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*64
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*128
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*256
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*512
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*1024
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*2048
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*4096
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*8182
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*16384
  6DSO(W)=1 5DS(=1CV,W) 6DSO(W)=0 2CV(W)=2CV+1CV*32767
  7DSO(w)=0
  2CV("Level~mm")
END
```

DT81 wiring

Because the DT80 only has 1 active drive digital channel equivalent to channels 5–8 on the DT8x, we have to provide pull ups for channels 1 & 2. The value of the pull-up resistor is 1k, however other values may work. Also the LM2940T-5 regulator is probably not necessary. However, not knowing the internal circuit of the water level meter and given that the manual specifies 5 V rather that 6 V if connected without the battery, I felt it is prudent to provide the regulator. The following is the Parts List, Wiring, and Program.

```
Parts List
1. LM2940T-5
2. 2x 1k resistors
DT81 Wiring
DT81 6V battery - LM2940T-5 Input
DT81 GND        - LM2940T-5 Gnd
LM2940T-5 Output- No1 1k side A
LM2940T-5 Output- No2 1k side A
DT81 4D   - Blue Wire (Data)
DT81 2D   - Red Wire (+5VDC) - No1 1k side B
DT81 DGND - Green Wire (COMMON)
DT81 1D   - Yellow Wire (Clock) No2 1k side B
```

DT81 code

```
BEGIN"6541AEx"
RA1S
  2DSO(w)=1 DELAY(w)=10
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=1CV
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*2
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*4
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*8
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*16
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*32
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*64
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*128
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*256
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*512
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*1024
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*2048
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*4096
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*8182
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*16384
  1DSO(W)=1 4DS(=1CV,W) 1DSO(W)=0 2CV(W)=2CV+1CV*32767
  2DSO(w)=0
  2CV("Level~mm")
END
```

Dimetix DLS–A Distance Laser Sensor

The simplest method to communicate with the DLS is simply to have the DT8x/800 poll the unit for measurement when required.

The DLS is polled to return a distance on request from the DT8x/800

The DLS command is;
sNg

Where s = command header N = Device number (0 for default) = carriage return / line feed combination.

RS232 wiring configuration

- DLS Unit Serial port “”RX””
- DLS Pin Number “”1””
- DT80 Serial Sensor Port “”Tx””

- DLS Unit Serial port “”Tx””
- DLS Pin Number “”2””
- DT80 Serial Sensor Port “”Rx””

- DLS Unit Serial port “”Gnd””
- DLS Pin Number “”14 & 15””
- DT80 Serial Sensor Port “”Power GND””

- DLS Unit Serial port “”Power””
- DLS Pin Number “”7 & 8””
- DT80 Serial Sensor Port “”Power supply Positive””

Notes:

- Pins 14 and 15 on DLS should be linked
- Pins 7 and 8 on DLS should be linked
- D Gnd on DT80, Gnd of DLS and Power supply ground must be connected
- External 9 to 30 VDC power supply required for DLS

DLS Default communications settings

The default communications settings on the DSL are

- Baud rate = 19200 • Data bits = 7 • Parity = Even • Stop bits = 1 • Address = 0
- Note: While RS232 is not addressable the address is required to talk to the DLS
- DeTransfer program 1

```
•       begin"DLS" 'Start DLS program
•       '=======================================================
•       '
•       '       Program to read a Dimetix Distance Laser Sensor
•       '                           20 January 2006
•       '                         support@datataker.com.au
•       '
•       '=======================================================
•       ps=19200,e,7,1    'Set SSP to 19200 baud, Even parity, 7 data bits, 1 Stop bit
•       ra1s              'Report schedule A every 1 second
•        1serial(rs232,"\\e{s0g^M^J}g0g%f[1cv]",w,1) 'Poll DLS and read result to 1CV
•        1cv("Distance ~mm")=1cv/10      'Divide reading by 10 to give distance in mm
•       logon             'Turn logging on
•       end               'End of program
```

DeTransfer program 2 (With full error checking)

```
begin”DLS” ‘Start DLS program
‘======================================================
‘
‘       Program to read a Dimetix Distance Laser Sensor
‘
‘               This example had error trapping added
‘
‘                               25 January 2006
‘
‘               For further detail please contact
‘
‘                       support@datataker.com.au
‘
‘       Note; Code is for DT8x/800 V5.xx and above firmware
‘
‘======================================================
‘Set SSP to 19200 baud, Even parity, 7 data bits, 1 Stop bit
ps=19200,e,7,1
‘Initialize Channel variables
1..3cv(w)=0
‘Report schedule A every 1 second
ra1s
‘Poll DLS for measurement and read result
 1serial(rs232,”\\e{s0g^M^J}%4s[‘g0g+’,’g0@E’,2cv]%f[1cv]”,1,=3cv,w)
 alarm(2cv><1,999)and
 alarm1(1cv><204,204.1)”E? Dimension error^M^J”{[1cv=99999]}
 alarm(2cv><1,999)and
 alarm2(1cv><252,252.1)”E? High temperature^M^J”{[1cv=99999]}
 alarm(2cv><1,999)and
 alarm3(1cv><253,253.1)”E? Low temperature^M^J”{[1cv=99999]}
 alarm(2cv><1,999)and
 alarm4(1cv><255,255.1)”E? Signal too weak^M^J”{[1cv=99999]}
 alarm(2cv><1,999)and
 alarm5(1cv><256,256.1)”E? Signal too strong^M^J”{[1cv=99999]}
 alarm(2cv><1,999)and
 alarm6(1cv><257,257.1)”E? Excessive background light^M^J”{[1cv=99999]}
 alarm(2cv><1,999)and
 alarm7(1cv>260)”E? Hardware error^M^J”{[1cv=99999]}
 alarm8(3cv>1)”Hardware timed out^M^J”
‘Divide reading by 10 to give distance in mm
 1cv(“Distance ~mm”)=1cv/10
logon           ‘Turn logging on
end             ‘End of program
```

Removing small numbers from data

Data acquired from the real world is very rarely exactly zero. and the data acquisition systems will return very small floating point numbers.

Code for V9.08 firmware and latter

Version 9.08 introduces an inline If Else (?:)

```
1I(=1CV,W)              ‘Read a voltage and save in a channel variable
2CV(“Current~mA”)=(1CV>1)?0:1CV ‘If the current is less than 1 mA return a 0 else return mA reading
```

Code for up to V9.08 firmware

The code below uses Boolean logic to suppress very small numbers below a preset threshold level.

```
1I(=1CV,W)              ‘Read a voltage and save in a channel variable
2CV(“Current~mA”)=1CV*(1CV>1) ‘If the current is less than 1 mA return a 0 else return mA reading
```

Feed drum

This code is to calculate the total feed from a drum spool. The sensor has a digital output for feed direction and a count for the distance. This code returns the total feed each time the direction changes.

Direction of feed is digital input 1 Distance of feed is high Speed Counter 1

```
BEGIN

RA2E
   1DS(=1CV,W)        ‘Read direction of feed
   2CV=-1*(1CV<0.5)+(1CV>0.5,W)  ‘Feed out = 1 Feed in = -1
   1HSC(=3CV,R,W)     ‘Read feed count
   4CV=4CV+3CV*2CV     ‘Total feed
END
```

Starting and stopping jobs using DT80/85 keypad and display functions

The following code can be used to start and stop a job with a menu. This code will also limit the file size to less that 32,000 records. This is useful because EXCEL can only graph 32,767 points. 9 CV is used as the counter. This should of course be set to a channel variable you are not currently using. If a memory stick is installed, then the data is ‘moved’ to the memory stick or else it is moved to internal store. When the Move Data button is pressed, all the 32,000 record ‘moved’ files are then moved to a memory stick or else the current data in the store is moved as a separate file to the internal store of the logger. The additional button will change the speed of logging, if appropriate.

Note: You will need to set the name of your particular job in the RUNJOB statement. These are the lines which are shown **thus**

Immediate code (Place After the BEGIN statement)

```
FUNCTION1=”Set&Go Job1”{RUNJOB”Job1”}
FUNCTION2=”Stop”{H}
FUNCTION3=”DEL Data”{deldata* Delalarms* 9CV=0}
FUNCTION4=”MOVE Data”{movedata 9CV(W)=0}
FUNCTION5=”15S rate”{RA15S}
FUNCTION6=”5S rate”{RA5S}
FUNCTION7=”1s rate”{RA1S}
9CV(W)=0
```

Schedule Code (This code is place in the recording schedule usually after the RA. statement)

```
9CV(W)=9CV+1
IF(9CV>32000){MOVEDATA 9CV=0}
```

Using digital inputs to synchronize two or more DataTakers

The concept is to use one DataTaker as a master and the others as slaves. The master sends a digital output which is used to synchronize the slave loggers. The slave loggers trigger their schedules on the falling edge of a digital input (Digital input 1 used in the examples). It is then a simple matter of connecting the digital grounds and digital 1 on all the DataTakers together.

Master Program

```
BEGIN”Master”

RA1M
   1DSO=0   ‘Turn digital input 1 off
   ‘Put your list of channels here
   1DSO=1   ‘Turn digital input 1 on when finished.
END
```

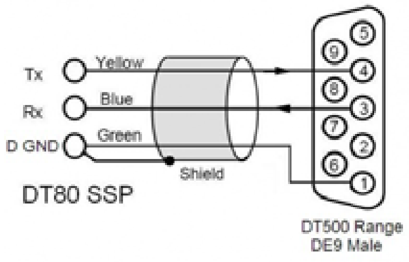
Slave Program

```
BEGIN”Slave1”

RA1-E ‘trigger the schedule on a falling edge of digital input 1

   ‘Put your list of channels here

END
```

Connecting DT5xx loggers to DT80 range loggers

Connecting a single DT5xx data logger

One or multiple DT5xx range loggers can be connected to a single DT80 range data logger to provide expansion or a gateway to additional DT80 range features (e.g. TCP/IP, RS232, MODBUS, USB slave and USB Flash memory). A single DT5xx range logger can be connected using RS232 via the DT80 range Serial Sensor Port, while additional DT5xx range loggers would be connected on the DataTaker DT5xx (RS485) network. The DT80 can then be used as a single contact point for down loading data. These examples should be used as a guide in conjunction with the relevant sections from the DT500 and DT80 range User Manuals.



The preferred connection is via the DT80 range Serial Sensor Port using RS232.

Alternatively, you can connect to the RS232 Host port using the DT500 Range PC to DataTaker Serial Cable (P/N PROIBM-1), this requires additional settings and DT80 range firmware of at least Version 6.18 to enable the 2SERIAL function of this port.

Configuring the DT80 range data logger for a single DT5xx range logger

In this example, the DT80 range logger will poll a single DT5xx range logger via the RS232 Serial Sensor Port. The DT5xx will be polled to read the temperature of thermocouples attached to inputs 1 to 5. The DT80 range data logger then stores the returned data. Two methods are shown–the 1st is more efficient as it polls the DT5xx logger only once to obtain data, and the 2nd requires individual polling for each channel but may be easier for some users to understand. The 1st method uses Channel Variables and will run considerably faster than the 2nd method due to this efficiency.

```
BEGIN
PS=RS232,9600,N,8,1,SWFC  ‘Configure the Serial Sensor Port
 RA5S ‘Scan and report values every 5 seconds
‘Clear the serial buffer, configure the DT5xx and scan the DT5xx temperature, save results to channel variables.
 1SERIAL(rs232,”\\e{P22=44 /u/n/m/e/R 1..5TK^M^J}%f[1cv],%f[2cv],%f[3cv],%f[4cv],%f[5cv]”,w)
 1..5CV(“Temperature ~DegC”) ‘Log the results
LOGON
END
```

OR

```
BEGIN
PS=RS232,9600,N,8,1,SWFC  ‘Configure the Serial Sensor Port
 RA5S
‘Clear the serial buffer, configure and scan the DT5xx for temperature on Channel 1, save result as “Temp01’.
 1SERIAL(“\\e{P22=44 /n/u/m/e/R 1TK^M}%f”,2,”Temp01~DegC”)
‘Clear the serial buffer and scan the DT5xx for temperature on Channel 2, save result as “Temp02’. repeat for
Channel 3-5.
 1SERIAL(“\\e{2TK^M}%f”,2,”Temp02~DegC”)
 1SERIAL(“\\e{3TK^M}%f”,2,”Temp03~DegC”)
 1SERIAL(“\\e{4TK^M}%f”,2,”Temp04~DegC”)
 1SERIAL(“\\e{5TK^M}%f”,2,”Temp05~DegC”)
LOGON
END
```

Alternative connection using RS232 Host port in Serial Sensor Mode

Functionality is as per the 1st example above but using the RS232 Host port configured for Serial Sensor mode. Firmware of at least Version 6.18 is required to enable the 2SERIAL function of this port. Connection is made using the DT500 Range PC to DataTaker Serial Cable (P/N PROIBM-1).

```
‘Clear existing jobs and data from the logger.
H           ‘Halt the current job
DELDATA*    ‘Deletes all data from all jobs.
DELALARM*   ‘Deletes all alarms from all jobs.
DELJOB*     ‘Deletes all jobs.

‘Configure the Host Serial Port as Serial Sensor Port 2SERIAL with correct baud rate for DT5xx/6xx connection.
PROFILE “HOST_PORT” “BPS”=”9600”
PROFILE “HOST_PORT” “FUNCTION”=”SERIAL”

‘Restart the logger and apply new Profile settings
SINGLEPUSH
‘At this point the logger will restart, allow the logger to restart before sending the remainder of this code.

BEGIN
 RA5S ‘Scan and report values every 5 seconds
‘Clear the serial buffer, configure the DT5xx and scan the DT5xx temperature, save results to channel variables.
 2SERIAL(“\\e{P22=44 /u/n/m/e 1..5TK^M^J}%f[1cv],%f[2cv],%f[3cv],%f[4cv],%f[5cv]”,w)
 1..5CV(“Temperature ~DegC”) ‘Log the results
LOGON
END
```

Connecting and Configuring multiple DT5xx data loggers

If connecting multiple DT5xx range data loggers, you will need to configure each DT5xx for use with the Datataker RS485 network. Each units requires a unique address number that is set using an internal DIP switch. The DT5xx range DataTakers are then connected together in the network configuration as described on page 14 of the DT500 User Guide. Only one of the DT5xx range loggers will also be connected to the DT80 range Serial Sensor Port using RS232 as described above.

Configuring the DT80 range data logger for multiple DT5xx range loggers

In this example, the DT80 range logger will poll two networked DT5xx loggers in sequence via the RS232 connection to the first DT5xx. Each DT5xx will be polled to read temperature of thermocouples attached to inputs 1 to 5. The DT80 range data logger then stores the returned data. Two methods are shown–the 1st is more efficient as it polls each DT5xx logger only once to obtain data, while the 2nd requires individual polling for each channel but may be easier for some users to understand. The 1st method uses Channel Variables and will run considerably faster than the 2nd method due to this efficiency.

```
BEGIN
PS=RS232,9600,N,8,1,SWFC  'Configure the Serial Sensor Port
 RA5S 'Scan and report values every 5 seconds
'Clear the serial buffer, configure the DT5xx and scan the DT5xx address #0 for temperature, save results to
channel variables.
 1SERIAL(rs232,"\\e{#0 P22=44 /u/n/m/e 1..5TK^M^J}%f[1cv],%f[2cv],%f[3cv],%f[4cv],%f[5cv]",w)
'Clear the serial buffer, configure the DT5xx and scan the DT5xx address #1 for temperature, save results to
channel variables.
 1SERIAL(rs232,"\\e{#1 P22=44 /u/n/m/e 1..5TK^M^J}%f[6cv],%f[7cv],%f[8cv],%f[9cv],%f[10cv]",w)
 1..10CV("Temperature ~DegC") 'Log the results
LOGON
END
```

OR

```
BEGIN
PS=RS232,9600,N,8,1,SWFC  'Configure the Serial Sensor Port
 RA10S
'Clear the serial buffer, configure and scan the DT5xx address #0 for temperature on Channel 1, save result as
"Temp01'.
 1SERIAL("\\e{#0 P22=44 /n/u/m/e 1TK^M}%f",2,"Temp01~DegC")
'Clear the serial buffer and scan the DT5xx address #0 for temperature on Channel 2, save result as "Temp02'.
repeat for Channel 3-5.
 1SERIAL("\\e{#0 2TK^M}%f",2,"Temp02~DegC")
 1SERIAL("\\e{#0 3TK^M}%f",2,"Temp03~DegC")
 1SERIAL("\\e{#0 4TK^M}%f",2,"Temp04~DegC")
 1SERIAL("\\e{#0 5TK^M}%f",2,"Temp05~DegC")
'Clear the serial buffer, configure and scan the DT5xx address #1 for temperature on Channel 1, save result as
"Temp11'.
 1SERIAL("\\e{#1 P22=44 /n/u/m/e 1TK^M}%f",2,"Temp11~DegC")
'Clear the serial buffer and scan the DT5xx address #0 for temperature on Channel 2, save result as "Temp02'.
repeat for Channel 3-5.
 1SERIAL("\\e{#1 2TK^M}%f",2,"Temp12~DegC")
 1SERIAL("\\e{#1 3TK^M}%f",2,"Temp13~DegC")
 1SERIAL("\\e{#1 4TK^M}%f",2,"Temp14~DegC")
 1SERIAL("\\e{#1 5TK^M}%f",2,"Temp15~DegC")
LOGON
END
```

In Australia:
For customer service, call 1300-735-292
To email an order, ordersau@thermofisher.com

In New Zealand:
For customer service, call 0800-933-966
To email an order, ordersnz@thermofisher.com

Find out more at **thermofisher.com/datataker**

ThermoFisher SCIENTIFIC