dataTaker

Training Note

TR-0011

DT80 Modbus Master

Firmware 8.06
October 2010

CONTENTS

DT80 Modbus Master

Introduction

Modbus is a simple communications protocol which is widely used in SCADA (supervisory control and data acquisition) systems. Modbus provides an efficient and standardized way to transport digital states and data values between a remote terminal unit (RTU) or programmable logic controller (PLC) and a supervisory computer.

In a Modbus-based SCADA system, each RTU/PLC acts as a Modbus server, or slave. These servers / slaves listen for and reply to requests from a Modbus client, or master system. A Modbus client is typically a computer that provides a mimic display, user interface and various data logging and alarm functions.

Modbus Master (Client) capability will be added with the next firmware release after version 7.10.

Application

The DT80 range having both master and slave capabilities will open new opportunities including:

- Integration into Industrial SCADA systems.
- Ability to use a new range of Modbus capable sensors.
- Real time data exchange between loggers.
- Gateway device.
    - SDI-12 to Modbus
    - RS232, RS422, RS485 to Modbus.
    - Modbus to web server

Limitations

The DT80 range Modbus Master has the following limitations.

- Does not support Modbus ASCII.
- Does not support long (32 bit) unsigned integer data type.

Modbus communications methods

Modbus can operate using a broad range of communications media. These fall into two main categories:

- Serial connection, typically RS232, RS422 or RS485
- TCP/IP network, which can use a variety of physical link types e.g. Ethernet, wireless, fibre-optic, serial (PPP)

The DT80 supports both TCP/IP and serial Modbus networks.

Serial Connection

A serial Modbus network has one master system connected to one or more slave devices. Serial networks using the RS485 or RS422 standards support multi-drop, i.e. multiple slaves connected to one master. RS232 can also be used for point-to-point connections (single master and single slave).

Slave devices on a serial Modbus network are identified by an 8-bit slave address (1-247). Every slave device on a particular serial network must have a unique address.

The DT80 can be connected to a serial Modbus network using either the serial sensor port, the host RS232 port, or the USB port. The serial sensor port on the DT80 can be configured for RS232, RS422 or RS485 modes.

TCP/IP Connection

Multiple Modbus Slave devices can simultaneously connect to the DT80 using TCP/IP.
The first step in setting up Modbus over TCP/IP is to establish a working TCP/IP connection between the client system and the DT80. This involves assigning an IP address to the DT80, along with a couple of other settings, depending on whether Ethernet or PPP is used.

By default, the DT80's TCP/IP Modbus server is always enabled. It will listen for connection requests from client systems which are directed to TCP port 502, which is the standard port number for Modbus. Slave addresses are not required on a TCP/IP Modbus network, because the slaves are identified by their IP address.

Modbus register types

The Modbus message consists of:

- Address of the Modbus slave device
  - In the range of 1 to 247 for serial devices
  - TCP/IP address for network devices
- Register type.
  - 0 = Coil (Digital, Read and Write)
  - 1 = Discrete input (Digital Input, Read only)
  - 3 = Input register (16 bit input registers, Read only)
  - 4 = Output register (16 bit output registers, Read and Write)
- Register address.
  - Either a 5 or 6 digit numbered address for the registers. (Device dependant)
  - In the range of 1 to 65535

Notes:

- The register number will depend on the device. Most devices have more than one register and most register may be used in more that one type.
- The devices are likely to support only specific subset of available device range, for one or more register types.
- In general case, the registers of the same address and different type are physically different registers. However some devices are mapping the registers of different types and the same address to physically the same register. See documentation of your device for details.

The Modbus convention for representing a resister type has the register type followed by the register number. A Modbus slave device will have a manufacturer supplied mapping for each valid register type and register number.

The DT80 uses the following mapping for its registers when addressed as a slave device:

Modbus slave register range	***DT80 Channel***
0001 to 1000	Channel variable 1 to 1000
4001 to 4053	System variables 1 to 53
8000 to 8009	Digitals 1 to 8 and 1Relay

Table 1 - DT80 range Modbus registers

Putting this into standard Modbus convention we get:

Modbus register mapping	*DT80 Channel*	*DT80 action*
Register type – Coil	**Comment: Coil type is read / write**	
00001 to 01000	Channel variable 1 to 1000	Returns 0 or 1 (If CV >0 then output = 1)
04001 to 04053	System variables 1 to 53	Returns 0 or 1 (If SV >0 then output = 1)
08000 to 08009	Digitals 1 to 8 and 1Relay	Returns digital state
Register type – Discrete	**Comment: Discrete type is read only**	
10001 to 11000	Channel variable 1 to 1000	Returns 0 or 1 (If CV >0 then output = 1)
14001 to 14053	System variables 1 to 53	Returns 0 or 1 (If SV >0 then output = 1)
18000 to 18009	Digitals 1 to 8 and 1Relay	Returns digital state
Register type - Holding register	**Comment: Holding register is read only**	
30001 to 31000	Channel variable 1 to 1000	Returns current CV value
34001 to 34053	System variables 1 to 53	Returns current SV value
38000 to 38009	Digitals 1 to 8 and 1Relay	Returns current digital value (0 or 1)
Register type - Input register	**Comment: Input register is read / write**	
40001 to 41000	Channel variable 1 to 1000	Returns current CV value
44001 to 44053	System variables 1 to 53	Returns current SV value
48000 to 48009	Digitals 1 to 8 and 1Relay	Returns current digital value (0 or 1)

Table 2 - DT80 range Modbus registers with function numbers

Note: The DT80 range user's manual uses the convention **<Type>:<Register>** e.g 3:0001 to distinguish between the register type and register number instead of the Modbus convention. This is to reduce the confusion caused to users new to Modbus.

As can be seen, the first digit of the register number indicates the type of register - coil, discrete input, input register or output register. This usage is, however, just a convention. This digit is not part of the actual address transmitted in the Modbus message.

A further potential source of confusion is the fact that the actual transmitted address is zero-based, so register number x0003 is actually transmitted as address 0002.

Note: In some Modbus client applications, register numbers are entered using these raw protocol addresses, while in others you specify register numbers including the initial "register type" digit, as described above. The documentation for the particular package should make clear which convention it uses.

The Modbus protocol then defines a set of messages which allow the client to:

- read the current value of one or more of the slave's coils, discrete inputs, input registers or output registers
- write to one or more of the slave's coils or output registers.

A given type of Modbus slave device will support some quantity of each type of resource - for example a hypothetical device might support 16 coils, 16 discrete inputs, 4 input registers and no output registers.

Furthermore, it is common for the different register arrays to overlap. In the example device mentioned above, the 16 coils and discrete inputs may actually refer to the same physical hardware - in this case 16 bi-directional I/O pins. So for this slave device, if a client wrote a "1" to coil 00007, it would then read the same value back if it did a read from discrete input 10007.

Modbus data types

The Modbus standard only defines the size of the registers as being 16 bit wide. It does not detail how the data is to be stored in the registers.

While the Coil and Discrete register types are only 1 bit, the representation of numbers can vary depending on the type of number and the type of computer system being used.

Note: Which ever method of representing data a system is using, it is important that the Modbus master be configured to read the matching data format types from the slave device.

Data Type	Range
16 bit signed integer	-32768 to 32767
16 bit unsigned integer	0 to 65535
32 bit signed integer	-2,147,483,648 to 2,147,483,647
32 bit floating point number	-3.4028e38 to 3.4028e38

Table 3 - Data types and approximate range

To represent decimal numbers with integer data types the slave will take a reading and multiply the number by a scalar then place the scaled number in the register. The master will then read the device register and divide the number by the same scalar.
For example a slave device may read a thermocouple temperature as 23.4 degrees C. If it puts this value in a holding 16 bit signed integer the decimal place will be lost. To overcome this limitation the slave multiples the temperature value by 10 so the register now hold the value 234. When the master reads this value it will divide the register value by 10 to restore the decimal place.

32 bit data types are stored across two consecutive Modbus registers. While the Modbus standard does state that data in a register is in High byte, Low byte format (Big endian) it does not specify the word order (endian) when data is across registers. For this reason you need to be aware of the word order of the slave device when reading 32 bit data.

A variation to Modbus known as ENRON Modbus is also supported by the DT80 range Modbus master implementation. ENRON Modbus supports 32 bit data registers.

Planning a Modbus Task

Because of the level of detail required for implementing a Modbus system it is strongly advised that the task is approached systematically.

Before you start:

- Understand the slave device capabilities.
  - Communication method and set up (RS232)
  - Assign unique device address.
  - Understand Modbus register types
  - Understand Modbus register data types
  - Understand data word order

- Understand the master device capabilities
  - Communication methods and set up
  - Slave device addressing.
  - How to read slave register types
  - How to read slave register data types

  - How to address slave data word ordering

- Design the system
  - Construct a Modbus register map that includes device numbers, register types, data types and details etc.
  - Communications type to be used and settings.
  - Other consideration. Displays, data handling etc.

- Implement and document
  - Program the system and deploy the system
  - Document the system components, structure, design details, etc

Don't forget to document the project. Remember you might be the person who has to repair or modify the system at a future date. Good documentation will greatly simplify the task.

DT80 Modbus commands

Modbus related profile settings

Serial Sensor Port

```
PROFILE SERSEN_PORT FUNCTION=mode
PROFILE SERSEN_PORT FLOW=NOFC
PROFILE MODBUS_SERVER SERSEN_ADDRESS=addr
```

Where:

- *mode* can be either
  - **Modbus** – for Modbus slave mode
  - **Modbus_Master** – for Modbus Master mode
- *addr* is 8 bit Modbus slave address in the range of 1 to 247.

Notes:

- Software flow control interrupts the Modbus protocol and thus should not be used.

Host Port

```
PROFILE HOST_PORT FUNCTION=mode
PROFILE HOST_PORT FLOW=NOFC
```

Where:

- *mode* can be either
  - **Modbus** – for Modbus slave mode
  - **Modbus_Master** – for Modbus Master mode

Notes:

- Software flow control interrupts the Modbus protocol and thus should not be used.
- The Host Port only supports RS232 and therefore can only be point to point. That is, only one Modbus slave device and one Modbus master.
- Use the other Host_Port profile setting to configure the other communications parameters (baud rate etc)

TCP/IP

```
PROFILE MODBUS_SERVER TCPIP_PORT=port
```

Where:

- *port* is the TCP/IP port number Modbus will use to communicate with the DT80. (Default is port 502)

Notes:

- The device address is the TCP/IP address or symbolic DNS address
- With Modbus TCP/IP the DT80 acts as both a master and slave
- Use the Ethernet profile settings to configure network settings
- Use a static TCP/IP address if you don't have a symbolic DNS address for the DT80

Modbus master commands, channel type and parameters

nMODBUS Channel Type

The DT80 uses the channel type *n*Modbus to read or write the registers of Modbus slave devices. Channel options are used to set the register address, data types etc.

Syntax: `nMODBUS(options)=value`

Where:

- *n* = the Modbus master port the command will be sent out (see Table 4 below)
- *options* = the dataTaker channel options

Port Number *n*	Physical connection
1	Serial sensor port
2	Host port
4	Ethernet port

Table 4 - DT80 range port numbers

nMODBUS Channel options

Address (ADn)

Where:

- n = address of slave device on serial link (1/2/3MODBUS): 1-247,
  default=1, 0=broadcast to all slaves (write-only)

Example 1: `1MODBUS(AD12,R3:2)`

Modbus read on the serial sensor port, Slave address 12, reading a 16 bit signed integer, register type 3 register 2.

Example 2: `2MODBUS(AD2,R3:500)`

Modbus read on the host port, Slave address 2, reading a 16 bit signed integer, register type 3 register 500.

Address (AD"ip_address")

Where:

- "ip_address" is the IP address of slave device on TCP/IP network (4MODBUS only)

Note: The IP AddressMay be numeric IP address or symbolic DNS address.

Example 1:

```
4MODBUS(AD"192.168.1.212",R3:2)
```

Modbus read on the Ethernet port, Slave TCP/IP address 192.168.1.212, reading a 16 bit signed integer, register type 3 register 2.

Example 2:

```
4MODBUS(AD"DT80.com.au",R3:2,MBI)
```

Modbus read on the Ethernet port, Slave TCP/IP address is Modbus.DT80.com.au, reading a 16 bit signed integer, register type 3 register 2.

Register (Rt:rrr:b)

Where:

- t = register type, one of
  - 0 = coil
  - 1 = discrete input
  - 3 = input register
  - 4 = output register
- rrr = Register number (1-65536)
- b = number of digital bits to pack into a single value (1-16, default=1).
  - Only valid for digital register types (t=0/1)

Example 1:

```
2MODBUS(AD12,R0:8001)=0
```

Modbus write on the Host port, Slave address 12, Register type 0 (Coil) register 8001. If connected to a DT80 this will set digital output 1 to 0.

Example 2:

```
2MODBUS(AD12,R1:8001)
```

Modbus read on the Host port, Slave address 12, Register type 1 (Discrete) register 8001. If connected to a DT80 this will read the current value of digital input 1.

Example 3:

```
2MODBUS(AD12,R3:1)
```

Modbus read on the Host port, Slave address 12, Register type 3 Input register 1. If connected to a DT80 this will read the current value of 1CV.

Example 4:

```
2MODBUS(AD12,R4:1)=600
```

Modbus read on the Host port, Slave address 12, Register type 4 Output register 1. If connected to a DT80 this write 600 to 1CV.

Example 5:

```
2MODBUS(AD12,R0:10:4)=16
```

Modbus read on the Host port, Slave address 12, Register type 0 coil register 10, bits 1 to 4. This will set bits 1 to 4 to 1

Data Type/Format (MBxx)

Where:

- xx = the data type, one of
  - I = 16 bit signed integer (default)
  - U = 16 bit unsigned integer
  - L = 32 bit signed integer transferred using two consecutive 16 bit registers
  - F = 32 bit float transferred using two consecutive 16 bit registers
  - LE = 32 bit signed integer transferred using "Enron Modbus" protocol variant

Example 1: `1MODBUS(AD12,MBI,R3:1)`

Modbus read on the serial sensor port, Slave address 12, Register type 3 Input register 1 as a signed integer.

Example 2: `1MODBUS(AD12,MBU,R3:1)`

Modbus read on the serial sensor port, Slave address 12, Register type 3 Input register 1 as a unsigned integer.

Word Order/Endian (MEx)

Where:

- x = the word order, one of
  - s = "straight endian", most significant word first(default)
  - r = "reverse endian", least significant word first

Note: The word order only applies to MBL or MBF data types

Example 1: `2MODBUS(AD12,MBF,MES,R3:1)`

Modbus read on the Host port, Slave address 12, Register type 3 Input register 1 as a 32 bit floating point number. Transfer is largest word first then small word second.

Example 2: `2MODBUS(AD12,MBF,MER,R3:1)`

Modbus read on the Host port, Slave address 12, Register type 3 Input register 1 as a 32 bit floating point number. Transfer is mall word first then large word second.

Timeout (TOn)

Where:

- n = the timeout for Modbus master requests (in seconds)

Note: The value must be between 1 and 255, with the default being 3.

Example : `2MODBUS(AD12,R3:2,TO10)`

Modbus read on the serial sensor port, Slave address 12, reading a 16 bit signed integer, register type 3 register 2. Time out between retries set to 10 seconds

Retries (RTn)

Where:

- n = the number of retries to perform following a Modbus master error or timeout

Note: The value must be between 0 and 25, with the default being 0

Example:
```
1MODBUS(AD12,R3:2,RT3)
```
Modbus read on the serial sensor port, Slave address 12, reading a 16 bit signed integer, register type 3 register 2. Three retires with default time out of 1 second.

Unit ID (MUIDn)

Where:

- n = the Modbus "unit id" field. Typically used when communicating to a serial slave via a TCP/IP connected gateway device.

Example:
```
4MODBUS(AD"192.168.1.212",R3:2,MUID12)
```

Modbus read on the Ethernet port, Modbus gateway TCP/IP address 192.168.1.212, Modbus Unit ID number 12, reading a 16 bit signed integer, register type 3 register 2.

Assign to Channel Variables (=m..nCV)

Where:

- m = the first channel variable to assign
- n = the final channel variable to assign

This channel option allows for multiple, consecutive reads to be performed from a slave device and stores the result(s) to a range of channel variables.

Example 1:
```
1MODBUS(AD12,R3:100,=1..5CV)
```

Read from slave at address 12, register type 3, registers #100-105 into channel variables 1CV to 5CV.

Example 2:
```
2MODBUS(AD12,R4:1)=10..15CV
```

Write the value of channel variables 10 to 15 to Modbus registers #1 to 5.

Reverse Span (SRn)

Where:

- n = the span number

This channel option applies a defined span in reverse.

Example:
```
S1=0,100,4,20      'Span to convert 4-20 signal to pressure
1MODBUS(AD12,R3:100,=1CV,S1) 'read 4-20 value from Modbus
1CV("pressure")              'store value as pressure
1MODBUS(AD10,R4:10,RS1) =1CV 'write to other device as 4-20
```

Read a value in from one Modbus device, scale and store it, then write the value to another Modbus device in original units

Channel Factor

When working with integer data types it is not possible to represent data with decimal places. It is common for the Modbus slave device to multiply the value by a scalar (ie. *10, *100 etc.) then transmit the scaled value. The Modbus master device must then divide the read value by the scalar to restore the decimal places.

Example 1: `1MODBUS(AD12,MBI,R3:1,0.1)`
Modbus read on the serial sensor port, Slave address 12, Register type 3 Input register 1 as a signed integer. The channel factor of 0.1 will divide the input by 10 thus restoring one decimal place.

Troubleshooting

Error numbers and messages

E124 - Modbus transaction failed
Modbus communications has failed.
Check:
- Communications cabling.
- Slave device is powered.
- TCP/IP network is active
- Software flow control is turned off on both master and slave.

E125 - Modbus - write attempt to read-only register
Attempting to write data to a read only register.

E127 - Modbus IP address specified on serial channel
Attempting to use TCP/IP address or domain name with a serial channel type.
e.g. 1Modbus(AD"192.168.1.212") or 2Modbus(AD"My.DT80.com")

Note: For this addressing mode the serial channel will need to be configured for PPP communications

E128 - Modbus serial address specified on TCP/IP channel
Attempting to use Serial address with a Ethernet channel type.
e.g. 4Modbus(AD192). Can also occur is serial channel is configured for PPP.

E129 - Modbus - unexpected format of response packet
Returned data in not in Modbus format. Most likely cause is device returning data is not Modbus device.

E130 - Modbus - unexpected function id in response.
Modbus slave returned incorrect function number.

E131 - Modbus - exception response received
Error in response from Modbus slave device. Probably due to slave device not supporting requested Modbus function.

E132 - Modbus - error in definition of CV block to send
Error in defining channel variables to be written to registers.
e. g. 2MODBUS(AD12,R4:1)=100..10cv

E133 - Modbus - reading values to CVs prevents writing
Can not read and write to channel variables in the one transaction
e.g 2MODBUS(AD12,R4:1,=10CV)=100CV

E134 - Modbus - 32 bit format not applicable to discrete transfer
Attempt to write a floating point number to a discrete input.

E135 - Modbus - volume of data to transfer exceeds Modbus message capacity
Attempting to read or write to too many registers in one try
e.g. 2MODBUS(AD12,R3:1,=1..500CV)

Note: Maximum Modbus message length is 255 bytes.

Modbus diagnostics P56=4

Setting P56=4 will turn on the Modbus debug output
When in this mode the Modbus communications between the master and slave devices will be returned.

Example:
This example is the conversation from a DT80 configured as a Modbus master and a Modbus slave device and show a successful Modbus transaction and a CRC failure.

```
1CV=1 2CV=2 3CV=3 4CV=5 5CV=0
2MODBUS(AD12,R4:1,=1..5CV)
Modbus TX >HOST: 0c 04000000053114 (8)
Modbus RX <HOST: 0c 040a0001000200030004000e9a1 (15)
2MODBUS 5

2MODBUS(AD12,R4:1)=1..10CV
Modbus TX >HOST: 0c
100000000a14000100020003000400050006000700080009000023d6 (29)
INVALID RX MSG (calc CRC=c130)
Modbus CRC check failed
Modbus RX <HOST: 0c 100000000a41 (7)
dataTaker 80 E124 - Modbus transaction failed
2MODBUS -9e9
```

Data type range limiting

When writing values to a Modbus slave device eg 4MODBUS(..)=1CV, if the value of the expression is outside the range of the target Modbus register then it will be set to the register's max/min value.

For example the following command 2MODBUS(ad12,MBI,R4:1)=483648 takes the value 483648 and tries to write it to a 16 bit signed integer. As the maximum value a 16 bit signed integer can hold is 32767 then the value 32767 is transmitted to the slave device.

Worked example

Configuring the DT80 as a Modbus master

Example:

A DT80 will be used to read data from the Modbus registers of other DT80's that are being used to monitor the performance of 3 environmental chambers.

The DT80 Modbus master will read the registers for average temperature and power state from each of the three environmental chambers and log the data.

Modbus communications will be over a TCP/IP network.

The DT80's attached to the environmental chambers will read 4 PT100 RTD temperature sensors and display the temperature with one decimal place. The average of the 4 temperature will also be displayed. The average reading will be tested in an alarm for over or under range condition, set point +/- 1 Deg C. The external power state will also be read from 5 SV.
Modbus communications will be via Ethernet.

Note: 5SV (System Variable) holds 1 if mains power is connected, 0 if disconnected.

Planning

The first step is to gather the Modbus register maps and Modbus addresses for each device.

Modbus Mapping Table				
Device details:				
Name	DT80			
Serial Number	085533			
Location	Chamber A			
Communications	Modbus TCP/IP			
Modbus address	192.168.11.69 on Port 502			
Modbus register details				
Modbus Register	Register type	Data type	DT80 Channel	Description
30001	3 – holding reg	Signed int	1CV	Temperature 1
30002	3 – holding reg	Signed int	2CV	Temperature 2
30003	3 – holding reg	Signed int	3CV	Temperature 3
30004	3 – holding reg	Signed int	4CV	Temperature 4
30005	3 – holding reg	Signed int	5CV	Average Temp.
04005	1 – Discreet	Digital	5SV	Power On / Off
Comments: Registers 1 to 5 have been multiplied by 10. All registers are read only				

Table 5 - Modbus register map. Chamber 1

Modbus Mapping Table				
Device details:				
Name	DT80			
Serial Number	085533			
Location	Chamber B			
Communications	Modbus TCP/IP			
Modbus address	192.168.11.70 on Port 502			
Modbus register details				
Modbus Register	Register type	Data type	DT80 Channel	Description
30001	3 – holding reg	Signed int	1CV	Temperature 1
30002	3 – holding reg	Signed int	2CV	Temperature 2
30003	3 – holding reg	Signed int	3CV	Temperature 3
30004	3 – holding reg	Signed int	4CV	Temperature 4
30005	3 – holding reg	Signed int	5CV	Average Temp.
04005	1 - Discreet	Digital	5SV	Power On / Off
Comments: Registers 1 to 5 have been multiplied by 10. All registers are read only				

Table 6 - Modbus register map. Chamber 2

Modbus Mapping Table				
Device details:				
Name	DT80			
Serial Number	085533			
Location	Chamber C			
Communications	Modbus TCP/IP			
Modbus address	192.168.11.71 on Port 502			
Modbus register details				
Modbus Register	Register type	Data type	DT80 Channel	Description
30001	3 – holding reg	Signed int	1CV	Temperature 1
30002	3 – holding reg	Signed int	2CV	Temperature 2
30003	3 – holding reg	Signed int	3CV	Temperature 3
30004	3 – holding reg	Signed int	4CV	Temperature 4
30005	3 – holding reg	Signed int	5CV	Average Temp.
04005	1 - Discreet	Digital	5SV	Power On / Off
Comments: Registers 1 to 5 have been multiplied by 10. All registers are read only				

Table 7 - Modbus register map. Chamber 3

Inspecting the Modbus register maps for these devices it can be seen that the data is read only and we need to read:

1. Holding Register 30005 for the average temperature
2. Discreet input register 04005 for power state.
3. TCP/IP address of each unit.

Implementation

Profile settings.
In this case there are no profile settings required to configure the DT80 as a Modbus master device other than the network setting needed to establish TCP/IP communications.

DT80 Code.

```
BEGIN"MASTER"
RA10S
      4MODBUS(AD"192.168.11.69",R0:4005,"CHAMBER A POWER")
      4MODBUS(AD"192.168.11.69",R3:5,MBI,"CHAMBER A (AVE)~\176C",0.1)

      4MODBUS(AD"192.168.11.70",R0:4005,"CHAMBER B POWER")
      4MODBUS(AD"192.168.11.70",R3:5,MBI,"CHAMBER B (AVE)~\176C",0.1)

      4MODBUS(AD"192.168.11.71",R0:4005,"CHAMBER C POWER")
      4MODBUS(AD"192.168.11.71",R3:5,MBI,"CHAMBER C (AVE)~\176C",0.1)

LOGON
END
```