dataTaker®

Intelligent Data Logging Products

DT80
DT81
User's Manual

A complete guide to:

- data acquisition
- data logging
- programming
- sensor wiring
- communications

www.datataker.com

DT80 Series User's Manual

UM-0085-A2

Warranty

Datataker Pty Ltd warrants the instruments it manufactures against defects in either the materials or the workmanship for a period of three years from the date of delivery to the original customer. This warranty is limited to the replacement or repair of such defects, without charge, when the instrument is returned to dataTaker or to one of its authorized dealers.

This warranty excludes all other warranties, either express or implied, and is limited to a value not exceeding the purchase price of the instrument.

Datataker P/L shall not be liable for any incidental or consequential loss or damages resulting from the use of the instrument, or for damage to the instrument resulting from accident, abuse, improper implementation, lack of reasonable care, or loss of parts.

Where Datataker P/L supplies to the customer equipment or items manufactured by a third party, then the warranty provided by the third party manufacturer remains.

Trademarks

dataTaker is a registered trademark of Datataker Pty Ltd.

All other brand and product names are trademarks or registered trademarks of their respective holders.

Related Software Products

DeLogger, DeLogger Pro, DeTransfer, DeLoad, DeView

dataTaker ActiveX, dataTaker LabVIEW™ instrument driver

DT80 Firmware Covered in This Manual

This version of the *DT80 dataTaker User's Manual* (UM-0085-A2) applies to *DT80* series products (DT80 and DT81) running **version 6.02 (or later)** firmware.

WARNING

dataTaker products are not authorized for use as critical components in any life support system where failure of the product is likely to affect the system's safety or effectiveness.

List of Major Tables

Contents

Part A — The *DT80*

Figure 1: The dataTaker DT80 (left) and DT81 (right)

DT80 Concepts

What is the *DT80*?

The *dataTaker DT80 series* data acquisition and logging instruments are tools to measure and record a wide variety of quantities and values in the real world.

With the *DT80* series loggers, basic measurement tasks are easy. For example, sending the command line

```
RA5S 1..4TJ LOGON
```

declares a report schedule (**RA**) that reports every five seconds (**5S**) the temperatures on four type J thermocouples connected to the *DT80*'s analog input channels 1 to 4 (**1..4TJ**), and stores the results in memory (**LOGON**).

Recovering the logged data is even easier. For example, sending the single-character command

```
U
```

(the **unload** command) to the *DT80* returns time-stamped data to your computer in a format ready to be imported into the preferred program. The connection between the *DT80* and the host computer could be via Ethernet, USB, RS232 or modem.

Alternatively, you could insert a USB "memory stick", and select the **COPYDATA** option using the built-in keypad and LCD display.

The *DT80* can be programmed to carry out extremely powerful tasks. To do this, it will be necessary to be familiar with more of the set of *dataTaker* commands. Explore the features that are available.

The *DT80* Product Family

There are currently two members of the *DT80* product family:

- The **DT80** is a full-featured data logger,
- The **DT81** is a lower cost variant of the DT80.

Both models operate in a very similar way. The main differences are as follows:

Feature	DT80	DT81
Analog input channels	5	1
Digital I/O channels (open-drain outputs)	4	3
Digital I/O channels (logic outputs / SDI-12)	4	1
Serial sensor channel	yes	-
Phase encoder inputs	2	1
LCD display & keypad	yes	-
Status LEDs	3	4

In this manual, the term *DT80* is used to refer to both products (DT80 and DT81). If a feature or behaviour is specific to a particular model, this will be made clear in the text.

DT80-Friendly Software

Programming and Configuration

There are three main ways to set up and program the *DT80*.

- **DeLogger** is a Windows based application for programming and monitoring any dataTaker data logger, including the *DT80*. It provides a totally graphical interface, which means that knowledge of the *dataTaker* programming language is not required. Channels and schedules are defined simply by clicking on icons and making selections from menus and dialog boxes; DeLogger will then generate the required *DT80* program and load it onto the logger.
- Alternatively, commands entered interactively and then sent to the *DT80* via one of its comms ports. This allows full access to the *DT80*'s capabilities. **DeTransfer** is the best tool for the job here. It has separate send and receive windows, a macro facility, and many other useful features. A standard terminal program (eg Hyperterminal) can also be used.
- Finally, you can develop a *DT80* program off-line (eg. using a text editor), then transfer it to the *DT80* using a USB memory device or send it as a file using DeTransfer. **DeLoad** is a Windows based application which allows a pre-written program to be transferred to the logger using a simple "drag and drop" operation.

Viewing Data and Status

Once the *DT80* has been set up, there are a number of options for retrieving data and monitoring status:

- The *DT80*'s inbuilt **web interface** provides a convenient way to access current data values and status information from any web browser (no additional software required). This can be customised if required to provide an application-specific user interface.
- **DeLogger** can read real-time or logged data from the *DT80*, then display it in dynamic table, chart and mimic (meter) views, load data into a fully-featured spreadsheet, and replay saved data.
- **DeLogger Pro** is the big brother of DeLogger. It has the added features of modem support, a database data storage option, the ability to connect to more than one data site at a time, enhanced mimic screens, additional spreadsheet and graphical analysis tools, and e-mail and web publishing capabilities.
- **DeTransfer** can be used to view real-time and logged data in text format.
- **DeView** is a lightweight application that is used to display the data from dataTaker replay files (.dlr) and binary data files (.dbd). It can display data in a time series, cross tabulated grid and a trend chart. It can also export data in 'csv' format for use in other programs such as spreadsheets. DeView is a good choice when you need a simple application for viewing and exporting data. It also handles very large data files.
- **DeLoad** provides an easy way of collecting logged data, which can then be saved or sent by email.
- ***dataTaker* ActiveX** is a software component that allows the creation of custom application software for use with *dataTaker* data loggers. It provides over 60 functions for automatically finding and creating connections to data loggers, sending commands and unloading data.
- ***dataTaker* Instrument driver for LabVIEW™** is a set of drivers and documentation which allows *dataTaker* data loggers to be incorporated in a LabVIEW environment. LabVIEW is National Instruments' industry-leading graphical software development environment for measurement and automation applications.

All software (except **DeLogger Pro**) is provided on the CD supplied with your *DT80*, and updates are available from the dataTaker website, **www.datataker.com**.

About This Manual

This manual is intended for all users of the *DT80*. It describes:

- how to connect sensors and other devices to the *DT80*'s input and output channels.
- how to program the *DT80* to collect and return data as required.
- how to manage the data that the *DT80* collects.

The main focus of this manual will be on directly programming the *DT80* using its command language. However, most of the concepts discussed here also apply when building programs using tools such as DeLogger.

A Tour of the *DT80*'s Interfaces

The *DT80*'s interfaces with the outside world are grouped into three main areas:

User Interface

On the top panel of the *DT80* you will find controls which allow the user to interact with the unit during operation – without requiring a host computer:

- A 2-line LCD display shows status messages, measured values, and a menu of pre-defined functions (DT80 only)
- Six keypad buttons allow the user to navigate between the various displayed options (DT80 only)
- Three status LEDs are provided – the blue **Sample** LED flashes each time a measurement is taken, the green **Disk** LED indicates internal flash disk activity, and the red **Attn** LED indicates various warning conditions.

A fourth indicator is present on the DT81 – the green **Power** LED flashes at 3 second intervals while the DT81 is powered and not in low power "sleep" mode. The duty cycle of the flash indicates whether the DT81 is externally powered (long flashes) or running from its internal battery (short flashes).

- A USB socket allows connection of a USB memory device, which provides a convenient way to retrieve data from the *DT80* (or load a program onto it)

Sensor Interface

On the sloping front panel of the *DT80* there are two rows of terminal blocks – digital channels on the left, analog channels on the right. The green terminal blocks can be quickly unplugged from the *DT80* without unscrewing the sensor cabling.

This interface includes:

- 8 digital input/output/counter channels (**1D** – **8D**), 4 of which are SDI-12 compatible (DT81: 4 channels, one of which is SDI-12 compatible)
- an input to wake the *DT80* from low power "sleep" mode (**WK**)
- 4 counter inputs (or two phase encoder inputs) (**1C** – **4C**) (DT81: one phase encoder input)
- a pair of voltage free relay contact outputs (**RELAY A** and **B**)
- an RS232/422/485 compatible serial port (**Tx**, **Rx**, **RTS** and **CTS**) (not present on DT81)
- 5 analog input channels (**1** – **5**) (**DT81:** one analog input channel)
- an external excitation input (**EXT ***)

(Note that early production DT80 models only had 4 analog inputs.)

Communications/Power Interface

On the left side panel you have a variety of connectivity options:

- 10-Base-T Ethernet for connection to a host computer or local area network
- USB for high speed connection to a host computer
- RS232 for connection to host computer or modem
- two alternative DC power connectors – a standard plug-pack socket (DC jack) and a 4-pin terminal block

For more details, see *Communications* (P113)

Getting Started

Power

Powering the DT80 (P144) discusses the ways to provide power to the *DT80*. The simplest option is to plug in the supplied AC adaptor.

The *DT80* includes an internal 6V lead-acid battery which can power the logger if the main external supply is interrupted.

Important The *DT80* is shipped with its main internal battery disconnected. We recommend the battery is connected as soon as practical so that it can charge from the mains adaptor or other external power source. This is achieved by simply plugging the green power connector, see *Powering the DT80* (P144).

Switch On!

When power is connected, you should observe:

- the LCD backlight switches on (DT80), or the green **Power** LED starts flashing (DT81)
- a brief clicking sound as the unit performs an initial self-calibration
- `DT80 restarted` / `Power loss` is displayed on the DT80's LCD
- the front panel LEDs flash a few times then the red **Attn** LED continues to flash.

The *DT80* is warning you that its power has been interrupted. Press any of the front panel keys (or send the command **CATTN**) to clear this indication. The **Attn** LED should stop flashing and the display should now read: `DT80 V6.02` / `No current job`. This indicates that:

- the version of *DT80* firmware in use is "6.02" (this number may vary), and
- no user program (or "job") has been loaded

The *DT80* is now idle and waiting for instructions.

Connecting to a Host Computer

In order to program the *DT80*, it is generally necessary to connect it to a "host" computer. The easiest option here is to use the supplied USB cable. Other options are to use a "null-modem" (cross-over) RS232 cable, or to connect the logger to an Ethernet network. See *Communications* (P113) for more details of the different communications options.

Very briefly, connecting the *DT80* via USB involves the following steps:

1. Install the required dataTaker software (DeLogger and/or DeTransfer) on the host PC.

2. Connect the USB cable between the *DT80* and the PC.
3. The Windows "New Hardware Found" wizard will then run automatically (if required) to install the necessary drivers.
4. Launch DeTransfer (or DeLogger)
5. In DeTransfer (or DeLogger), create a "connection"; this involves selecting the port to use when communicating with the *DT80*. A "virtual COM port" (e.g. COM5) will have been assigned by the USB driver.
6. Press the "Connect" button in DeTransfer (or DeLogger).

The above is only an brief overview. See *USB Communications* (P114) for detailed, step by step instructions.

The remainder of this manual will assume you have successfully established a connection between the host PC and the *DT80*.

Sending Commands

The *DT80* is programmed by sending it textual **commands**. Commands are executed by the *DT80* only after it receives a carriage-return character (↵).

Commands are not case-sensitive; that is, they may be entered using either uppercase or lowercase characters.

In this manual all commands are shown in **UPPERCASE**. Responses from the *DT80* are shown `like this`.

After receiving a command, the *DT80* will normally **echo** the command, after converting it to uppercase. Note that the *DT80* does not echo each character as it is received.

After a command has been processed, the *DT80* will normally indicate that it is ready for the next one by transmitting a **prompt** string:

```
DT80>
```

(Command echo and the prompt string can be turned off if required using the **/e** switch command.)

The maximum length of a command is 255 characters.

The general categories of commands are:

- **channel definitions** (P27) (e.g. **2TK("Kiln temp",FF4)**) – these define what measurements are to be taken, how they are to be acquired and how the measured values are to be presented.
- **schedule definitions** (P42) (e.g. **RA(DATA:2MB)10S**) – these define **when** a set of measurements are to be taken and where the results are to be stored
- **job management commands** (P56) (e.g. **BEGIN**, **END**, **SHOWPROG**) – these allow a set of schedule and channel definitions to be grouped into a single program, or "job", which can then be treated as a unit.
- **data management commands** (P84) (e.g. **U** (unload), **COPYDATA**, **DELALARMS**) – these allow logged data points and alarms to be retrieved, displayed or deleted.
- **configuration commands** (P129) (e.g. **PROFILE**, **P***n* (parameter), **/***char* (switch)) – these allow various aspects of the *DT80*'s operation to be adjusted to suit particular requirements.

Jobs (sets of commands) are stored in the *DT80*'s internal file system along with the data they generate. Different jobs can be loaded under manual or program control. In addition, the *DT80* can automatically run a particular job every time it is reset or powered up. See *Startup Job* (P57).

Localisation

Many different aspects of the *DT80*'s operation can be customised. Some of these relate to the locale in which it is operating – in particular the local mains frequency and date/time format. For best performance it is recommended that these settings (especially the mains frequency) be configured and saved before taking any serious measurements.

The *DT80* parameter **P11** specifies the local mains frequency, in Hz (default 50Hz). When taking an analog measurement, the *DT80* integrates over one or more complete mains periods, in order to minimise any mains-related noise pickup.

The parameter **P31** specifies the date format: **1** for European (DD/MM/YYYY), **2** for North American (MM/DD/YYYY) and **3** for ISO (YYYY/MM/DD) (default is European format).

These (and any other settings) can be applied in a "set and forget" fashion by entering them into the *DT80*'s **startup profile**. For example, the following commands will set up the *DT80* for North American mains frequency and date format:

```
PROFILE "PARAMETERS" "P11"="60"
PROFILE "PARAMETERS" "P31"="2"
SINGLEPUSH
```

(The **SINGLEPUSH** command resets the *DT80*, which is necessary in order to apply profile settings.) For explanations of parameters and profiles see *Configuration* (P129).

Ways of Using the *DT80*

The *DT80* can be deployed in many ways depending on factors such as location, data volume and power availability:

- on-line to a host computer – data is returned in real-time as it is acquired
- periodic downloading to an on-line host

- periodic downloading to a portable computer
- periodic downloading by modem to a host computer, initiated by either the computer or the *DT80*
- data recovery (and programming) using removable USB memory devices

The method of deployment influences the fine tuning of the *DT80*'s programming. As a general rule, it is better to recover data as often as reasonably possible so that sensor failures, program faults and so on are detected earlier.

Fundamental Inputs and Ranges

The *DT80* can directly measure the following **fundamental inputs:**

- voltage
- current
- resistance
- frequency
- digital input state
- pulse count
- phase encoder position

Many other quantities can be measured by connecting appropriate **sensors** which convert a physical quantity into something that the *DT80* can measure. The *DT80* directly supports:

- 4-20mA current loop sensors (0 to 100%)
- temperature sensors (thermocouples, RTDs, thermistors, IC sensors)
- bridges and strain gauges

This list can be extended by means of user specified scaling calculations.

Fundamental Input Ranges

The following table lists the available measurement ranges and resolutions for the fundamental input types.

Input Type	Range	Resolution
DC Voltage	±30 mV	0.25 µV
	±300 mV	2.5 µV
	±3000 mV	25 µV
	±30 V	250 µV
DC Current Internal Shunts(100Ω)	±0.3 mA	2.5 nA
	±3 mA	25 nA
	±30 mA	250 nA
External Shunts (typically 20~200Ω)	any range	depends on shunt
Resistance	100 Ω	1.5 mΩ
	1000 Ω	15 mΩ
	10,000 Ω	150 mΩ
Frequency	0.1 to 20,000 Hz	0.0002%
Digital Bit	0 or 1	1
Counter	-2,147,483,648 to 2,147,483,647 counts	1 count
Phase Encoder	-2,147,483,648 to 2,147,483,647 counts	1 count

Accuracy of the DT80

Maximum measurement error is given by:

*error = (reading * Basic Accuracy) + (FullScale Reading * 0.01%)*

where *Basic Accuracy* is as specified in the following table:

	5ºC to 40ºC	-45ºC to 70ºC
DC voltage measurement	±0.1%	±0.35%
DC current measurement	±0.15%	±0.45%
DC resistance measurement	±0.1%	±0.35%
Frequency measurement	±0.1%	±0.25%

Derived Measurement Ranges

The following table indicates typical measurement ranges and resolutions for derived measurements using external sensors:

Input Type	Range	Resolution
4-20mA Loop	0 to 100%	0.01%
Temperature	–250.0 to 1800 °C	depends on sensor
Strain Gauges and Bridges	$\pm 10^4$ ppm	1 ppm
	$\pm 10^5$ ppm	10 ppm
	$\pm 10^6$ ppm	100 ppm
Analog State	0 or 1	1

Analog Channels — Introduction

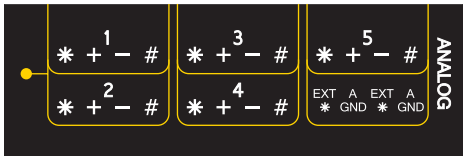


Figure 2: DT80 analog terminals

Input Terminals

The DT80 provides five analog input channels, numbered **1** to **5**. Depending on the wiring configuration used, these allow between 5 and 15 separate voltages to be measured. The DT81 has one analog input channel, allowing 1-3 separate voltages to be measured.

Each analog input channel on a *DT80* is a 4-wire connection (see Figure 3) that allows voltage, current, resistance and frequency to be measured. These are the fundamental signals output by most sensors. It is not necessary to use all four terminals on each channel— two are often adequate.

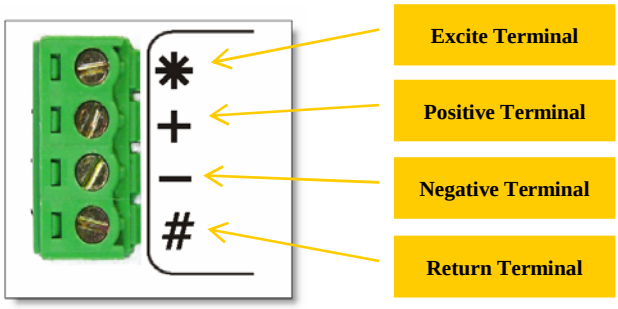


Figure 3: Analog input channel terminal labels

The exact function of each terminal varies depending on how the channel is programmed. In general terms:

- The * ("Excite") terminal can be a voltage input, or it can provide sensor excitation current (for example, for resistance measurement).
- The **+** ("Plus") and **–** ("Minus") terminals are voltage inputs
- The **#** ("Return") terminal is normally used as a common or return terminal. It can also be used as a current input, using the *DT80*'s internal shunt resistor.

Multiplexers

The *DT80*'s analog input channels are **multiplexed**. The required input terminals are first connected to the input of the *DT80*'s instrumentation amplifier and analog to digital converter, then a measurement is taken. The next channel to be sampled is then switched through to the amplifier and ADC, and so on.

Channel definition commands in the *DT80* program determine which terminals are used for a particular measurement. For example, the channel definition **1+V** measures the voltage between the **+** and **#** terminals on channel 1.

Gain Ranges and Attenuators

The *DT80*'s instrumentation amplifier has three switchable gain settings. These give three basic voltage measurement ranges (3V, 300mV and 30mV full scale)

The *DT80*'s default is for its instrumentation amplifier to automatically change gain range to suit the input signal applied to it by the multiplexers.

If the amplitude of your input signals are known, then the gain can be set manually. Do this by applying the **GL***x* (gain lock) channel option, which disables autoranging for that channel and sets the gain to a fixed range.

The analog inputs also include switchable 10:1 **attenuators**, which effectively provide a fourth range (30V).

Note however, that the autoranging process does not affect the attenuator setting. Each channel definition command specifies (either implicitly or explicitly) whether the attenuators should be on or off.

Warning Knowledge of the output signal type and magnitude for each sensor is essential. Make sure that the input signal to the *DT80* does not exceed the input voltage rating. As a general rule, the voltage on any analog input terminal should be within ±30V or ±3V (depending on whether the channel's attenuators are on or off) relative to the **AGND** terminal.

Analog Input Configurations

The basic quantity that the *DT80* measures is voltage. Voltages can be measured using two different input configurations:

- **independent** analog inputs
- **shared-terminal** analog inputs

Independent Analog Inputs

Sensors and signals connected using the independent configuration are often simply called "inputs" (sometimes also known as "basic", "default", "unshared", "differential" or "double-ended" inputs).

An independent input is one that connects to its own terminals and does not share any of those terminals with any other inputs. For example, in *Figure 4*, sensor A is connected to channel 1's + and – terminals, and sensor B is connected to the other two terminals of the channel. In other words, each sensor's terminals are independent of the other's — no terminal is used by both sensors.

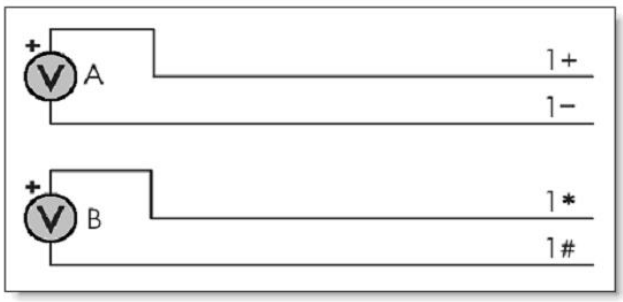


Figure 4 Wiring one or two independent inputs to a single channel (voltage inputs used as example)

For an independent input, the signal voltage is measured between a pair of terminals and neither terminal is necessarily at ground potential.

Note that each analog input channel can support **two** independent voltage inputs. In the above example, the channel definition **1V** will read sensor A while **1*V** will read sensor B. The channel definition syntax is fully described in *Channels* (P27)

Shared-Terminal Analog Inputs

Sometimes called "single-ended" inputs, a shared-terminal input is one that shares one or more of its terminals with another input. For example, in *Figure 5*(P19), the three sensors share channel 1's # terminal. Each of the three inputs is a shared-terminal input.

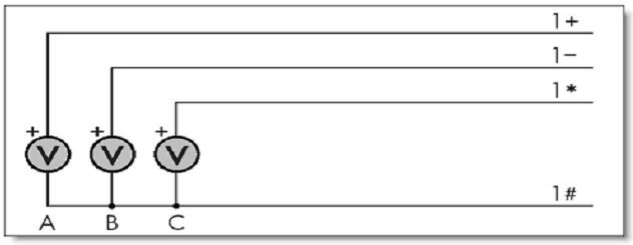


Figure 5 Shared-terminal voltage inputs sharing a channel's # terminal (voltage inputs used as example)

In a shared-terminal configuration, a sensor's "return" or "negative" wire is usually connected to the channel's **#** terminal. The remaining sensor wire (the "positive" or "signal") is connected to any of the channel's other three terminals.

For shared-terminal inputs, the channel number is given a suffix indicating the terminal to which the positive wire is connected. For example, a shared-terminal **V**oltage input applied to channel **1** between the **+** and **#** terminals (*Figure 5* P19)) is recognized by the channel definition **1+V**.

Which Analog Input Configuration Should I Use?

- **Shared input (single-ended)** wiring uses the **#** terminal as a common sense point between multiple sensors. Each of the *, + and - inputs are measured with respect to the **#** input. The main advantage of shared inputs is that the number of measurement points per channel is increased – each *DT80* analog channel can measure three separate voltages.
- **Unshared (differential)** inputs do not share any measurement wires. Unshared inputs allow for easier connection to sensors where there is a **common mode voltage** (an unwanted voltage offset applied to both sensor wires relative to

ground). Because unshared input pairs are totally independent from one other, different sensors can have different common mode voltages without affecting measurement accuracy.

- **Shielded** input cable may be helpful when the signal source has a high output impedance or when noise pickup (especially from power cables) is a problem. Ensure the shield is only connected to the ground at one point (see P24) – usually the logger # terminal on the same channel as the sensor.

Important Unshared (differential) inputs can effectively remove the unwanted common mode component from the input signals provided that the maximum input voltage for each terminal is not exceeded (max. ±3V/30V for attenuators off/on, relative to **AGND**).

Sensor Excitation

Many sensors require **excitation** (electrical energy) so that they can provide an output signal. For example, to read the temperature of a thermistor, excitation current is passed through the thermistor to generate a voltage drop that can be measured.

The *DT80* can provide

- Voltage source of 4.5V via 1kΩ. Useful for powering some sensors however the supply is not regulated and consequently liable to drift with temperature
- 200µA Default current source for resistance measurement. Very stable over environmental temperature range.
- 2.5mA Default source for RTD and bridge measurement. Very stable over environmental temperature range.
- User supplied external excitation **Ext *** terminal. The user can provide an external excitation which is appropriate to the sensor being used.

See the Excitation category in the *Table 3: DT80 Channel Options* (P41) table.

Digital Channels — Introduction

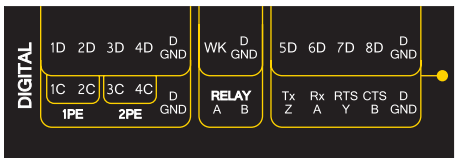


Figure 6: DT80 digital terminals

The *DT80* provides:

- 4 bidirectional digital I/O channels (**1D**-**4D**) with open drain output driver and pull-up resistor (**DT81:** 3 channels, **1D**-**3D**)
- 4 bidirectional digital I/O channels (**5D**-**8D**) with tri-stateable output driver and weak pull-down resistor. These channels may also be used for controlling intelligent sensors using the SDI-12 protocol (Serial Digital Interface – 1200 baud) (**DT81:** 1 channel, **4D**)
- 1 voltage free latching relay contact output (**RELAY**)
- 1 LED output (**Attn**)
- 4 hardware counter inputs (**1C**-**4C**) which can be used as independent counter channels or as two quadrature (phase encoder) inputs (**DT81:** one phase encoder input, shared with inputs **3C** and **4C**)

As with analog channels, **channel definition commands** are used to specify which digital inputs are to be measured and/or what digital output states are to be set. For example, the command **1DS** will read the digital state (0 or 1) on channel **1D**, while **3DSO=0** will set channel **3D** low.

A transition on a digital channel can be used to trigger a schedule. This allows a series of measurements to be made (or commands executed) in response to an incoming digital pulse.

The *DT80* can count the number of pulses received on any digital input. The four dedicated counter inputs provide additional capabilities:

- a higher maximum count rate
- the ability to keep counting even if the logger is in low-power "sleep" mode
- optional low-level (5mV) input threshold levels
- optional decoding of phase-encoded input signals

For more details, see *Digital Channels* (P155)

Serial Channel – Introduction

The *DT80*'s serial channel (not available on the DT81) allows a wide variety of sensors and devices to be controlled and polled. The serial channel:

- supports RS232, RS422 and RS485 signal levels
- supports point-to-point or multi-drop operation

- features programmable output (poll) strings and a variety of options for parsing returned data
- can trigger execution of a schedule in response to received data

For more details, see *Serial Channel* (P166)

Programming the *DT80*

Typical Workflow

When creating a program to send to the *DT80*, typically the work will follow this order:

Specify Channel Types

The input channels are very versatile, but the *DT80* does not automatically know what type of sensor is connected — it must be informed. A channel is defined by a **channel type**, which determines how the *DT80*'s multiplexer is set up and how the readings are to be processed. There are more than thirty different channel types (see *Table 1: DT80 Channel Types* (P32)).

A particular physical channel can be read using different channel types. For example, a thermocouple can be read as a thermocouple or as a voltage. The command

```
1TK 1V
```

returns both a temperature and a voltage based on two readings of the same sensor.

The first task is therefore to select the most appropriate **channel type** for each sensor from the table of available types (P32). The Wiring Configuration column shows appropriate wiring configurations; connect the sensors accordingly.

Add Channel Options

Then, use **channel options** to modify the channel function as required. In a channel definition these are listed in round brackets immediately after the channel type. The *Channel Options* (P35) table describes the channel options.

The sensor can now be tested. For example, entering:

```
2PT385(4W)
```

will measure a platinum resistance temperature sensor (**PT385**) connected as a 4-wire resistance (**4W** channel option) on channel **2**and return the temperature in degrees C.

Define Measurement Schedule(s)

A **schedule** defines when a set of channels should be measured. It consists of a list of channel definitions preceded by a scan trigger specification — see *Figure 8*(P42).

As a general rule when creating schedules, don't instruct the *DT80* to read channels more frequently than is really necessary. For example, temperatures generally change slowly so rapid reading does not provide extra useful information.

Up to eleven different schedules can be declared (**A** to **K**), each with a different trigger based on a time interval or a digital input event. The schedule's trigger can be changed at any time, either manually or under program control.

A list of channels without a trigger specification can be entered at any time. These are scanned immediately, without affecting other schedules that may be operating. For more information, see (P49)

Important Whilst a schedule's **trigger** can be changed at any time, its **channel list** cannot be altered without re-entering all schedules. In fact, all schedules must be entered at the same time, either all on one line or between **BEGIN** and **END** keywords (see *Working with Schedules* (P51)).

Jobs

A *DT80* **job** is a logical "hold-all" for a group of schedule definitions and other commands. The command **BEGIN** signifies the start of a job, and the command **END** signifies the end of the job. Once a job has been fully entered, the *DT80* will activate all schedules defined therein.

The *DT80* can store more than one job (each with its own separate logged data and alarms), but only one can be the current/active job. See *Jobs* (P54) for more details.

Scaling and Calculations

The *DT80* can scale the channel input data to engineering units by applying intrinsic functions, spans or polynomials. Arithmetic expressions provide cross-channel and other calculations. Various statistical functions, including averaging and histogram channel options, can be applied. See *Scaling* (P80).

Reducing Data

In many instances the volume of the data recorded can be reduced by taking averages, maximums, minimums, standard deviations, histograms or integrals. See *Statistical Channel Options* (P65).

Alarms and Conditional Execution

The *DT80*'s alarm facility is flexible and powerful. Alarms are used to warn of certain conditions (eg. setpoint exceeded) and to control the *DT80*'s operation. Alarms can

- control *DT80* digital state outputs
- initiate execution of *dataTaker* commands
- trigger the sending of messages to the host computer.
- set variables

Executing *DT80* commands from an alarm can be particularly useful in modifying the *DT80*'s programming in response to changes in input(s). See *Alarms* (P71).

Data Logging

The *DT80* stores measurements in its internal data store and in removable USB memory device.

Logging begins only after you issue the **LOGON** command. Time and date stamping is automatic.

By default, the *DT80* overwrites the oldest data with new data once the memory is full. If you prefer to have the logger stop logging once the memory is full then you need to set the no-overwrite schedule option (**NOV**) (P43).

For more details see *Logging and Retrieving Data* (P80)

Selective Logging

To selectively log channels and schedules:

- For channels, use the **NL** (no log) channel option
- For schedules, use the **LOGON***x* & **LOGOFF***x* commands

See *Enabling and Disabling Data Logging* (P80).

Retrieving Data

The *DT80* can do two things with the data it measures:

- Return it immediately to the host computer, where it can be seen arriving on-screen. This monitoring function is data return in **real time**.
- Store it in its internal memory and/or an inserted USB memory device ready for retrieval (unload) to the host computer at a later time. This is data **logging**.

The *dataTaker DT80* can carry out these functions separately, or at the same time.

Retrieving Real-Time Data

The *DT80*'s default is to return data to a connected host computer instantaneously — that is, as it is measured. (To override this send the **/r** switch to the data logger (P132)). Store this real-time data as a file on the computer if required.

Real-time data can be returned to the host in either free-format mode (the *DT80*'s default) or fixed-format mode. (P23)

Retrieving Logged Data

Data stored in a *DT80*'s internal memory or USB memory device can be retrieved (or **unloaded**) by means of the Host RS-232 port, the Ethernet port, or the USB port. Data can be retrieved for an individual schedule or all schedules, or for all jobs or an individual job. Here are a few useful commands when retrieving logged data:

Command	Function
U	begins to unload stored data
A	begins to unload stored alarms
Q	terminates unload

See *Retrieving Logged Data* (P84) for more details.

USB memory devices

The *DT80*'s USB port supports USB memory devices, which can be used

- as a medium for transferring logged data from the internal memory of a *DT80* to a computer (see *Archiving Logged Data* (P86))
- as removable data storage. See *Logging Data* (P80)
- to load a job into a *DT80.* See *ONINSERT Job* (P57).
- to upgrade a *DT80*'s firmware (contact dataTaker for details)

Data is stored on the USB memory device is in a Windows-compatible file structure – see *The DT80 File System* (P89).

Format of Returned Data

The *DT80* can:

- return data to a host computer as it is measured (**real-time** data), and/or
- store data in memory to be retrieved at a later date (**logged** data)

You can control whether data is returned or logged on a per channel, per schedule or global basis.

Real-time data

The *DT80* can be configured to return real-time data in one of two possible formats:

- free format mode
- fixed format mode (also known as "host mode", or "formatted mode")

The **/h** switch command selects free format mode (which is the default); **/H** selects fixed format mode.

Free Format Mode /h

In free format mode, data is returned as human-readable ASCII text. Various settings are available to control how the data is presented. By default, each channel is printed on a separate line, prefixed by its name (either a standard *DT80* channel name e.g. "3TK", or a user-specified name e.g. "Inlet temp") and followed by appropriate units.

Thus the following program:

```
RA30S 1V("Pressure~kPa") 2TK 5DS("Valve state")
```

would result in text similar to the following text being sent to the active communications port:

```
Pressure 102.3 kPa
2TK 98.0 degC
Valve state 1 State

Pressure 107.3 kPa
2TK 98.2 degC
Valve state 1 State
```

and so on.

By applying various formatting settings you can get different results. One possible example would be:

```
/n/c/u/T P33=10 RA30S 1V("Pressure~kPa",FF2) 2TK(FF2) 5DS("Valve state")
```

which would format the data thus:

```
12:46:00.029     102.32      97.98          1
12:46:30.017     107.34      98.22          1
```

In this example, **/n/c/u** are **switch commands** *(P132)* that have been used to switch off output of channel numbers, channel names and units. The **/T** switch causes each data record to be prefixed by a timestamp. **P33=10** is a **parameter setting** *(P129)* that sets each data value to a fixed width (10 characters). Finally, the **FF2 channel option** *(P41)* specifies that the channel value is to be rounded to 2 decimal places.

Fixed Format Mode /H

Fixed format mode is designed for use with *dataTaker* host software. Data is still returned in ASCII form, but the record format is **fixed** to allow it to be easily parsed by a computer. If **/H** is specified then both of the above examples will return data as:

```
D,081044,"JOB1",2005/03/29,12:46:00,0.0293681,1;A,0,102.322,97.9799,1;0070;065F
D,081044,"JOB1",2005/03/29,12:46:30,0.0170320,1;A,0,107.341,98.2200,1;0070;3BEB
```

In fixed format mode:

- all formatting commands (e.g. **FF2**, **/n**, channel names) are ignored – fixed settings are used
- all records are prefixed by a **header**, which specifies that this is a data record (D), from *DT80* serial number 081044, running a job called "JOB1". This is followed a timestamp (date, time, and sub-second time). The 1 indicates that this is real-time data, the A identifies the schedule, and the 0 is the index within the schedule of the first data value.
- floating point data values are always specified to 6 significant digits
- each record includes an error-detection code (CRC) on the end. This allows host software to reject corrupted records.

Data records such as the above are only one of several types of fixed format message. A comprehensive description of all fixed format message types is beyond the scope of this manual.

Logged Data

Logged data can be returned in two different formats:

- **native** format
- **fixed format** records

Native Format

When the *DT80* logs data to its internal memory, it stores it in fixed size data files, one for each schedule. These files have a **.DBD** file extension, e.g. **DATA_A.DBD**.

One way of getting data out of a *DT80*, therefore, is to transfer relevant **.DBD** files to the host computer. These files can then be opened using tools such as *DeView,* which provides plotting facilities and can export the data in a form that can be loaded into spreadsheets.

Native format *DT80* data files can be transferred to the host by:

- copying to a USB memory device. The simplest way to do this is to send the **COPYDATA** command, or select the **Copy Logged Data** option from the LCD function menu. This will take a "snapshot" of the stored data from all schedules in the current job and copy it to appropriate directories on the USB memory device.
- using FTP (File Transfer Protocol). This would involve taking a snapshot of the data using the **ARCHIVE** command, then using an FTP client program to copy the snapshot files.

The "snapshot" or "archive" files created by **COPYDATA** or **ARCHIVE** are compressed versions of the "live" data files, and have names like **2005-04-01T12-42-09.DBD**.

Fixed Format

The other way to retrieve logged data is to send an **unload** command. This causes the *DT80* to read the data file and output the data in the form of fixed format records (as described above). For example the command **U** will unload all data from all schedules in the current job. For more details, see *Unloading Data and Alarms* *(P84)*.

Guidelines for Successful Data Gathering

The Procedure

Data acquisition and data logging are orderly processes and should be undertaken in a systematic way. In order to obtain effective information efficiently, do the following:

- Identify the quantities to be measured.
- Select the sensors, considering measurement range, accuracy, stability, ruggedness and cost.
- Select the wiring configuration. For example, resistive sensors can be connected in 2, 3 or 4 wire configuration.
- Determine sensor output scaling, that is, the relationship between sensor output voltage/current/resistance/etc. and the actual quantity. For many sensor types this calculation is performed automatically by the *DT80* – all you need to do is specify the appropriate channel type.
- Determine how data is to be processed, for example statistical functions such as max/min or histograms may be required.
- Decide on the sample frequency – don't sample faster than you need to.
- Calculate the volume of data to be collected.
- Decide on the method of data recovery and archiving – real-time data return or logging or both? Will logged data be unloaded via a comms port or collected using a USB memory device? How often?
- Consider the power consumption. If power resources are limited, low power sleep mode can be enabled.

Having defined the task, connect sensors and program the *DT80*.

Grounds, Ground Loops and Isolation

Experience has shown that ground loops (sometimes called "earth loops") are the most common cause of measurement difficulties. Excessive electrical noise, unexpected offset voltages and erratic behaviour can all be caused by one or more ground loops in a measurement system.

Grounds are Not Always Ground

Electrical grounds in a measuring system can be an elusive cause of errors.

In the real world, points in a system that one could reasonably consider at ground potential are often at different and fluctuating AC or DC potentials. This is mainly due to earthed neutral returns in power systems, cathodic corrosion protection systems, thermocouple effects in metal structures, lightning strikes and solar storms. Whatever the cause, the result can be loss of measurement integrity.

Ground Loops

If grounds of different potential are connected by cabling used in the measuring system, ground currents flows — this is the infamous **ground loop**. The magnitude of the currents can be from milliamperes to tens of amperes, and in the case of a lightning strike, can be as high as five thousand amperes. Frequently, voltage drops along cables (caused by these current flows) are superimposed on the desired signal voltage.

A ground loop can arise when a measurement system has more than one path to ground. As *Figure 7* shows, this can be caused by

- connecting a sensor to a ground point that has a different potential to the ground of another sensor — a **sensor-to-sensor** ground loop is likely to flow through the return wires of the two sensors
- connecting the *dataTaker* to a ground point that has a different potential to the ground of one or more of the sensors or instruments connected to the *dataTaker* inputs — a **sensor-to-equipment** ground loop
- connecting the *dataTaker* to a ground point that has a different potential to the ground of the host computer — an **equipment-to-computer** ground loop.

In these situations, conduction paths can occur from one ground point to another through the sensor and/or equipment and/or computer, making measurement errors inevitable (particularly if sensor wires are part of the conduction path).

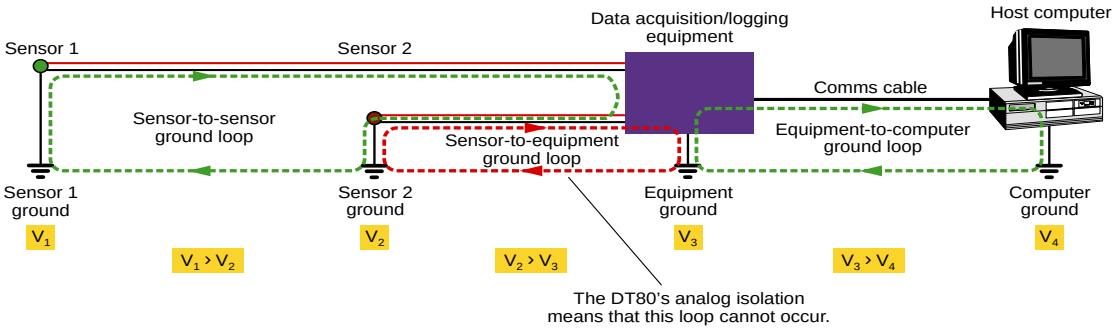


Figure 7: Some of the possible ground-loops in a measurement system

Avoiding Ground Loops

For each type of ground loop, the basic strategy is to break the ground loop.

Isolation

The design of the *DT80* helps eliminate ground loops between the sensors and the *DT80*/computer, because the *DT80* provides over 100V of **isolation** between the **analog section** (which connects to analog sensors) and the **digital section** (which connects to digital devices and computers).

The ground point for the analog section is the **AGND** terminal, which is electrically isolated from the "system" ground (**DGND** terminal).

Furthermore, each analog channel is electrically isolated (to 100V) from other channels.

Connecting Sensors

The following points should be considered:

- There should normally not be any external connection made between **AGND** and **DGND**.
- The sensor's "return" wire should be grounded at one end only (the *DT80* end) of the sensor cable. Normally the return wire would be connected to the **-**, **#** or **AGND** terminal on the *DT80*.
- There should be no connection to ground at the sensor itself, unless that connection is isolated from the sensor return wire. For example, if the sensor has its own power supply then the power supply should be isolated from ground (eg. by using a transformer-isolated mains supply)
- Use an unshared input configuration in preference to a shared configuration.

Noise Pickup

There are two main ways in which noise can be introduced into signal wiring: by capacitive coupling and by magnetic induction. There are different counter-measures for each.

Shield signal wiring to minimize capacitive noise pick-up. Signal wiring that is close to line voltage cable should always be shielded.

Magnetic induction of noise from current-carrying cables or from electrical machines (especially motors and transformers) is a greater problem. Shielded cable is not an effective counter-measure. The only practical measures are to

- avoid magnetic fields
- use close-twisted conductors for the signal wiring.

Shielding in steel pipe can be effective, but is generally not economic or convenient.

Noise Rejection

The *DT80* is designed to reject mains noise. For best noise rejection, set the *DT80*'s parameter 11 to your local mains frequency, 50Hz or 60Hz — see *P11* (P129).

To force the *DT80* to load this parameter setting every time it restarts use the following command

```
PROFILE "PARAMETERS" "P11"="60"  'for 60Hz line frequency
```

Self-Heating of Sensors

Sensors that need excitation power to be read are heated by power dissipation. This issue can be particularly acute with temperature sensors and some sensitive bridges. If self-heating is a problem, consider:

- selecting 250µA excitation (**I** channel option) in preference to 2.5mA excitation (the trade-off is a reduction in the range of resistances that can be measured).
- reducing measurement time, eg. setting **P11=200** will reduce the time spent exciting the sensor from the default 20ms to 5ms (the trade-off is a reduction in mains noise rejection).

Getting Optimal Speed from Your *DT80*

In applications where it is important to sample as rapidly as possible, the following guidelines may assist:

- Switch off data return and logging for channels you are not interested in.
- Set P11 to a higher value, which will reduce the time over which an analog acquisition is integrated. For example, if P11 is changed from 50 to 500 then the sample time will be reduced from 20ms to 2ms. Be aware that this will cause a degradation in the *DT80*'s ability to reject noise.
- Reduce the channel's settling time, using the **MD***n* channel option. The default is 10ms. It is not recommended to reduce this below 5ms because the *DT80*'s relays need about this long to switch before a measurement can commence.
- Disable automatic calibrations using **/k**. Be aware that readings will now be subject to drift with temperature.
- Set **P62=1**, which will leave the *DT80*'s relays set when the schedule completes (normally they are set back to a quiescent state at the end of a schedule). If you are measuring a single voltage input then you will now be able to do repeated samples without changing the relay settings at all, which would then allow you to reduce the settling time to zero (ie. **MD0**)

Part B — Channels

Channel Definitions

A **channel definition** defines a measurement to be taken. It is therefore the fundamental building block that you use when programming the *DT80*.

Channel definitions are normally enclosed in a **schedule definition**. The schedule definition specifies **when** to take the measurements. The channel definitions specify **what** to measure, on **which** terminals and **how** to sample and process the data value.

A sample schedule definition is shown below

```
RA2S  2DS  3R(4W)  2*V(0.1,GL3V,"Speed~km/h",FF0)  9CV(W)=9CV+1
```

This shows four channel definitions which are part of the "A" schedule. Each time this schedule runs (which will be every 2 seconds), four measurements will be taken:

1. The logic state of digital channel 2 will be sampled
2. A resistance connected to analog channel 3 (4-wire connection) will be measured
3. A voltage connected to analog channel 2 (* and # terminals) will be measured and displayed as a speed value
4. An internal general purpose variable will be updated (incremented)

Let us now examine the syntax of a channel definition more closely.

A channel definition consists of up to four components

- the **channel type** is a mnemonic code which tells the *DT80* what sort of quantity is being measured, or what sort of sensor is attached. In the above example the channel types are **DS** (digital state), **R** (resistance), **V** (voltage) and **CV** (channel variable). A channel definition must always include a channel type.
- a **channel number** prefix is required for most channel types. This specifies which channel to measure. In the above example we are measuring digital channel **2**, analog channel **3**, analog channel **2*** and internal variable **#9**
- **channel options** are enclosed in round brackets after the channel type and further specify how the channel is to be measured and processed. In the above example, the **3R**, **2*V** and **9CV** channels have user-specified options, the **2DS** channel does not.
- some channel types are "writable" (eg. internal variables and digital output channels) and therefore allow a value to be assigned using an **expression**. In the above example the **9CV** channel definition contains an expression.

Channel Numbers

A *DT80* **channel number** identifies a particular channel within a certain **class** of channels. The following table lists the various classes of *DT80* channels. As can be seen, each class has its own range of channel numbers.

channel class	terminal labels	channel numbers	applicable channel types
analog	**1** – **5** *(DT80)* **1** *(DT81)*	**1** – **5** *(DT80)* **1** *(DT81)* plus optional ***** **+** **–** **#** modifier	**V HV I L R BGI BGV AS F T***x* **AD5***xx* **CU NI LM***x***35 LM***xx* **PT3***xx* **TMP***xx* **Ys***xx*
digital	**1D – 8D** *(DT80)* **1D – 4D** *(DT81)*	**1** – **8** *(DT80)* **1** – **4** *(DT81)*	**C DB DBO DN DNO DS DSO**
counter	**1C – 4C**	**1** – **4**	**HSC**
		1 – **2** *(DT80)* **1** *(DT81)*	**PE**
relay	**RELAY**	**1**	**RELAY**
LED	**Attn**	**1**	**WARN**
serial		**1** *(DT80 only)*	**SERIAL SSPORT**
SDI-12	**5D – 8D** *(DT80)* **4D** *(DT81)*	**5** – **8** *(DT80)* **4** *(DT81)*	**SDI12**
channel variable	internal	**1** – **800**	**CV**
system variable	internal	**1** – **53**	**SV**
string	internal	**1** – **10**	**$**
timer	internal	**1** – **4**	**ST**
special	internal	no number	**D T DELAY CMRR IBAT R100 REFT VANA VBAT VCC VDD VEXT VLITH VREF VRELAY VSYS VZERO**

The "applicable channel types" column lists the different ways in which a physical channel can be measured. For example, analog channel **1** can be used to measure a voltage (specified by entering **1V**), or a PT385 RTD (**1PT385**) or a frequency (**1F**). All of these **channel types** fall into the analog class, so when we talk about channel 1 we are talking about **analog** channel 1.

Because each channel type is a member of one class only, there is never any confusion about which of the channel 1s is being referred to. **1C** refers to **digital** input 1 because, from the above table, the **C** (counter) channel type is in the digital class. **1HSC**, on the other hand, refers to **counter** input 1 because the **HSC** (high speed counter) channel type is in the counter class.

An **analog** channel number can be suffixed by a **modifier** character, which identifies the pair of terminals between which to measure, as shown in the following table:

Modifier	Measure voltage between
none	**+** and **-**
*****	* and **#**
+	**+** and **#**
–	**-** and **#**
#	**#** and **AGND** (normally only used for current measurements)

Thus the channel ID **3V** defines an independent input between the + and – terminals, while **3*V**, **3+V** and **3-V** define shared-terminal inputs between the *, **+** or **–** terminals (respectively) and the **#** terminal.

Channel Number Sequence

A channel ID that contains two channel numbers separated by two decimal points (for example, **1..3**) defines a **continuous sequence** of channels. If the first channel ID indicates a shared channel, the remaining channels in the sequence follow the order *****, **+**, **–** then **#** (where valid for the channel type). For example

Sequence	is equivalent to
1..4V	**1V 2V 3V 4V**
1+..3-I	**1+I 1-I 1#I 2*I 2+I 2-I 2#I 3*I 3+I 3-I**
1+..3-R(3W)	**1+R(3W) 1-R(3W) 2+R(3W) 2-R(3W) 3+R(3W) 3-R(3W)**

Channel Types

The following table lists all of the channel types supported by the *DT80*. For each channel type, the table shows:

- the **channel type** mnemonic (eg **HV**). Remember that in most cases this will be prefixed by a channel number. Refer to *Channel Numbers* (P28) for details of the allowable range of channel numbers for each channel type.
- whether the channel type is "**writable**" (shown in the Channel Type column). Writable channel types can be assigned a value, eg. **2C=200**.
- the **default channel options** for this channel type. These override the standard default values shown in the channel option table. See also *Channel Options* (P35).
- what the **channel factor** does for this channel type
- the **units** in which data will be returned. By default, the indicated units string will be shown on the display and appended to free format returned data, although it can be overridden if required.
- references to typical **wiring configuration diagrams** for the channel type

Category	Signal\ Sensor Details	Channel Type	Default Channel Options	Channel Factor	Output Units	Wiring config / Comments
Voltage	Voltage input ranges are ±3V, ±300mV & ±30mV	**V**	**(T)** *Note 1*	scaling factor	mV	Wiring: **V1**, **V2** (P178)
	Higher Voltage input ranges are ±30V, ±3V & ±300mV	**HV**	**(A)**	scaling factor	V	
Current	Current	**I**	**(100,T)** *Note 1, 2*	current shunt Ω	mA	Wiring: **C1**, **C2**, **C3**, **C4** (P179) Internal 100Ω shunt is between # and AGND terminals. Other inputs require an external current shunt (typically 100Ω, but higher for small currents)
	4-20mA current loop	**L**	**(100,A)** *Note 2*	current shunt Ω	%	
Resistance	Resistance by 2, 3 or 4-wire methods, 10KΩ maximum.	**R**	**(I,3W)**	offset adjust Ω *Note 4*	Ohm	Wiring: **R1**, **R2**, **R3** (P180)
Bridge See *Bridges* (P151)	3 & 4-wire; quarter, half & full bridge; current excitation	**BGI**	**(350,II, 3W)**	arm resistance Ω	ppm	Wiring: **B3, B4** (P182) External completion required for 1/2 & 1/4 bridges.
	Ratiometric, 4 & 6-wire bridges, voltage excitation .	**BGV**	**(V,4W)**	offset adjust ppm *Note 4*	ppm	Wiring: **B1, B2** (P181) External completion required for 1/2 & 1/4 bridges. Measure reference voltage using another channel with **(BR)** option, otherwise excitation assumed to be 5.0V
Frequency	Frequency measurement	**F**	**(30,T)** *Note 1*	sample period ms	Hz	Wiring: **V1**, **V2** (P178) Threshold is 0V. Use **(2V)** option to set threshold to 2.5V.
Time, Date and System Timers	Time of day	**T** *writable*				Specified in current time/date format
	Day or date	**D** *writable*				
	System timers See *System Timers* (P33).	**1ST** **2ST** **3ST** **4ST** *writable*	**(60)** **(60)** **(24)** **(7)**	range	Counts	Increment every sec (**1ST**), min (**2ST**), hour (**3ST**), day (**4ST**)
Delay	Delays schedule execution for nominated time	**DELAY** *writable*			ms	*Note 6*
System Data	System variable	**SV**				See *System Variables* (P34).

Category	Signal\ Sensor Details	Channel Type	Default Channel Options	Channel Factor	Output Units	Wiring config / Comments
		some are writable				
Variables	Channel variables: general purpose holders for data, calculation results	**CV** *writable*		scaling factor		Use channel options (*Variables* (P40)) to assign data to a CV. Read the CV as for a normal channel.
	Integer variables See *Rainflow Cycle Counting* (P68).	**IV**		scaling factor		
Text	General purpose text for headings, etc. (ten 80-character channels)	**$** *writable*				Assign by sending *n*$="*my text*" See *Text* (P33).
Serial Channel *(DT80 only)*	Transmit to and receive from serial device via RS-232, RS422 & RS485	**SERIAL**	**(***P53***)** ie. parameter P53 specifies default timeout	timeout (sec)	State	Wiring: **S1**, **S2**, **S3** (P187) See *Serial Channel* (P166)
SDI-12	Control SDI-12 sensors using digital channels 5-8	**SDI12**	**(AD0,R1)**			Wiring: **S4** (P187) See *SDI-12 Channels* (P159)
Serial Channel Enable *(DT80 only)*	Enable/disable serial sensor port	**SSPORT** *writable*			State	Power supply for transceiver 0=disabled, 1=enabled Automatically enabled if any serial sensor channels or schedule triggers defined
Temperature	Thermocouples Type B, C, D, E, G, J, K, N, R, S and T See *Thermocouples* (P148).	**TB**, **TC**, **TD**, **TE**, **TG**, **TJ**, **TK**, **TN**, **TR**, **TS**, **TT**	**(T)** *Note 1*	scaling factor	degC *Note 3*	Wiring: **V1**, **V2** (P178)
	Platinum RTDs (α = 0.00385, 0.00392) See *RTDs* (P150)	**PT385** **PT392**	**(100,3W, II)**	0°C resistance Ω	degC *Note 3*	Wiring: **R1**, **R2**, **R3** (P180)
	Nickel RTD (α = 0.005001) See *RTDs* (P150)	**NI**	**(1000,3W, I)**	0°C resistance Ω	degC *Note 3*	
	Copper RTD (α = 0.0039) See *RTDs* (P150)	**CU**	**(100,3W, II)**	0°C resistance Ω	degC *Note 3*	
	Thermistors: Yellow Springs 400XX series See *Thermistors* (P150).	**YS01** **YS02** **YS03** **YS04** **YS05** **YS06** **YS07** **YS16** **YS17**	**(3W,I)**	parallel resistor Ω	degC *Note 3*	
	Semiconductor current source types (Analog Devices)	**AD590** **AD592** **TMP17**	**(100,V,U)** *Note 2*	current shunt Ω	degC *Note 3*	Wiring: **A1** (P184) Calibrate by variation of shunt value channel factor.
	Semiconductor (zener diode) voltage output types (National Semiconductor Corp.)	**LM135** **LM235** **LM335**	**(2,V)**	scaling factor	degC *Note 3*	Wiring: **L3** (P185) Calibrate using scaling factor relative to 0 Kelvin. Default scaling factor is 2 to suit external voltage divider.

Category	Signal\ Sensor Details	Channel Type	Default Channel Options	Channel Factor	Output Units	Wiring config / Comments
	Semiconductor voltage output types (National Semiconductor Corp., Analog Devices)	**LM34** **LM35** **LM45** **LM50** **LM60** **TMP35** **TMP36** **TMP37**	**(V)**	offset adjust degC *Note 4*	degC *Note 3*	Wiring: **L1**, **L2** *(P184)* Offset adjustment is always in degC
Digital See *Digital Channels* *(P155)*.	Digital state input (1 bit)	**DS**			State	Wiring: **D1**, **D2** *(P185)* Result is 0 (low) or 1 (high) Max channel number = 8
	Digital nybble input (4 bits)	**DN**	**(15)**	bit mask *Note 5*	Nybble	Wiring: **D1**, **D2** *(P185)* Result is 0 to 15. Channel number = LSB of nybble. Max channel number = 5
	Digital byte input (8 bits)	**DB**	**(255)**	bit mask *Note 5*	Byte	Wiring: **D1**, **D2** *(P185)* Result is 0 to 255. Channel number = LSB of byte. Max channel number = 1.
	Digital state input on an analog channel	**AS**	**(2500)**	threshold (mV)	State	Wiring: **V1**, **V2** *(P178)* Result is 0 (<threshold) or 1 (>threshold)
	Output on a single digital channel.	**DSO** *writable*		delay (ms) *Note 6*	State	Wiring: **D3**, **D4**, **D5** *(P186)* *n***DSO=0**: output low *n***DSO=1**: output high
	Nybble output on a group of digital channels	**DNO** *writable*	**(15)**	bit mask *Note 5*	Nybble	Wiring: **D3**, **D4**, **D5** *(P186)* Channel number = LSB of byte. Max channel number = 5.
	Byte output on a group of digital channels	**DBO** *writable*	**(255)**	bit mask *Note 5*	Byte	Wiring: **D3**, **D4**, **D5** *(P186)* Channel number = LSB of byte. Max channel number = 1.
	Pulse count on digital input	**C** *writable*		range	Counts	Wiring: **D1**, **D2** *(P185)* Max count rate 30Hz Not active during sleep Counter resets after *range* counts, if set
Counter	High Speed Up Counter	**HSC** *writable*		range	Counts	Wiring: **D1**, **D2** *(P185)* Max count rate 100kHz Active during sleep Counter resets after *range* counts, if set
	Phase Encoder	**PE** *writable*			Counts	Wiring: **D7** *(P185)*
Relay	Relay Output	**RELAY** *writable*		delay (ms) *Note 6*	State	Wiring: **D6** *(P186)* **1RELAY=0**: open **1RELAY=1**: closed Relay is latching type, so it only draws current when changing state.
Attention LED *Figure 66 Digital Input Wiring* *(P186)*	Activate **Attn** LED *(P141)*	**WARN** *writable*		delay (ms) *Note 6*	State	**1WARN=0**: LED off **1WARN=1**: LED on **Attn** LED may also be used by the *DT80* to indicate various warning conditions
Internal Maintenance	Reference temperature of terminal block (the *DT80*'s body temperature)	**REFT**			degC *Note 3*	Used for thermocouple reference junction compensation
	Terminal voltage of internal 6V lead acid	**VBAT**			V	Battery flat if below 5.6V.

Category	Signal\ Sensor Details	Channel Type	Default Channel Options	Channel Factor	Output Units	Wiring config / Comments
	battery					
	Internal Lithium memory-backup battery voltage.	**VLITH**			V	Replace battery if below 2.8V.
	Internal main battery current	**IBAT**			mA	Positive if charging, negative if discharging
	Analog 2.5V voltage source reference	**VREF**			V	
	Analog zero voltage reference	**VZERO**			mV	
	Internal 100 Ohm Shunt	**R100**			Ohms	
	Internal analog 3.8V rail voltage	**VANA**			mV	
	Internal 3.3V rail voltage	**VDD**			mV	
	Internal system supply rail voltage	**VSYS**			mV	
	Internal relay supply voltage	**VRELAY**			mV	
	Raw voltage onto system from external supply	**VEXT**			V	
	Common-mode rejection ratio at maximum gain	**CMRR**			dB	

Table 1: DT80 Channel Types

Notes

1. Input termination is on by default **(T)** for **independent (differential) inputs only**. For shared inputs the # terminal is connected to **AGND** via an internal 100 ohm resistor, so the 1M Ohm termination used for differential measurements is not required
2. If the current shunt value is specified (as the channel factor) then that value is used. Otherwise, if the measurement uses the *DT80*'s internal shunt on the # terminal (eg. **3#I**), then the *DT80* uses the actual calibrated resistance of its shunt. Otherwise, the external shunt is assumed to be 100.0 ohms.
3. Alternatively, parameter P36 can be set to force all temperatures to be returned in degF, degR or K.
4. Offset corrections are subtracted from the measured value.
5. The bitmask specifies which channels are affected by a multi-bit read or write. Channels where the corresponding bitmask bit is zero are not affected. For example **1DNO(3)=0** will set digital outputs **1D** and **2D** low but the state of outputs **3D** and **4D** will be unchanged.
6. The *delay* channel factor can be used in conjunction with the **R** channel option to generate a fixed width pulse output.

   Note: use *delay* carefully as it prevents execution of any other schedules, measurements or outputs during the delay.

Internal Channel Types (in detail)

The *DT80* has its own internal channels, which can be read in exactly the same way as the obvious "external" channels. Use the channel types below.

Time & Date

The *DT80*'s real-time clock/calendar has a resolution of 122µs, based on a 24-hour clock. Time is read in the same way as any channel, but without a channel number. That is, sending

```
T
```

returns

```
Time 11:45:10.213
```

This channel type is writable, so you can set the time by sending:

```
T=12:20:00
```

Time can be in several formats, selected by parameter P39 as follows:

P39=	Format	Example
0 (default)	Hours:minute:seconds.seconds P41 controls the number of sub-second digits between 0 and 6; default is 3 digits	11:45:10.003
1	s.s (decimal seconds) since midnight	42310.003
2	m.m (decimal minutes) since midnight	705.1667
3	h.h (decimal hours) since midnight	11.7528

The current date can also be returned:

```
D
Date 22/05/2005
```

Date can be in several formats, selected by P31 as follows:

P31=	Format		Example
0	Day number	DDDDD	86
1 (default)	European	DD/MM/YYYY	28/03/2002
2	North America	MM/DD/YYYY	03/28/2002
3	ISO	YYYY/MM/DD	2002/03/28

System variable **12SV** returns the combined day.time as decimal days. System variable **15SV** returns the day of the current year.

See also *Setting the DT80's Clock/Calendar* (P135).

Text

Ten 80-character text channels (**1$** – **10$**) are available for labelling, data headings, site identification, *DT80* identification, and so on.

Define the string by sending, for example

```
2$="my text string^M^J"
```

Then, the string is returned (unloaded) whenever *n*$ is included in a channel list.

Text channels can also be set based on data returned via the serial channel. *Control String – Input Actions* (P171)

Control characters may be included in the text string, eg. **^M** for carriage return.

Internal Maintenance

There are several internal maintenance channels, which are read in the same way as normal channels. These allow, for example, the terminal voltage of the *DT80*'s internal batteries to be measured. See the *Internal Maintenance* section of the *DT80* Channel Types table.

System Timers

There are four internal reloading system timers, which are read in the same way as channels. The four timers increment at the following rates, and reset to zero when their range (maximum value) is reached:

System Timer	Channel Type	Increments Every	Default range	Provides
1	**1ST**	1 second	60 (1 minute)	Second of the minute
2	**2ST**	1 minute	60 (1 hour)	Minute of the hour
3	**3ST**	1 hour	24 (1 day)	Hour of the day
4	**4ST**	1 day	7 (1 week)	Day of the week: **0** Sunday **1** Monday **2** Tuesday **3** Wednesday **4** Thursday **5** Friday **6** Saturday

System timers are normally synchronised to the previous midnight or Sunday, and increment at the beginning of each second, minute, hour or day.

If the *DT80*'s date/time is set, the system timer channels will be updated to match the new time.

The range of a system timer can be set using the channel factor. For example, **2ST(15)** will count from 0 to 14, resetting every quarter hour, on the quarter hour.

If the range is set to 0 then the timer will not reset, except at midnight (1-3ST) or midnight Sunday (4ST)

If a system timer is explicitly set to a value, eg. **1ST=12**, then it will no longer necessarily be synchronised to the actual time. In this example, after being set 1ST will count up from 12 to 60, at which point it will reset back to 0 and start counting again. It will always differ from the time-of-day seconds count by a fixed offset.

If a system timer's range is set, it will automatically be resynchronised to the actual time. Therefore **2ST(60)** can be entered at any time to return 2ST to its default behaviour.

If a system timer is set to a value outside its range, it is immediately adjusted so that it is in range. When you enter *n***ST=***x*, you are actually doing *n***ST=***x* mod *range*. Thus **2ST=62** will actually set 2ST to 2.

Examples

Assume the time is now 12:34:56. Then:

```
2ST
2ST 34.0        (34 minutes past the hour – counter resets on the hour)
2ST(0)
2ST 754.0       (754 minutes since midnight – counter resets at midnight only)
2ST(22)
2ST 6.0         (754 mod 22 – counter resets at midnight and every 22 minutes thereafter)
2ST=1
2ST 1.0         (counter is no longer synchronised to midnight)
2ST(22)
2ST 6.0         (setting range value resynchronises timer to current time)
```

2ST will now increment every minute, resetting back to 0 each time it reaches 22. When midnight comes around, it will again be reset to 0.

System Variables

System variables provide various pieces of information about the state of the *DT80* and its current job. All system variables are read-only except where indicated as **writable** in the table below.

System Variable	Function		Writable
1SV	Returns kB free in the internal file system (B:)		
2SV	Returns kB used in the internal file system (B:)		
3SV	Returns kB free in USB memory device (A:) (0 if no memory device inserted)		
4SV	Returns kB free in USB memory device (A:) (0 if no memory device inserted)		
6SV	Returns build number of the *DT80*'s firmware (see also **14SV**)		
7SV	Returns job presence	= 0 if no current job = 1 if a job is loaded	
8SV	Returns current mains frequency setting in Hz (P11)		☑
9SV	Returns USB memory device presence	= 0 if none = 1 if USB memory device inserted	
10SV	Returns ID of the owning schedule	= 0 for RX schedule = 1 for RA schedule = 2 for RB schedule ↓ = 11 for RK schedule = 12 for immediate schedule	
11SV	Returns 0.0	Can be used as thermocouple reference channel for cases where the thermocouple output is already compensated, eg. **RA1S 11SV(TR) 1TT**	
12SV	Returns decimal days since base date (1-Jan-1989) Use formatting for more precision — for example, **12SV(FF4)**		
13SV	Returns *DT80*'s serial number		
14SV	Returns version number (major.minor) of the *DT80*'s firmware (see also **6SV**). Use **14SV(FF2)** to see all decimal places in version number		
15SV	Returns date as day number of the current year (zero =1st of January)		
16SV	Returns Host RS-232 port input handshake line states	Bitmask, 0 to 15 **8** = RI, **4** = DCD, **2** = DSR, **1** = CTS	
17SV	Returns Host RS-232 port output handshake line states	Bitmask, 0 to 3 **2** = DTR, **1** = RTS	☑
18SV	Returns Serial Channel input handshake line states (RS232 mode only)	Bitmask, 0 to 1 **1** = CTS	
19SV	Returns Serial Channel output handshake line states (RS232 mode only)	Bitmask, 0 to 1 **1** = RTS	☑
25SV	Returns current status of a modem connected to the *DT80*'s Host RS-232 port	**0** =no modem connected (direct connection assumed) **1** =modem connected and no call in progress **2** =modem connected and call in progress	
30SV	Number of logged data records for current job, RX schedule		
31SV	Number of logged alarm records for current job, RX schedule		
32SV	Number of logged data records for current job, RA schedule		
33SV	Number of logged alarm records for current job, RA schedule		
34SV	Number of logged data records for current job, RB schedule		

System Variable	Function	Writable
35SV	Number of logged alarm records for current job, RB schedule	
	↓	
52SV	Number of logged data records for current job, RK schedule	
53SV	Number of logged alarm records for current job, RK schedule	

Table 2: DT80 System Variables

Channel Options

Overview

All channel types can be modified in various ways by **channel options**, which define the way in which the input channel is managed when sampled. There are channel options that specify the type of sensor excitation, the termination of the input channel, scaling and linearization of the input signal, the format and destination of channel data, fixed channel gain values, resistance and bridge wiring methods, statistical operations on the channel data, and so on.

As shown below, channel options are placed in round brackets immediately following the channel ID (channel number and type). If multiple channel options are specified then they should be separated by a comma (no spaces).

```
RA2S 1TK 3R(4W) 2*V(0.1,GL3V,"Speed~km/h",FF0)
```

In the above example:

- The first channel, **1TK**, has no channel options specified so it will measure the thermocouple using default settings.
- The second channel (**3R**) includes the **4W** channel option, which specifies that a 4-wire resistance measurement should be taken.
- Finally, the **2*V** channel is in this case used to read a speed sensor which outputs a voltage that is directly proportional to speed (10mV per km/h). The **0.1** channel option is the **channel factor**, which for a voltage channel is interpreted as a simple scaling factor (mV * 0.1 = km/h). The **GL3V** (gain lock) option tells the *DT80* to select the 3V measurement range (rather than auto-ranging). The last two options concern the presentation of the data on the LCD display and in returned real-time data when in free format (**/h**) mode. In particular, they define the channel name and units, and specify that no decimal places be displayed (**FF0**).

  The channel's data will therefore be returned/displayed as:
  ```
  Speed 72 km/h
  ```
  instead of the default:
  ```
  2*V 721.3 mV
  ```

Only certain channel options can be applied to each channel type. If an inappropriate channel option is applied (or an incompatible combination of options), the *DT80* notifies by returning an `E3 - Channel option error` message.

The same channel can be put in the list more than once, with the same or different channel options. The *DT80* treats each occurrence as a separate measurement.

A Special Channel Option — Channel Factor

The *DT80*'s **channel factor** channel option is simply a floating point number. This number is interpreted in different ways depending on the channel type, as indicated by the following table.

Channel Type	Channel Factor's function
V, **HV**, **T***x*, **LM***x***35**, **CV** (voltage/variable)	Scaling factor – for example, **1V(5.5)** means multiply the reading by **5.5**
I, **L**, **AD59x**, **TMP17** (current)	Resistance (ohms) of the external current shunt (the *DT80* uses this value and the voltage it measures across the shunt to calculate current flow)
BGI (current excited bridge)	Bridge arm resistance (ohms)
BGV (voltage excited bridge)	Offset adjustment (ppm)
PT3xx, **NI**, **CU** (RTD)	Resistance of the RTD element at 0°C (ohms)
YS*xx* (thermistor)	Value of connected parallel resistance (ohms)
R (resistance)	Offset adjustment (ohms)
AS (state)	Logic threshold value (mV)
F (frequency)	Sample Period (ms)
C, **HSC**, **ST***x* (counter, timer)	Count modulo value (reset after every n counts)
DN, **DB**, **DNO**, **DBO** (digital multiple)	Bitmask (only channels with 1 in bitmask are read/output)
DSO, **WARN**, **RELAY**, **SSPORT** (digital output)	Delay time (ms)
SERIAL	Timeout (sec)

For example, the three channel definitions in the schedule command

```
RA30S 1V(10.1) 4PT385(200.0) 2DSO(100,R)=0
```

contain channel factor channel options that instruct the *DT80* to do the following:

- scale (multiply) the voltage measured on input channel 1 by **10.1**
- use **200.0**Ω (instead of the default 100.0Ω at 0°C) when calculating the temperature represented by the signal from the RTD on channel 4
- output a **100**ms pulse on digital channel 2.

Multiple Reports

The *DT80* samples each channel in the channel list once every scan. However, by adding additional **channel option sets** (each set enclosed in round brackets) you can generate additional reports. That is, you can report the same data value in different ways.

The first channel option set determines how the channel is sampled, and must include all sampling options required for the channel. These channel options are listed above the **configuration line** in the *Channel Option Table* (P38). Second and subsequent option sets may only contain reporting options (those below the configuration line.

Multiple reports are particularly useful for statistical reports (see *Statistical Report Schedules* (P50)) in that several different statistical operations can be performed on the same data set.

For Example:

```
RA1H 3YS04(II,AV)(MX)(TMX)(MN)(TMN)
```

defines five option sets. The first option set specifies one sampling option (**II** – use 2.5mA excitation) and returns the average temperature value, calculated over the period (1 hour in this case) since the last report scan. The remaining option sets will return the maximum reading over the same interval, the time at which it occurred, the minimum and the time of minimum.

Remember that the first option set can contain options from any part of the channel option table, while subsequent option sets can only contain options from below the Configuration Line.

Mutually Exclusive Options

Options grouped by a shaded rectangle in the Mutual Exclusions column of the table below are **mutually exclusive**. If more than one channel option from a mutual exclusion group is placed in a channel list, only the **last** one specified is recognised.

Order of Application

The *DT80* applies channel options in a specific order, regardless of the order in which they are specified in a channel definition. The channel option table below lists the channel options more or less in the order of application.

In general terms, the ordering is as follows:

1. First, the raw value is sampled, taking note of **sampling options**, ie. those relating to the physical measurement process. These include options in the input termination (**T**, **U**), input attenuator (**A**, **NA**), resistance/bridge wiring (**3W**, **4W**), gain lock (**GL30V**, **GL3V**, **GL300MV**, **GL30MV**) and excitation (**I**, **II**, **V**, **E**, **N**) categories, along with **NSHUNT**, **2V**, **ES***n* and **MD***n*.

   The raw value may then be linearised according to the channel type (e.g. for thermocouples the appropriate polynomial will be applied). The resulting linearised value is then further processed as follows.
2. The **channel factor** is then applied, if specified. For most channel types this is a simple scaling (multiplier) value.
3. A **user specified scaling** option – a span (**S***n*), polynomial (**Y***n*), thermistor scaling (**T***n*) or intrinsic function (**F***n*) – is then applied.
4. The resulting scaled and linearised value may then be manipulated using a **data manipulation** option – difference (**DF**), time difference (**DT**), rate of change (**RC**), reading per time (**RS**) or integrate (**IB**).
5. A **digital manipulation** option for measuring the timing of signal transitions may then be applied (**TRR**, **TRF**, **TFR**, **TFF**, **TOR** or **TOF**)
6. The data value processed up to this point may then be used as a **reference** value for other measurements (**TR**, **BR** or **TZ**)
7. The data value may then be accumulated using one or more **statistical** options (each one in a separate option set). Statistical channel options include **AV**, **SD**, **MX**, **MN**, **TMX**, **TMN**, **DMX**, **DMN**, **IMX**, **IMN**, **INT**, **NUM** and **H** (histogram).
8. Finally, the resultant value after applying the above options (or values if multiple option sets are used) may be stored in a channel variable using **=CV** and *op*=**CV** options. Return, logging and/or display of the data may be disabled using the **NR**, **NL**, **ND** and **W** options, and output formatting can be specified using **FF***n*, **FE***n* and **FM***n* and **"***name*~*units***"**.

Default Channel Options

All channel options have default values. The *DT80* follows a 3-step procedure to determine what options to apply:

1. Start with the basic set of default options specified in the channel option table.
2. If the **channel type** specifies any default options then they are applied, overriding any conflicting basic default options. Default options for each channel type are listed in the channel type table refer *Table 1: DT80 Channel Types* (P32)
3. Finally, if an option is explicitly specified in the channel definition then that setting is used, overriding any default setting. If more than one **mutually exclusive** option is specified then only the **last** one is used, e.g. **1V(AV,MX)** is interpreted as **1V(MX)**. (If you want to output both the average and the maximum then use two separate option sets, i.e. **1V(AV)(MX)**.)

For example, if you specify:

1V(GL3V)

then you are really specifying:

1V(U,NA,N,ES0,MD10,FF1,T,GL3V)

In this case the basic default options are **(U,NA,N,ES0,MD10,FF1)**. The **V** channel type specifies **(T)** as its default option, which overrides the **(U)** option. Then the user specifies **(GL3V)** which overrides the default gain lock option setting (auto).

Channel Option Table

Category	Channel Option	Mutual Exclusions	Function	Range of Option (n)	Comment
Input Termination	**T**		Terminate +, – inputs with 1MΩ to **AGND** terminal		Provides input bias current path to ground to prevent inputs "floating" – particularly when independent (differential) inputs are used.
	U *default*		Unterminate +, – inputs		
Input Attenuators	**A**		Enable ÷10 input attenuators		Attenuators default ON for **HV**, **L** channel types, OFF for other types. Attenuators cannot be used if the *DT80* is supplying excitation.
	NA *default*		Disable input attenuators		
Resistance and Bridge	**3W**		3-wire measurement		Specifies the number of wires run between the *DT80* and the resistance or bridge. More wires generally mean better accuracy.
	4W		4-wire measurement		
Gain lock	(none) *default*		Auto-range over 3 gain ranges		Selects between 3V, 300mV, 30mV ranges if input attenuators disabled Selects between 30V, 3V, 300mV ranges if input attenuators enabled
	GL30V		Lock channel gain for ±30V input signal range		Valid only if input attenuators are enabled
	GL3V		Lock channel gain for ±3V input signal range		
	GL300MV		Lock channel gain for ±300mV input signal range		
	GL30MV		Lock channel gain for ±30mV input signal range		Valid only if input attenuators are disabled
Excitation	**I**		Supply 250µA current excitation on * terminal		Precision current source. Low excitation current minimises self-heating in resistive temperature sensors
	II		Supply 2.5mA current excitation on * terminal		Precision current source. Higher excitation current extends measurement range when measuring large resistances
	V		Supply approx. 4.5V voltage excitation on * terminal		Voltage source is not regulated
	E		Connect external excitation source (**EXT** * terminal**)** to channel's * terminal		
	N *default*		No excitation by *DT80* (assumes externally applied excitation)		Excite terminal (*) may be used as a shared-terminal input channel
Internal Shunt	**NSHUNT**		Disconnect internal 100R shunt between # terminal and **AGND**		Allows # terminal to be used for shared-input voltage measurements
Reference Offset	**2V**		Measure relative to 2.5V rather than 0V		Used with **F** channel type to set threshold to +2.5V (suitable for TTL level input signals) rather than the default of 0V.
Extra Samples	**ES***n* *default = 0*		Perform *n* additional samples and average them	0 to 65535	Can reduce noise. Total measurement time is *n*+1 mains periods.
Measurement Delay	**MD***n* *default = 10*		After selecting channel, delay for *n* ms before starting measurement	0 to 65535	Specifies the settling time required before a sensor can be measured. Default is 10ms.
Reset	**R**		Reset channel after reading		Valid for **C**, **HSC**, **ST** and **CV** channel types, which are reset to 0 after returning their current value. Also valid for digital output channel types (**DSO**, **DNO**, **DBO**) which invert the state of each bit after returning its value.
Channel factor	*f.f*		Linearise/scale the measured value	depends on chan	A scale factor or other parameter specific to channel type (see the channel factor column

				type	in *Table 1: DT80 Channel Types* (P32))
Scaling	**S***n*		Apply span *n*	1 to 50 (poly & span index is shared)	Applies a previously-defined span See *Spans* (P65))
	Y*n*		Apply polynomial *n*		Applies a previously-defined polynomial See *Polynomials* (P59)
	F*n*		Apply intrinsic function *n*.	1 to 7	*n* **Function** 1 1/x 2 √x 3 Ln(x) 4 Log(x) 5 Absolute(x) 6 X^2 7 Grey code to binary conversion (32bit) See *Spans (Sn)* (P58).
	T*n*		Apply thermistor scaling (correction) *n*	1 to 20	Applies a previously-defined thermistor scaling equation See *Thermistor Scaling* (P59) and *Thermistors* (P150)).).
Data Manipulation	**DF**		Difference Δx		Returns the difference (xUnits) between the latest reading and the previous reading
	DT		Time difference Δt		Time difference (seconds) between the latest reading and the previous reading
	RC		Rate of change Δx / Δt		Rate of change (xUnits per second) based on latest and previous readings and their respective times
	RS		Reading / time difference x / Δt		Rate of change (xUnits per second). Useful when the sensor reading is already a difference (e.g. resetting counters)
	IB		"Integrate" (x – Δx / 2) Δt		"Integration" with respect to time (xUnits . seconds) between the latest and the previous readings (area under curve)
Digital Manipulation	**TRR**		Time from rising edge to rising edge		Normally used for digital channels. If used on analog channels then channel factor is interpreted as a threshold value.
	TRF		Time from rising edge to falling edge		
	TFR		Time from falling edge to rising edge		
	TFF		Time from falling edge to falling edge		
	TOR		Time of rising edge		
	TOF		Time of falling edge		
Reference Channel	**TR**		Use this channel's value as thermocouple reference junction temperature		Any non-thermocouple temperature sensor measuring isothermal block temperature. If already compensated use **11SV(TR)** as reference channel (11SV always returns 0.0). TR channel temperature is used for all subsequent thermocouple measurements in this schedule
	TZ		Use this channel's value to correct the *DT80*'s electrical zero		This zero would be measured at the isothermal block TZ channel zero is used for all subsequent thermocouple measurements in this schedule
	BR		Use this channel's value as bridge excitation voltage		BR channel voltage used for all subsequent **BGV** measurements in this schedule
SDI-12	**AD***n*		Sensor address	0 to 9	See *SDI-12 Channels* (P159)
	R*nnn*		Register to read	1 to 999	

	CM	Configure sensor to measure continuously		
Serial Channel	**"***commands***"**	Input and output actions	ASCII text	See *Serial Channel* (P166)
Rainflow Cycle Analysis	**RAINFLOW**			See *Rainflow Cycle Counting* (P68).
CONFIGURATION LINE (see *Multiple Reports* (P36))				
Data tags	**DDE**	Prefix returned channel ID with DDE tag (**&!**)		Only for use with DeTransfer version 3.00 or later. DDE or OLE tags can also be added to a schedule ID, date or time — see *P45* (P130) in the *Table 4: DT80 Parameters* (P131).
	OLE	Prefix returned channel ID with OLE tag (**$!**)		
Statistical See *Statistical Channel Options* (P65).	**AV**	Average of channel readings		These channel options link the channel to the statistical sub-schedule RS. The channel is sampled at times determined by the RS trigger (which defaults to 1S). At the report time as determined by the report schedules, the statistical summary is reported. If insufficient samples have been taken before the reporting time, an error is reported (-9e9). .
	SD	Standard deviation of channel readings		
	MX	Maximum channel reading		
	MN	Minimum channel reading		
	TMX	Time of maximum channel reading		
	TMN	Time of minimum channel reading		
	DMX	Date of maximum channel reading		
	DMN	Date of minimum channel reading		
	IMX	Instant (time and date) of maximum		
	IMN	Instant (time and date) of minimum		
	INT	Integral for channel (using time in seconds)		
	NUM	Number of samples in statistical calculation		
	H*x***:***y***:***m***..***n***CV**	Histogram	*x*, *y* ±1e18 *m*, *n* 1-800	Divide data range *x* to *y* into discrete buckets and accumulate in CVs the number of samples in each bucket See *Histogram (Hx:y:m..nCV)* (P67)
Variables See *Channel Variables (nCV)* (P80).	**=***n***CV**	Assign channel reading to channel variable. *n***CV** = reading	1 to 800	Channel variables are like memory registers in a calculator. They can be assigned directly (e.g. **1CV=2.5**), or assigned a channel reading at scan time (e.g. **1V(=7CV)**). The contents of a variable can also be read, modified and then updated. For example **1V(/=7CV)** means that the value of 7CV is divided by the reading on channel 1 and the result is returned to 7CV. Note that these actions occur only at report time and not during statistical sampling.
	+=*n***CV**	Add channel reading to channel variable. *n***CV** = *n***CV** + reading	1 to 800	
	–=*n***CV**	Subtract channel reading from channel variable. *n***CV** = *n***CV** – reading	1 to 800	
	=n***CV**	Multiply channel variable by channel reading. *n***CV** = *n***CV** * reading	1 to 800	
	/=*n***CV**	Divide channel variable by channel reading. *n***CV** = *n***CV** ÷ reading	1 to 800	
Destination	**NR**	No return		Channels tagged with **NR** are not returned to the host computer (they may still be logged or displayed).
	NL	No log		Channels tagged with **NL** are not logged (they may still be returned or displayed).
	ND	No display		Channels tagged with **ND** are not displayed on the LCD (they may still be returned or logged).
	W	Working channel		Same as **(NR,NL,ND)** Working channels are usually used to hold intermediate values in calculations.

Output Data Format	**FF***n* *default = 1*		Fixed-point format *n*=decimal places	0 to 7	Specifies numeric format for display and free format (**/h**) real-time data. For example, **FF2** returns `71.46 mV` For example, **FE2** returns `7.14e1 mV` **FM***n* uses exponential format if exponent is less than –4 or greater than *n*
	FE*n*		Exponential format, *n*=decimal places	0 to 7	
	FM*n*		Mixed: FF or FE, *n*=significant digits	0 to 7	
	BG*min* **:** *max*				Show value as a bargraph on the display.
Channel name and Units	**"***name***"**		User-specified name Default units	ASCII text	Allows channel name and/or units to be overridden for display and free format (**/h**) real-time data. Max 16 characters for user-specified channel name; 8 characters for units. If
	"*name***~***unit***"**		User-specified name User-specified units		
	"*name***~"**		User-specified name No units		
	"~*unit***"**		No channel name User-specified units		
	"~"		No channel name No units		

Table 3: DT80 Channel Options

Part C — Schedules

Schedule Concepts

What are Schedules?

Schedules (full name: **schedule commands**) are the workhorses of the *DT80*. They are the underlying structures that you use to manage the repetitive **processes** of the *DT80* such as

- scanning input channels
- evaluating calculations
- processing alarms
- managing output channels
- returning data to a host computer
- logging data.

The *DT80* supports the following schedules:

- 11 **general purpose** schedules, A-K, which can be triggered by a variety of different events
- a **polled** schedule, X, which is normally triggered by a poll command from the host computer (although most of the other triggers can also be applied to it, making it effectively a 12th general purpose schedule)
- the **immediate** schedule, which executes once immediately after being entered
- the **statistical** schedule, which collects and accumulates data to be returned as statistical summaries by the other schedules.

Schedule Syntax

A typical schedule definition is shown below:

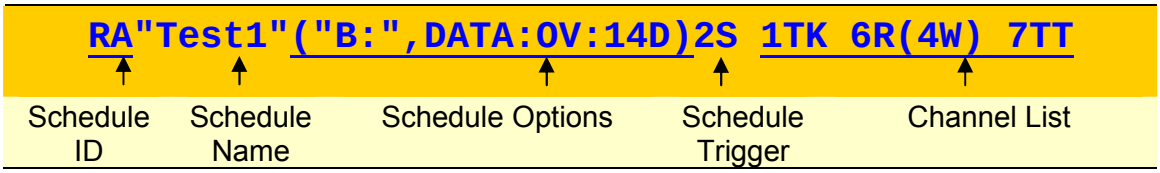
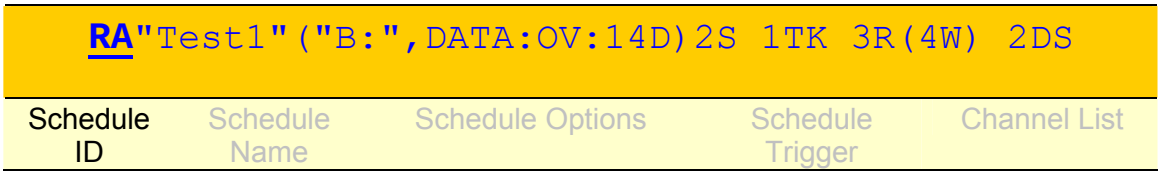


Figure 8: Components of a typical schedule command

A schedule consists of a number of parts. Firstly the Schedule ID, next the schedule options and finally the schedule trigger. There are no spaces between the different parts

Schedule ID

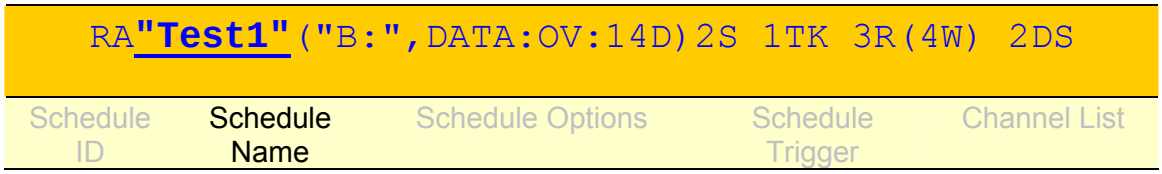


The **Schedule ID** consists of the letter **R** ("Report schedule") followed by a letter identifying the schedule.

Each schedule has a unique identifier; these are summarized in the following table:

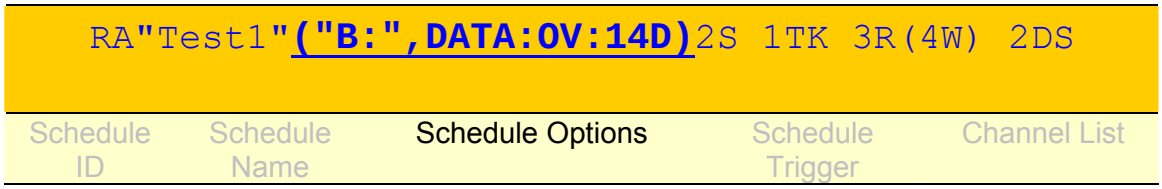
	Schedule ID	Quantity
Report schedules *(P45)*	**RA, RB, RC, RD, RE, RF, RG, RH, RI, RJ, RK**	11 available
Polled schedule *(P49)*	**RX**	1 available
Immediate report schedule *(P49)*	No schedule ID	Each time you enter an immediate schedule it replaces the previous one, if any.
Statistical sub-schedule *(P50)*	**RS**	1 available, but is applied to one or more of the other report schedules

Schedule Name



The schedule name is optional, and consists of text (max 8 characters) enclosed in double quotes. This name is used in reports and is normally used to document the purpose of a given schedule.

Schedule Options



Schedule options are enclosed in brackets. They define:

- where to log the data and alarms generated by the schedule (ie. internal file system or removable USB device)
- the amount of space to allocate for storing **data** records, and whether old records are to be overwritten when it is full.
- the amount of space to allocate for storing **alarm** records, and whether old records are to be overwritten when it is full.

For more details on how data logging works, see *Logging Data* (P80)

The schedule option syntax is as follows:

(*Dest***,DATA:***DataOverwrite***:***DataSize***,ALARMS:***AlarmOverwrite***:***AlarmSize***:***AlarmWidth***)**

Item	Possible Values	Explanation
Dest	**"A:"**	Logged data and alarm information from this schedule will be stored on the USB memory device.
	"B:"	Logged data and alarm information from this schedule will be stored on the internal flash disk **(default)**
DataOverwrite	**OV**	Data for this schedule will be overwritten when the data store is full **(default)**
	NOV	Data will NOT be overwritten when the data store is full. When the data store fills logging will stop and the 'Attn' LED will flash
DataSize	*n***B**	Allocate *n* bytes for storing data for this schedule
	*n***KB**	Allocate *n* kilobytes for storing data for this schedule
	*n***MB**	Allocate *n* megabytes for storing data for this schedule
	*n***R** *Note 2*	Allocate space for *n* data records for this schedule
	*n***S** *Note 1*	Allocate space for *n* seconds worth of data for this schedule
	*n***M** *Note 1*	Allocate space for *n* minutes worth of data for this schedule
	*n***H** *Note 1*	Allocate space for *n* hours worth of data for this schedule
	*n***D** *Note 1*	Allocate space for *n* data worth of data for this schedule
AlarmOverwrite	**OV**	Alarms for this schedule will be overwritten when the store is full **(default)**
	NOV	Alarms will NOT be overwritten when the store is full. When the store fills logging will stop and the 'Attn' LED will flash
AlarmSize	*n***B**	Allocate *n* bytes for storing alarms for this schedule
	*n***KB**	Allocate *n* kilobytes for storing alarms for this schedule
	*n***MB**	Allocate *n* megabytes for storing alarms for this schedule
	*n***R**	Allocate space for *n* alarm records for this schedule
AlarmWidth	**W***n*	Allocate *n* bytes for storing each alarm string **(default=60 bytes)**

Note 1: These are only valid for time-triggered schedules (not for polled or event triggered schedules). Furthermore, if the schedule rate is changed after the job has started running then the store file may no longer contain data for the indicated time span.

Note 2: A special **discontinuity record** is logged every time the schedule is halted (see *Halt and Go During Data Logging* (P83)). Thus if you specify a schedule option **DATA:10R**, you may get fewer than 10 actual data points logged, because some of the space may be taken up by discontinuity record(s).

Default Schedule Options

All schedule options are optional. Default settings are:

- Destination is **B:** (internal flash drive)
- New data and alarms **overwrite** earlier data/alarms once the store file fills

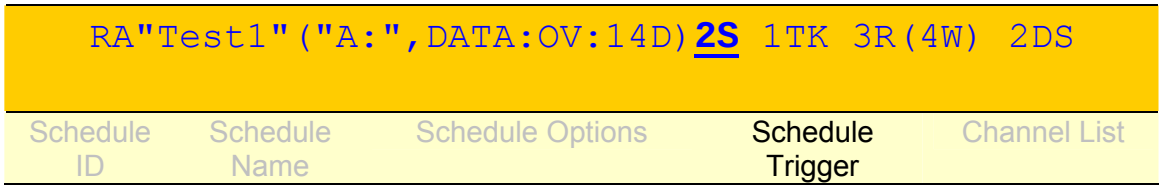
- Space allocated for data is **1MB**
- Space allocated for alarms is **100KB**
- Space allocated for each logged alarm text string is 60 bytes.

Examples

RA"Fred"(DATA:NOV:15D)15M

Schedule A is given the name "Fred". Data and alarms are stored on the internal drive and sufficient space is allocated for 1440 readings (15 days worth, based on a 15 minute scan rate). In this case earlier data is considered more valuable than later data, so no-overwrite mode is selected. If any alarms are defined in this schedule they will use the default storage parameters (100KB, overwrite enabled)

Schedule Trigger

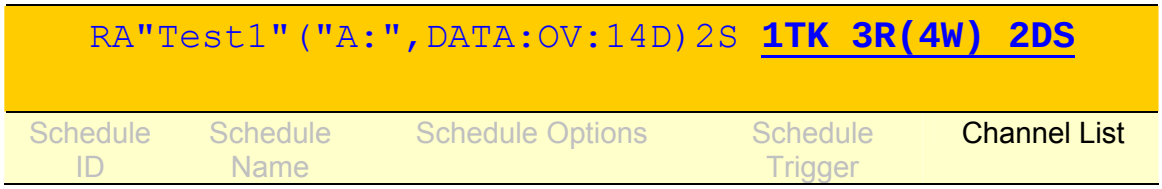


All schedules have a **trigger**, which defines when the schedule is to execute the processes assigned to it. Here are the *DT80*'s schedule triggers:

- an interval of **time**, see *Trigger on Time Interval* (P45)
- an **external event** (such as a digital input transition), see *Trigger on External Event* (P46)
- an **internal event** (such as a CV changing), see *Trigger on Internal Event* (P46)
- a **poll command**, see *Trigger on Schedule-Specific Poll Command* (P47)

Triggers can also be conditional upon an external or internal state (that is, trigger only while a particular external state or internal state exists) — see *Trigger While* (P48).

Channel List



Most often schedules will be created that instruct the *DT80* to carry out channel-related tasks, such as scanning one or more of its input channels and/or setting one or more of its output channels. When these schedules are created, group the channel details (their IDs and optional instructions) together in a **channel list** within the schedule. *Figure 8* (P42) shows a typical schedule — notice its schedule header and channel list components.

A channel list may contain just one channel entry or many, and each channel in the list must be separated from the next by one or more space characters. Similarly, a schedule's header must be separated from its channel list by one or more space characters.

The *DT80* processes the channels in a channel list from left to right.

Example — Channel List

The channel list

1V 3R 5..7DS 4TK("Boiler Temp") 3DSO=0

specifies the following channels (each is separated from the next by a space character):

- **1V** — read analog input channel 1 as a voltage
- **3R** — read analog input channel 3 as a resistance
- **5..7DS** — read the state of digital input channels 5 through 7 (inclusive)
- **4TK("Boiler Temp")** — read analog input channel 4 as a type K thermocouple and assign it the name **Boiler Temp**
- **3DSO=0** — set digital state output channel 3 low

Note that the example above is only a channel list and not a complete schedule. Here's the same channel list used in a schedule (the schedule header **RJ2M** has been added):

RJ2M 1V 3R 5..7DS 4TK("Boiler Temp") 3DSO=0

The header identifies the schedule as **R**eport schedule **J** that runs every **2 M**inutes.

A Simple Schedule

A schedule comprises a schedule ID (schedule identifier), a trigger that determines when the schedule runs, and a list of processes to be carried out every time the schedule runs. For example, the schedule

RA10M 1V 3R

specifies report schedule A as follows:

- **RA** — schedule ID
- If logging is enabled then data will be stored to the internal flash disk, 1MB will be allocated and old data will be overwritten when full. This schedule does not define any alarms, so no alarm storage will be allocated.
- **10M** — trigger (run the schedule every 10 minutes)
- **1V 3R** — channel list

Groups of Schedules — Jobs

A *DT80* **job** is essentially a group of one or more schedules (each specifying a set of processes) that performs the overall task.

It's entirely at the user's discretion how the processes of an overall task are allocated between schedules; there are no hard-and-fast rules. For example, choose to differentiate schedules on the basis of function or purpose — some collect primary data, others perform intermediate calculations, others process alarms, and yet others are responsible for returning and logging data; or choose to assign a schedule to a single channel, such as the *DT80*'s Serial Channel.

See also *Jobs* (P21).

Types of Schedules

General-Purpose Report Schedules (RA, RB,…RK)

The *DT80* supports eleven general-purpose **report schedules**, which you use to carry out the repetitive processes of scanning input channels, evaluating calculations, handling alarms, managing output channels, returning and logging data, and so on.

These report schedules have the identifiers **RA**, **RB**, **RC**, **RD**, **RE**, **RF**, **RG**, **RH**, **RI**, **RJ** and **RK**.

A report schedule executes the processes assigned to it whenever it is triggered. A schedule trigger can be

- an interval of time
- an external event
- an internal event
- a poll request from a host computer.

Trigger on Time Interval

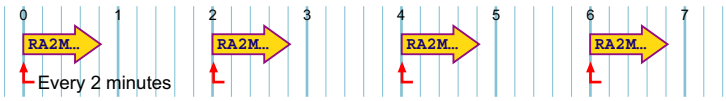


Figure 9: Time interval schedule

Report schedules can be triggered at regular intervals of time, determined by the *DT80*'s real-time clock. Intervals can be an integer number of seconds, minutes, hours or days:

Trigger	Run every n	Range
*n*D	Days	1<*n*<65535
*n*H	Hours	1<*n*<65535
*n*M	Minutes	1<*n*<65535
*n*S	Seconds	1<*n*<65535
*n*T	Milliseconds (Thousandths of seconds)	5<*n*<65535
none	Continuous	

Note The schedule first runs on the next multiple of the interval since last midnight (see *Time Triggers — Synchronizing to Midnight* (P51)), and subsequently runs every multiple of the interval thereafter. If the interval is not an even multiple of 24 hours, the *DT80* inserts a short interval between the last run of the schedule prior to midnight, and the run of the schedule beginning at midnight.

Examples — Trigger by Time Interval

The schedule header

RA5S

instructs the *DT80* to run **R**eport schedule **A** every 5 seconds (**5S**).

The schedule header

RG

instructs the *DT80* to run **R**eport schedule **G** continuously (as fast as possible).

Trigger on External Event

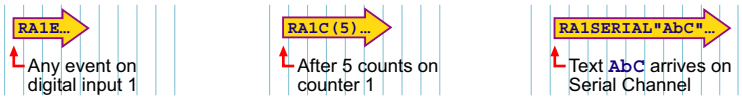


Figure 10: Various external event schedules

Report schedules can also be triggered by external events, which are manifested to the logger as state changes on the digital input channels *n***DS**, or as pulses on the counter channels *n***C**:

Trigger	Action
*n***E**	Trigger on a rising or falling transition of digital input channel *n*
*n***+E**	Trigger on a rising transition of digital input channel *n*
*n***-E**	Trigger on a falling transition of digital input channel *n*
*m***..***n***E**	Trigger on a rising or falling transition of any of digital input channels *m***..***n*
*m***..***n***+E**	Trigger on a rising transition of any of digital input channels *m***..***n*
*m***..***n***–E**	Trigger on a falling transition of any of digital input channels *m***..***n*
*n***C(***c***)**	Trigger after *c* counts on a low speed counter channel *n*
1SERIAL"*text***"** *(DT80 only)*	Trigger on the arrival of characters (from an external serial device) at the *DT80*'s Serial Channel. The trigger can be of the form **1SERIAL""**, where any character arriving triggers the schedule (note that there is no space between **""**), or **1SERIAL"AbC"**, where arrival of the exact string **AbC** triggers the schedule. See *Serial Channel* (P166).

where

n	is a digital channel number
*m***..***n*	is a sequence of digital channel numbers (see *Digital Channels* (P155))
text	is a string of characters arriving at the *DT80*'s Serial Channel terminals from an external serial device

Note: For edge triggering the minimum pulse width is approximately 16ms.

Triggering on Preset Counters

If a counter is preset to a value greater than its specified trigger count, the schedule is not triggered. For example, a schedule set to trigger after 10 counts on digital counter 2 (**2C(10)**) cannot be triggered if counter 2 is assigned a value of 15.

Examples — Trigger on Digital Channel Event

The schedule header

RC1E

instructs the *DT80* to run **R**eport schedule **C** on every transition of digital input 1 (**1E**).

The report schedule trigger

RA3+E

instructs the *DT80* to run **R**eport schedule **A** whenever digital input channel 3 receives a low to high (positive/rising) transitions.

Examples — Trigger on Serial Channel Event

The schedule header

RB1SERIAL"Pasta8zZ"

instructs the *DT80* to run **R**eport schedule **B** on the arrival of the specific character sequence **Pasta8zZ** at the *DT80*'s Serial Channel terminals.

The schedule header

RG1SERIAL""

instructs the *DT80* to run **R**eport schedule **G** on the arrival of any character at the *DT80*'s Serial Channel terminals.

Trigger on Internal Event

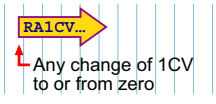


Figure 11: Internal event (CV change) schedule

Report schedules can also be triggered by internal events, this must be specified to the *DT80* as channel variables (CVs) changing value:

Trigger	Action

*n*CV	Trigger on any change of *n*CV to zero or from zero
n+CV	Trigger on any change of *n*CV from zero
n–CV	Trigger on any change of *n*CV to zero

where *n* is the channel variable number. (See *Channel Variables (nCV)* (P80).

Examples — Trigger on Internal Event

The schedule header

```
RK6CV
```

instructs the *DT80* to run **R**eport schedule **K** upon any change of channel variable 6 to or from zero. For instance, the schedule RK

- will trigger when 6CV changes from 0.0 to 1.0, from 0.06 to 0.0, or from –1.3 to 0.0
- will not trigger when 6CV changes from 0.0 to 0.0, 7.0 to 6.0, or from –112.3 to 0.001.

The schedule header

```
RK12+CV
```

instructs the *DT80* to run **R**eport schedule **K** whenever the value of channel variable 12 changes from 0 to any value.

Trigger on Schedule-Specific Poll Command

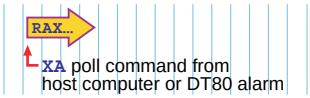


Figure 12: Polled schedule

Instead of a time or event trigger, the poll trigger (**X**) can be applied to a report schedule. Then the schedule can be polled (that is, information requested) at any time by the appropriate schedule-specific poll command **X***a* (where *a* is the schedule letter).

The poll command can be issued

- by a host computer, or
- by an alarm action (see *Using an Alarm to Poll a Schedule* (P78)).

See also *Using Digital Outputs* (P158)

Example — Trigger on Schedule-Specific Poll Command

The schedule

```
RDX 1..3TK
```

samples analog channels 1 to 3 as type K thermocouples (**1..3TK**) whenever the *DT80* receives an **XD** poll command (that is, whenever it receives the character sequence **XD**) either from a connected computer, or by means of an alarm action from within the *DT80*.

Using Poll Commands with Standard Report Schedules

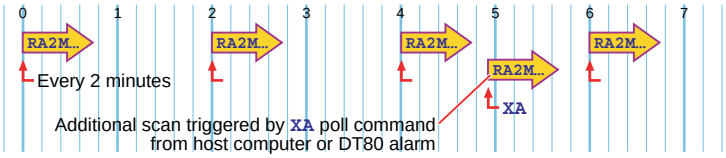


Figure 13: Polling a time interval schedule

A report schedule defined with a time or event trigger can also be polled by its appropriate poll command at any time. For example, the report schedule

```
RC5M 1V 2V 3V
```

normally runs every 5 minutes (**5M**), but it can also be run at any time by an **XC** poll command (from the host computer or an alarm).

For schedules that have a long interval, this is useful for checking that a sensor is functioning.

Trigger While

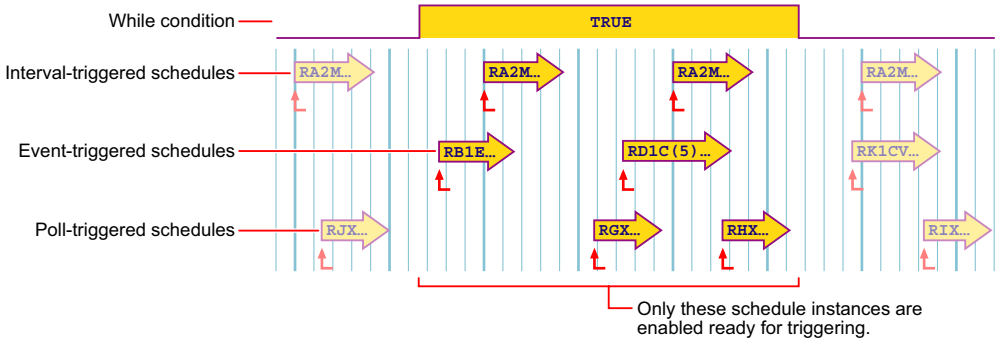


Figure 14: Schedule "while" condition

A report schedule's trigger can be enabled or disabled by an external condition. This is called the **While** condition — that is, trigger only while the external or internal condition is true.

The While condition can be either

- states on one or more of the *DT80*'s digital input channels (*n***DS**), or
- internal conditions specified to the *DT80* as states of channel variables:

While clause	Action
:*n***W**	Enable schedule While digital input *n* is high (true)
:*n***~W**	Enable schedule While digital input *n* is low (false)
:*m*..*n***W**	Enable schedule While ANY digital input *m* to *n* is high (true)
:*m*..*n***~W**	Enable schedule While ANY digital input *m* to *n* is low (false)
:*n***CV**	Enable schedule While *n***CV** is non-zero
:*n***~CV**	Enable schedule While *n***CV** is zero

Note that the colon (**:**) is required.

Examples — While Condition

The schedule header
RA1E:2W
instructs the *DT80* to run **R**eport schedule **A** on every transition of digital input 1 (**1E**) only while digital input 2 is high (**:2W**).

The schedule header
RD1S:4~W
instructs the *DT80* to run **R**eport schedule **D** every second (**1S**) while digital input 4 is low (**:4~W**).

The schedule header
RK2H:9W
instructs the *DT80* to run **R**eport schedule **K** every two hours (**2H**) while digital input 9 is high (**:9W**).

The schedule header
RC5M:12CV
instructs the *DT80* to run **R**eport schedule **C** every 5 minutes (**5M**) while channel variable 12 is not zero (**:12CV**).

The schedule header
RF6..8E:5W
instructs the *DT80* to run **R**eport schedule **F** on any transition of digital channels D6, D7 or D8 (**6..8E**) while digital input 5 is high (**:5W**).

Continuous Report Schedules (No Trigger)

Figure 15: Continuous schedule

Report schedules that run continuously can be created. These schedules start scanning as soon as they are received by the *DT80* (they are not activated by a trigger), and run until it is stopped (by sending a halt command or resetting the *DT80*, for example).

Define a continuous schedule simply by omitting the trigger from a report schedule.

Example — Continuous Schedule

Sending
RA 1TK 2R(3W) 3TT
causes the *DT80* to scan channels 1, 6 and 7 continuously.

Special-Purpose Report Schedules

Polled Report Schedule (RX)

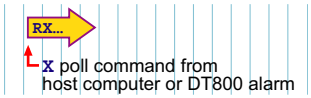


Figure 16: Polled X schedule

The **polled schedule** is a report schedule with a trigger of "request information now" command issued

- by a host computer connected to the *DT80* during data acquisition, or
- by an alarm action (see *Using an Alarm to Poll a Schedule* (P78)).

The *DT80* supports one polled schedule. It is specified by the **RX** schedule ID, and triggered by an **X** poll command (that is, by an **X** character followed by CR) sent from the host computer or from an alarm.

Channels, calculations and alarms included in a polled schedule are processed, reported and/or logged once every time the *DT80* receives an **X** poll command.

Note that underneath, the X schedule is really the same as the general purpose schedules. It can therefore be used as a 12th general purpose schedule, except that:

- the continuous trigger is not available. The syntax **RX** is treated the same as **RXX**, ie. it specifies a polled trigger, not a continuous trigger,
- a single **X** character can be used to poll the schedule, which is treated the same as **XX**.

Example — Polled Report Schedule

The schedule

```
RX 1V 2V
```

runs once every time the *DT80* receives an **X** (or **XX**) command.

Immediate Report Schedules

Figure 17: Immediate schedule

Instead of scanning according to time or event triggers, **immediate schedules** run immediately — and once only — when they are received by the *DT80*.

An immediate schedule is simply a list of input channels, output channels, calculations and/or alarms with no schedule header (that is, no schedule ID and no trigger). The *DT80* executes the list (up to the next carriage return) immediately and once only.

Note Any data resulting from an immediate schedule is returned to the host computer, but is not logged.

Example — Immediate Report Schedule

Sending

```
1TK 2R(3W) 3TT
```

causes the *DT80* to immediately scan channels 1, 6 and 7 once only and return the data. Notice that this schedule has no schedule ID and no trigger.

Cautions for Using Immediate Schedules

When programming the *DT80*, give an immediate schedule time to execute before issuing a following **BEGIN** command, otherwise the immediate schedule's data may not be returned. Using DeTransfer can be helpful, by inserting a **\W** wait command (for example, **\W5**, which pauses program execution for five seconds — between immediate schedule commands and a **BEGIN** command.

If successive immediate schedules are entered too rapidly, then the channels may be appended as if they were part of a single schedule. Setting P22=13 (see *P22* (P130)) can overcome this by ensuring a return character is placed after each reading.

Re-Running an Immediate Schedule

The last-entered immediate schedule can be run again by sending the * (asterisk) command — that is, by sending a * character.

Statistical Report Schedules

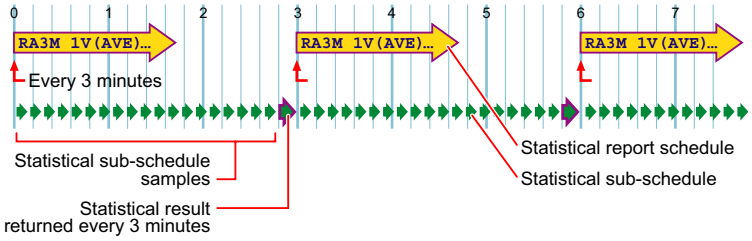


Figure 18: Statistical and report schedules

A report schedule can instruct the *DT80* to return statistical information (average, SD, max., min.,…) for one or more channels. The *DT80* does this by

- scanning its input channels and executing calculations at frequent intervals of time, then
- retaining intermediate values to produce a statistical data summary at longer intervals.

Note that there are two schedules involved:

- The primary statistical data is collected at frequent intervals, which are determined by the **statistical sub-schedule RS**.
- The statistical data summary (average, SD,…) is returned and logged at longer intervals, which are determined by the report schedule that is requesting the statistical information — the **statistical report schedule**.

Think of the statistical sub-schedule as a fast schedule (the slave) running within/below its slower statistical report schedule (the master). This is why **RS** is called the statistical sub-schedule.

The statistical sub-schedule has its own interval trigger. The default is one second, but you can change that — see *Redefining the Statistical Sub-Schedule's Trigger* (P50) below.

To return statistical data, include — in any report schedule — a statistical channel option for the specific input channels, calculations, and so on where statistically scanned is required. For example

```
RA1H 2TT(AV)
```

returns, every hour (**RA1H**), the average of one-second readings (because one second is the default scan rate for the statistical sub-schedule) taken from the type T thermocouple connected to channel 2 (**2TT(AV)**).

Note:

- Simply including a statistical channel option (**(AV)** in the example above) invokes the statistical sub-schedule.
- There is no need to include **RS**, the statistical sub-schedule's ID, anywhere (unless you want to alter **RS**'s trigger — see *Redefining the Statistical Sub-Schedule's Trigger* (P50) below).

For details of the statistical channel options available, see *Statistical Channel Options* (P65).

Redefining the Statistical Sub-Schedule's Trigger

The statistical sub-schedule's trigger can be altered from its default of one second. Define the statistical sub-schedule's trigger in the same way as for report schedules (see *Changing a Schedule Trigger* (P51)), by using the **RS** schedule ID and sending an **RS...** schedule command to the *DT80*. If you don't specify the **RS** schedule's trigger in this way, it defaults to once per second. Here are some examples:

The schedule header	instructs *DT80* to accumulate specified statistical data
RS10S	every 10 seconds
RS1-E	on each 1 to 0 transition of digital input 1
RS	continuously

Statistical Sub-Schedule Halt/Go

The statistical sub-schedule can be halted by sending the **HS** command, and start it again by sending the **GS** command.

Important Because statistical sampling of channels stops the moment the **HS** command is sent, be aware that the reported statistical summaries do not include data from this halt period. This is most significant for the integral summary.

See also *Halting & Resuming Schedules* (P51).

Multiple Statistical Information for a Channel

If more than one type of statistical information is required for a channel, then each statistical option must be placed in a separate channel option list. For example, the channel list

```
1TT(AV)(SD)(MX)
```

results in periodic average, standard deviation and maximum data for the **1TT** channel.

Insufficient Statistical Samples

If no statistical data has been scanned before being reported, then the reported data value will be set to **-9.0e9**, a special value that indicates "not yet set". This will also occur if **insufficient** samples have been taken – for example, the standard deviation (**SD**) option requires at least two samples to be able to return a value.

This condition may occur when

- the statistical sub-schedule is event-triggered
- the statistical sub-schedule has been halted
- a statistical sub-schedule scan interval is longer than its statistical report scan interval.

Example — Statistical Report Schedule

The command

```
RS10S RA1H 1TT 2TT(AV)(MX)
```

sets the statistical sub-schedule's scan rate to 10 seconds (**RS10S**). The main report schedule returns three temperature readings: a spot reading of channel 1 each hour (**RA1H 1TT**), and the average and maximum over the hour from 10-second samplings of channel 2.

Working with Schedules

Entering Schedules into the *DT80* (BEGIN–END)

Report schedules must be entered into (that is, sent to) the *DT80* as a group. Since the schedules and processes that comprise a job or program often extend over more than one line, you normally enclose them between the keywords **BEGIN** and **END** to designate the beginning and end of the group. Here's an example:

```
BEGIN"walrus"
 1DSO=1
 3CV(W)=2
 RA10S
  4TT("Oven Temp")
  5TT("Flue Temp")
 RB1S
  2C("Water Flow")
END
```

A group of schedules such as this is called a **job**. In the above example, the job has been named "walrus" and comprises two report schedules, **RA** (which measures two thermocouples every 10 seconds) and **RB** (which measures a counter once a second). Note also the two "immediate" channels (**1DSO** and **3CV**) which are not part of any schedule. These are executed once only, when the job is entered.

For more details on how jobs are entered and processed, see *Jobs* (P54)

Triggering and Schedule Order

When different schedules are due to trigger at the same time, the schedules execute in the order of **RA**, **RB**,…**RK**.

When there are statistical channels in a schedule and the statistical sub-schedule is due at the same time as the report schedule, the statistical sub-schedule runs prior to the report schedules. You cannot change this order.

Channels within schedules are sampled in the order of entry (left to right).

Changing a Schedule Trigger

The schedule's trigger can be changed at any time simply by sending a new schedule ID and trigger without any channel definitions. For example, suppose a schedule had been defined as follows:

```
RA10M 1V 2DS
```

This will measure a voltage and a digital input every 10 minutes. If you then send:

```
RA10S
```

the schedule will then, from that point on, measure every 10 seconds.

Important If any channel definitions are included on the same line (eg **RA10S 2V**) then this will be interpreted as a whole new job being entered, which will replace the currently running job.

Halting & Resuming Schedules

Schedules can be halted individually or as a group using the following commands:

Command	Function
H	Halt all schedules
HA, **HB** … **HK**	Halt **RA**, **RB**,…**RK** schedule
HS	Halt the statistical sub-schedule (see *Statistical Sub-Schedule Halt/Go* (P50))

Schedules can then be resumed ("GOed") individually or as a group:

Command	Function
G	Resume all schedules
GA, **GB** … **GK**	Resume **RA**, **RB**,…**RK** schedule
GS	Resume the statistical sub-schedule (see *Statistical Sub-Schedule Halt/Go* (P50))

Executing Commands in Schedules

It is important to distinguish between **commands** and **channel definitions**. Commands (eg. **H**, **/S**, **P11=60**, **DIRJOB**, **COPY** etc.) are always executed once only, immediately they are received – even if they appear to be within a schedule definition.

For example, if you enter

```
RA1+E 1V HB SATTN 4CV=4CV+1
```

you might expect that when the schedule was triggered (by a positive going edge on digital input 1) it would measure a voltage (**1V**), halt schedule B (**HB**), switch on the **Attn** LED (**SATTN**) and increment a channel variable (**4CV=4CV+1**).

In fact, the **HB** and **SATTN** will execute once only, when the job is entered. The **1V** and **4CV=4CV+1** are channel definitions, so they will execute each time schedule A is triggered.

To execute commands within a schedule, the **DO** construct can be used. This is actually a special case of the **ALARM** statement (see *Alarms* (P71)) – one where the condition is always true. The syntax is the same as **ALARM** except that there is no test condition.

So if the above job was rewritten as:

```
RA1+E 1V DO{HB SATTN} 4CV=4CV+1
```

then the following actions would be performed each time schedule A is triggered:

- Channel **1V** is measured, then channel variable **4CV** is incremented
- The commands **HB** and **SATTN** are queued for execution. They will be actioned "as soon as possible" – once all schedules that are currently due have completed, and any previously queued commands have been executed.

Note that this means that it is not possible to interleave the execution of commands and channels within a schedule. Channels are always performed first; commands are executed a short time later.

Commands can also be executed conditionally, using the **IF** construct (which in fact is just a synonym for **ALARMR** – repeating alarm), eg.

```
RA1M IF(1CV>3.57){XB}
```

will test the value of 1CV once a minute. If it exceeds 3.57 then schedule B will be triggered.

For more details on **ALARM**/**IF**/**DO** syntax and usage, see *Alarms* (P71).

Time Triggers — Synchronizing to Midnight

Time triggers for report schedules function in two different ways depending on the setting of the synchronize-to-midnight switch (**/s** or **/S**, see (P132)).

Synchronize-To-Midnight Switch Enabled

If the synchronize-to-midnight switch is enabled (**/S**, the *DT80*'s default), the intervals of all schedules with time triggers are synchronized to the previous midnight.

When a time-triggered schedule is entered, the schedules first run on the next multiple of the interval since last midnight, and subsequently run on every multiple of the interval thereafter.
If the interval is not an even multiple of 24 hours, the *DT80* inserts a short interval between the last run of the schedule prior to midnight and the next run of the schedule at midnight.

For example, if you send the schedule

```
RA10H
```

to the *DT80* at 06:00:00, it first runs at 10:00:00 (4 hours since entry, but 10 hours since midnight) and then at 20:00:00 that day; then at 00:00:00, 10:00:00 and 20:00:00 the next day; and so on.

If you enter an interval longer than 24 hours then the interval is rounded down to the nearest multiple of 24 hours. So if the schedule

```
RA50H
```

was entered at 09:00 Monday morning then the schedule will first run at 00:00 Wednesday morning, then every 48 hours thereafter.

Synchronize-To-Midnight Switch Disabled

If the synchronize-to-midnight switch is disabled (**/s**), the schedules run at intervals relative to the time that the schedule is entered. For example, if the same **RA10H** schedule is sent to the *DT80* at 09:30:00, it first runs at 19:30:00 that day; then at 05:30:00 and 15:30:00 on the next day; at 01:30:00 and 11:30:00 on the following day; and so on. That is, every 10 hours of

elapsed time.

Note that the base time (the time at which the specified schedule interval begins) is reset whenever:

- the schedule rate is changed, or
- the schedule is restarted (using the **G** command), or
- the system time is changed (using the **D=**, **T=** or **DT=** commands)

Part D — Jobs

What is a Job?

A **job** is a collection of related schedule definitions and commands which together configure the *DT80* to perform a particular data logging task.

Several different jobs can be stored on the *DT80*'s internal file system, but only one can be active at any one time. Each job has its own separate data/alarm storage area.

Jobs are identified by their **job name**, which is a user-defined string of up to 8 characters. If a job name is not specified when the job is entered, the default name **UNTITLED** is used.

When the *DT80* is first started or reset, there is no active job. The logger is idle and `No current job` is displayed on the LCD to indicate this.

To make the logger do something useful, you need to either:

- enter a new job, or
- run (load) an existing job.

Once a job has been entered or loaded successfully, it becomes the currently active job and its name will be displayed on the LCD. If you then enter or load a different job, all schedules and channels defined by the original job are cleared and replaced by those of the new job.

Entering a Job

To enter a new job you send the required commands and schedule definitions to the logger using one of the communications ports (USB, Ethernet or RS232). Once the complete job has been entered, the *DT80* will automatically store the job in its internal file system and activate it.

To begin entering a new job, the **BEGIN** command is used. This command:

- causes the currently active job to be cleared. All defined schedules will stop running.
- specifies the name of the new job, eg. **BEGIN"GOOSE"** indicates that the new job will be called "GOOSE". (Just **BEGIN** by itself is equivalent to **BEGIN"UNTITLED"**.)
- places the *DT80* in "job entry mode". After each line of the job is entered the *DT80* will output a `job>` prompt, rather than the usual `DT80>` prompt.

As each line of the job is entered the *DT80* executes any commands or immediate channels that it finds. Report schedule definitions, including their constituent channel definitions are recorded but they are not activated just yet.

The **END** command marks the end of a job. At this point all schedules defined within the job are activated, and the *DT80* is no longer in job entry mode.

If an error occurs during job entry, the *DT80* will clear all schedule/channel definitions and ignore the remainder of the job, up until the **END** command is seen.

Note In some circumstances the *DT80* will not allow a new job to be entered:

- if the "fix schedules" switch *(P132)* is active (**/F**) – this prevents any change to the currently active job. (Use **/f** command to allow the current job to be changed.)
- if a different job with the same name already exists, and it has been locked using the **LOCKJOB"***jobname***"** command – this prevents a stored job being accidentally overwritten. (Choose a different job name, or unlock the existing job using **UNLOCKJOB"***jobname***"**.)
- if a different job with the same name already exists, and it has logged data or alarms – this prevents data from different jobs (which happen to have the same name) from being mixed up in the one data file. (Choose a different job name, or delete the existing job's using **DELDATA"***jobname***"** and/or **DELALARMS"***jobname***"**.)

Note An error message will be returned if you attempt to send more than 255 characters on a single line.

Single Line Jobs

As a shortcut, it is also possible to enter a job simply by entering one or more schedule definitions all on one line, eg:

```
RA1S 1TK
```

The above is then equivalent to:

```
BEGIN"UNTITLED" RA1S 1TK END
```

that is, it will create a new job called "UNTITLED".

Note that entering just a schedule trigger (with no channel definitions after it), eg.

```
RA2S
```

does not create a new job – it simply changes the trigger condition for the currently defined A schedule (if any).

It is recommended that, for clarity, **BEGIN** and **END** are always included explicitly when entering a job.

Loading an Existing Job

The *DT80* can also read job text from a file stored in its internal file system and automatically enter it. This job will then become the current active job, replacing whatever was previously the current active job.

A new job may be loaded when:

- the **RUNJOB"***jobname***"** command is issued. This will read the job from the file **B:\JOBS***jobname***\PROGRAM.DXC**.
- the *DT80* is reset. If a file **B:\ONRESET.DXC** is present then any commands therein will be executed. These may include a job definition, in which case the job will become the current active job. (See *Startup Job* (P57))
- A USB memory device is inserted. If a file **A:\ONINSERT.DXC** or **A:***serialnum***\ONINSERT.DXC** is present then any commands therein will be executed. These may include a job definition, in which case the job will become the current active job. (See *ONINSERT Job* (P57))

Job Structure

A typical *DT80* job is shown below (the line numbers are for reference only and are not part of the job)

```
1      BEGIN"Boiler01"
2      ' No. 1 Boiler monitoring job for DT80  03-Dec-2005
3      /n/u/S/e
4      P22=44
5      Y10=4.5,0.312"kPa"
6      S1=0,50,0,100"L/m"
7      1DSO=0    ' Enable sensor power relay
8      RS5S
9
10     RB1M 2..3TT("Temp")
11
12     RC(DATA:NOV:365D)15M 1V(Y10,AV) 4#L(S1,AV)
13
14     RK10S
15      ALARM1(1V(Y10)>2.25)3DSO
16      ALARM1(4TT>110.0)3DSO,1CV"Over Temp ?"{RB5S}"
17
18     LOGON
19     END
```

Note the following salient points:

- Line 1 – the **BEGIN** command tells the *DT80* to clear the current job and prepare to receive a new one.
- Line 2 – anything following a single quote character (up to the end of the line) is considered a **comment** and is ignored. Blank lines are also ignored.
- Line 3-7 – the first part of a job normally consists of commands to set switches and parameters, define polynomials and spans, and evaluate any immediate channels.
- Line 8 – this line sets the scan rate for the statistical sub-schedule. All channels which include a statistical channel option, ie. **1V(AV)** and **3#L(AV)**, will therefore be scanned every 5 seconds. The measured values will not be logged or returned; they will only be used for accumulating the average values.
- Line 10 – defines the B schedule (measure two thermocouples once per minute).
- Line 12 – defines the C schedule (report pressure and flow rate values, averaged over a 15 minute period. Note the use of a polynomial (**Y10**) to convert the measured voltage in mV to pressure in kPa. Similarly, a span is used to convert a current loop % value to a flow rate in l/m.)
- Line 14-16 – define the K schedule (every 10s check pressure and temperature against limits. If pressure exceeds 2.25kPa then set digital output 3 to LOW (active); if temperature exceeds 110°C then set digital output 3 to LOW, set 1CV=1, output and log an "Over Temp" alarm string, and change the scan rate of schedule B to 5s.)
- Line 18 – enables logging for all schedules (by default logging is disabled)
- Line 19 – marks the end of the job; all schedules will now be activated.

Job Commands

A number of commands are provided for managing jobs on the *DT80*.

Listing Job Names

The **DIRJOBS** command lists the names of all jobs stored in the *DT80* internal file system, eg.

```
DT80> DIRJOBS
  FRED
 *GEORGE
+ RON
+ GINNY
  UNTITLED
```

An asterisk (*) indicates the currently active job, if any. Locked jobs are indicated by a plus sign (+).

The **CURJOB** command simply displays the name of the current active job, eg.

```
DT80> CURJOB
GEORGE
```

Specifying Jobs

The following commands act on a specific job or jobs. To specify the job you can enter a *jobspec* parameter after the command, where *jobspec* can be either:

- a job name in double quotes, eg. **"FRED"**, or
- an asterisk (*****), which will apply the command to all stored jobs, or
- nothing, in which case the command will operate on the current active job (an error message will be reported if there is no currently active job).

Showing Program Text

To show the commands which define a job, use **SHOWPROG***jobspec*, eg.

```
DT80> SHOWPROG"RON"
Job Program - RON
BEGIN"RON"
RA1S 3TT 1DS
END
```

Locking Jobs

If a job is **locked** then its program text cannot be deleted or overwritten, and nor can its logged data or alarms. To lock a job use the **LOCKJOB***jobspec* command; to unlock use **UNLOCKJOB***jobspec*, eg:

```
DT80> LOCKJOB*
Locking Job FRED - Done
Locking Job GEORGE - Done
Locking Job RON - already locked
Locking Job GINNY - already locked
Locking Job UNTITLED - Done
```

Deleting Jobs

The **DELJOB***jobspec* command can be used to delete a job from the *DT80*'s internal file system.

However, this command will fail and the job will not be deleted if any of he following apply:

- the job is the current active job and the **/F** (fix schedules) switch is set (use **/f** to turn this switch off)
- the job is locked (use **UNLOCKJOB** to unlock it)
- the job has logged data and/or alarms (use **DELDATA** and **DELALARMS** to delete them)

Managing a Job's Logged Data and Alarms

The following commands allow you to manage the data and alarms logged by a job:

- **DIRJOB***jobspec* (not to be confused with **DIRJOBS**!) lists details of the number of logged records and the associated time range
- **U** and **A** (with various parameters) are used to **unload** data and alarms respectively – that is, output the data to the currently active comms port as fixed format (ie. comma separated) records
- **DELDATA***jobspec* and **DELALARMS***jobspec* will delete a job's logged data and alarms respectively (provided that the job is unlocked).

For more details on these commands, see *Logging and Retrieving Data* (P80).

Startup Job

The *DT80* can automatically load a user-defined job every time it is restarted by a hard reset (**SINGLEPUSH** command or power failure or pressing the manual reset button). This allows the *DT80* to operate as a dedicated instrument.

On startup, the *DT80* checks its internal file system for a text file **B:\ONRESET.DXC**. If found, the contents of the file are processed as commands in exactly the same way as if they had been received via a comms port.

To make the *DT80* automatically run an existing job on startup, use the command
RUNJOBONRESET"*jobname***"**
which will copy the specified job's program text to the **B:\ONRESET.DXC** file.

To remove the startup job:
DELONRESET

Note that the startup job is a copy of the original job, so deleting the original job using **DELJOB** will not delete the startup job. In fact, the next time the startup job is loaded (ie. the next time the logger is reset) it will re-create the original job, just as if the job had been re-entered via a comms port. The only way to remove the startup job is to use **DELONRESET** (or **FACTORYDEFAULTS**).

Note also that if a triple-push reset is performed, the startup job will not be loaded. This allows you to recover if there is some problem with the startup job which causes you not to be able to communicate with the *DT80*.

Backup Copy of ONRESET.DXC

When the **RUNJOBONRESET** command is issued, a second copy of the job is written to a hidden area of the *DT80*'s internal flash memory. If the **B:\ONRESET.DXC** file is accidentally deleted (eg. using **FORMAT"B:"**), it will be automatically restored from the flash copy.

Note that **DELONRESET** will delete both the **B:\ONRESET.DXC** file and the backup copy in flash memory.

ONINSERT Job

When a USB memory device is inserted into a *DT80*, the *DT80* first looks for a file on the USB device named **A:***serialnum***\ONINSERT.DXC**, where *serialnum* is the serial number of the *DT80* (eg. **SN80322**). If found, the commands in this file are entered into the *DT80* exactly as if they had been received via a comms port.

If the above file is not found, the *DT80* looks for a command file in the root directory, ie. **A:\ONINSERT.DXC**; if found it is loaded into the *DT80* in the same way.

This auto-programming function means that a single USB memory device can be inserted into a number of *DT80*s, one at a time, and either:

- automatically program all the *DT80*s with the same job — if no serial-number-specific subdirectories containing **ONINSERT.DXC** files exist on the card and an **ONINSERT.DXC** file exists at the root level, or
- automatically program particular *DT80*s with their own specific job — if serial-number-specific subdirectories containing **ONINSERT.DXC** files exist on the card, or
- carry out a combination of these two options — *DT80*s that do not find a subdirectory named with their serial number automatically load and run the "standard" **ONINSERT.DXC** file at the root level, and *DT80*s that find their specific subdirectory automatically load and run the "specific" **ONINSERT.DXC** file found there.

These files are typically created by inserting the USB memory device into a PC and copying the required program files to the required directories.

Alternatively, the *DT80* command:
RUNJOBONINSERT"*jobname***"**
can be used to copy the specified job's program text to **A:***serialnum***\ONINSERT.DXC**.

Similarly,
RUNJOBONINSERTALL"*jobname***"**
will copy the specified job's program text to **A:\ONINSERT.DXC**.

To delete the **ONINSERT.DXC** files from the inserted USB memory device, you can use the
DELONINSERT
and
DELONINSERTALL
commands.

Part E — Manipulating Data

Scaling

Most *DT80* channel types automatically scale measured values so that the returned values are in appropriate engineering units. For example, the thermocouple channel types (eg. **TK**) automatically apply the appropriate scaling polynomial so that the data is returned in °C. However, a number of additional facilities are provided for applying custom scaling or corrections:

- channel factor
- spans
- polynomials
- thermistor scaling
- intrinsic functions
- expressions

Channel Factor

For many channel types, the channel factor (a channel option consisting of just a floating point number) can be used to provide a simple multiplication factor. See *A Special Channel Option — Channel Factor* (P36).

For example, if a high voltage is being measured using an external 12.5:1 voltage divider then the following channel definition:

```
1V(12.5)
```

will multiply the raw reading by 12.5 so that the returned value reflects the actual voltage.

Note that for some channel types the channel factor performs a special function, and therefore cannot be used as a scaling factor. In these cases a span should be used (see below).

For example, if you are measuring a frequency which has passed through a 100:1 prescaler then you will need to use a span to scale it

Spans (S*n*)

A span transforms a measured **signal** value (eg. mV) into the corresponding **physical** value (eg. kPa) using a straight line function:

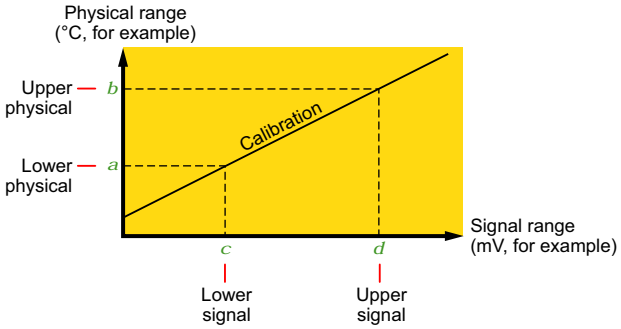


Figure 19: Span coordinates

A span must be **defined** before it is applied. This is normally done at the start of a job, before any schedules are defined. The syntax is as follows:

```
Sn=a,b,c,d"units"
```

where:

- *n* is the **poly/span number** (1 to 50), which is used simply to distinguish one span from another. Note that a span and a polynomial cannot have the same number.
- *a* and *b* are the **physical** coordinates of two points on the calibration line
- *c* and *d* are the **signal** coordinates of two points on the calibration line. If not specified, 0 and 100 are assumed.
- *units* replaces the channel's default units text

Spans are particularly suited to 4–20mA current loop inputs. The following defines a span suitable for a current loop sensor that measures pressure in the range 0-300kPa:

```
S2=0,300"kPa"
```

Note that in this case the default signal coordinates (0 and 100) are used, because the **L** (current loop) channel type returns

a value in the range 0-100%.

Once defined, a span may be **applied** to any number of channels in any schedules or alarms using the **S*n* channel option**.

In the following example, two of the above current loop pressure sensors are used, plus a frequency input which passes through a 10:1 prescaler (frequency divider):

```
BEGIN"MONKEY"
S1=0,10,0,1     ' multiply measured freq by 10
S2=0,300"kPa"   ' scale 0-100% to 0-300kPa
RA10S 1#L(S2,"Inlet") 2#L(S2,"Outlet") 4F(S1)
END
```

This will return data in the form:

```
Inlet 23.9 kPa
Outlet 119.0 kPa
4F 3920 Hz
```

(Note that for the **F** channel type the channel factor indicates the sample period, so it cannot be used as a scaling factor. A span is therefore used instead.)

Polynomials (Y*n*)

Polynomials are used to define calibrations for non-linear sensors. Each defined polynomial can have up to six polynomial coefficients.

The *DT80* evaluates a polynomial according to the formula

$$y = \sum_{n=5}^{0} k_n x^n = k_0 + k_1 x + k_2 x^2 + k_3 x^3 + k_4 x^4 + k_5 x^5$$

where x is the raw channel reading, and the k's are coefficient terms.

A polynomial is defined in a similar way to a span:

Y*n*=*k0*,*k1*,*k2*,*k3*,*k4*,*k5*"*units*"

where:

- *n* is the **poly/span number** (1 to 50), which is used simply to distinguish one polynomial from another. Note that a span and a polynomial cannot have the same number.
- *k0* … *k5* are the polynomial coefficients. If not specified, a coefficient value of zero is assumed
- *units* replaces the channel's default units text

The required coefficients may be supplied by the sensor manufacturer, or they may be determined from a calibration curve or from measured data points using a least squares regression technique. Various statistical programs are available for this purpose.

Once defined, a polynomial may be applied to any number of channels using the **Y***n* channel option. For example:

```
Y1=23.5,0,0.987"deg C"
RA1M 1V(Y1) 2V(Y1)
```

A "null" polynomial may also be used as a shortcut when defining several channels which all require the same custom units string, eg:

```
Y9=0,1"m/s"
RA10S 3CV(Y9) 5CV(Y9) 22CV(Y9) 33CV(Y9)
```

Thermistor Scaling (T*n*)

The *DT80* has channel types (eg. **YS03**) for many 2-wire YSI thermistors (Yellow Springs Instruments www.ysi.com). For other thermistor types, the *DT80* supports thermistor scaling — the conversion of a resistance reading to a temperature. The *DT80* does the conversion from resistance to temperature using

$$T = \frac{1}{a + b\mathrm{Ln}(R) + c\mathrm{Ln}(R)^3}$$

To apply thermistor scaling, firstly obtain the constant terms a, b and c from the thermistor manufacturer, then define a thermistor conversion in a similar way to a polynomial or span:

T*n*=*a*,*b*,*c*"*units*"

where:

- *n* is the **thermistor conversion number** (1 to 20), which is used simply to distinguish one thermistor equation from another.

- *a, b, c* are the constants from the above thermistor equation. If not specified, a coefficient value of zero is assumed
- *units* replaces the channel's default units text

Once defined, a thermistor conversion may be applied to any number of resistance channels using the **T***n* channel option. For example:

```
T1=26.5,1.034,8.77e-3"K"
RA1M 3R(T1,"Solvent temp")
```

See also Thermistors *(P150)*.

Intrinsic Functions (F*n*)

The *DT80* has seven inbuilt and mutually exclusive intrinsic functions that may be applied as channel options. The intrinsic functions available are

F*n*	Description		Text Modifier
F1	1/x	inverse	(Inv)
F2	√x	square root	(Sqrt)
F3	Ln(x)	natural logarithm	(Ln)
F4	Log(x)	base ten logarithm	(Log)
F5	Absolute(x)	absolute value	(Abs)
F6	x * x	square	(Squ)
F7	Grey code conversion (32-bit)		(Gc)

The text modifier is appended to the channel's default units string. If a channel's units string has been explicitly set (using the "*name*~*units*" channel option) then no modifier is appended.

If more than one intrinsic function is placed in a channel's channel option list, only the last is applied. Channel variables and expressions (P) can be used if multiple functions need to be combined.

For example, the channel definition

```
1V(F2) 2F(F1,"period~sec")
```

will return data in the form:

```
1V 455.7 mV (Sqrt)
period 1.7 sec
```

Calculations

Channel Variables (*n*CV)

Channel variables (CVs) are memory locations (registers) for holding and manipulating floating-point data. The *DT80* has 800 channel variables, identified as **1CV** to **800CV**.

All channel variables are reset to 0.0 when the *DT80* is reset (**SINGLEPUSH**) or cleared (**RESET**).

Reading Channel Variables

A channel variable behaves in much the same way as any other channel type. Its value may be read (ie. returned and/or logged) by including the appropriate *n***CV** channel definition in a schedule. For example, sending

```
12CV
```

will immediately return the value of channel variable #12, while

```
RA10S 1..5CV
```

will report the values of 1CV through 5CV every 10 seconds.

If a CV is being used for holding an intermediate value then you would normally use the **W** channel option to make it a working channel (which is not returned or logged).

Setting Channel Variables

A channel variable's value may be set in one of three ways:

- the CV is set to the value of an **expression**, ie. *n***CV=***expression*
- any channel's value may be assigned to a channel variable by using the **=***n***CV channel option**
- certain special channel options (eg. histogram) return multiple data values, which are written to a specified range of channel variables.

Some examples of using expressions to set CVs:

```
1..20CV=10.2        'initialise multiple CVs
```

```
RA1S 1V 9CV=9CV+1  'count the number of measurements taken
5CV(W)=3CV*SIN(21CV)+2CV*COS(21CV)
```

See also *Expressions* (P62)

The **=*n*CV** channel option allows a channel value to be assigned directly to a CV, typically so it can then be used in further calculations.

This can be used to apply a complicated linearisation equation, eg:

```
1V(=2CV,W) 2CV(S9,"temp~K")=2CV/(LN(2CV+1))
```

This will measure a voltage and assign it to 2CV (note the **W** option – we are not interested in logging/returning the raw voltage value). The value in 2CV is then plugged into the specified expression and the result stored back in 2CV. Finally a span (**S9**, which must have been previously defined) is applied and the result is returned with appropriate name and units.

An arithmetic operator may also be applied during the assignment, using the **+=*n*CV**, **-=*n*CV**, ***=*n*CV** and **/=*n*CV** channel options.

These allow a CV to be used as an accumulator, eg

```
RA1M 3C(+=2CV) 2CV("Total")
```

will report the number of counts received in each one minute period, plus the total counts, ie:

```
3C 192 Counts
Total 192

3C 77 Counts
Total 269
```

Note also that

```
1V(/=1CV)
```

is equivalent to

```
1V(=2CV) 1CV(W)=1CV/2CV
```

Integer Values

Channel variables can be used to store **integers** (eg. counter values, data read from the serial sensor port, iteration counters) but note that only 24 bits of precision is available. This means that integer values whose absolute value is greater than 2^{24} (16,777,216) may be rounded.

A consequence of this is that if you manually count something using a channel variable (eg. **1CV=1CV+1**) then it will stop counting once its value reaches 16,777,216. (Note that this only applies to manual counting using CVs – hardware and software counters (**HSC** and **C** channel types) can count over the full 32-bit range.)

If you need a CV to count beyond 16,777,216 then you will need to use two CVs to hold the count value, eg:

```
1CV=1CV+1 IF(1CV>1000000){2CV=2CV+1 1CV=0}
```

Naming Channel Variables

As with any other channel type, **CV** channels can be given name and units strings using the **"***CVname*~*Units*"** channel option.

The command

```
NAMEDCVS
```

will return a summary of all CVs that have been explicitly named, eg:

```
 CV S  CV Name                Value  Units
=========================================
  5 A  Temp                   89.1  Deg C
  1 A  Speed                  23.4  m/s
```

(The "S" column is the schedule identifier)

Expressions

The *DT80* has a powerful expression evaluation capability. Results can be assigned to channel variables, output channels, system timers and system variables.

Expressions can ONLY contain channel variables and constants. Data from input channels must first be assigned to channel variables to be used in expressions.

Expressions can contain the following elements:

Type	Elements
Arithmetic operators	**+**, **–**, *****, /, **%** (modulus) and **^** (exponent)
Relational operators	**<**, **<=**, **=**, **!=**, **>=**, **>** (result is 0 (false) or 1 (true))
Logical operators	**AND**, **OR**, **XOR**, **NOT** (0 is false, non-zero is true; result is 0 or 1)
Functions	**ABS()**, **LOG()**, **LN()**, **SIN()**, **COS()**, **TAN()**, **ASIN()**, **ACOS()**, **ATAN()**, **SQRT()**, **S*n*(), Y*n*(), F*n*()**
Other	Round brackets (parentheses) **()**

Note: The trigonometric functions require arguments in radians, where 1 radian = 57.296 degrees.

Note: The modulus operator (**%**) converts both operands to integer.

Operators are applied in the following order

Order		
1st	**^**	
2nd	*****, **/** or **%**	← These operators have equal precedence
3rd	**+** or **-**	← These operators have equal precedence
4th	**<**, **<=**, **=**, **>=**, **>**	← These operators have equal precedence
5th	**AND**, **OR**, **XOR** or **NOT**	← These operators have equal precedence

Operators with equal precedence are evaluated left to right.

Parentheses can be used to alter the order of evaluation.

For example:

```
4CV=1.5+2*3^2      '4CV = 19.5
4CV=(1.5+2)*3^2    '4CV = 31.5
4CV=((1.5+2)*3)^2  '4CV = 110.25
```

Boolean logic within expressions can be used to return a result that is dependent on a condition being true or false, eg

```
2CV=(1CV*2*(1CV<1000))+(1CV*4*(1CV>=1000))
```

returns a value of **1CV*2** if 1CV is less than 1000, or a value of **1CV*4** if 1CV is greater than or equal to 1000.

Combining Methods

The different scaling and calculation methods can be used together. The following comprehensive examples are the best way to demonstrate.

Example 1

In this program, a vector average is calculated. The inputs are wind speed and direction.

```
BEGIN"Wind-01"

'Wind speed calibration 0-50m/s = 0-1000mV
S1=0,50,0,1000"m/s"
'Wind direction 0-2Pi radians (0-360deg) = 0-1000mV
S2=0,6.2832,0,1000"radians"
Y3=0,1"m/s" 'Units text for wind speed report
Y4=0,1"Deg" 'Units text for wind direction report

RA5S 'Schedule to scan every 5 seconds
 1V(S1,=1CV,W) 'Sample wind speed
 2V(S2,=2CV,W) 'Sample wind direction
 3CV(W)=3CV+1CV*COS(2CV) 'Sum x components
 4CV(W)=4CV+1CV*SIN(2CV) 'Sum y components
 5CV(W)=5CV+1.0 'Number of scans

RB1M 'Calculate, report and log every minute
 'Calculate mean magnitude:
 6CV(W)=SQRT((3CV*3CV)+(4CV*4CV))/5CV
 6CV("Mean Wind Magnitude",Y3,FF1)
 'Calculate direction
 7CV(W)=ATAN(4CV/3CV)*57.29
 'Determine direction quadrant
 7CV(W)=7CV+((3CV>0)AND(4CV<0))*360
 7CV(W)=7CV+((3CV<0)AND(4CV<0))*180
 7CV(W)=7CV+((3CV<0)AND(4CV>0))*180
 'If wind speed is zero, return -1.0:
 7CV(W)=7CV-(6CV<=0)*(7CV+1)
 7CV("Mean Wind Direction",Y4,FF0)
 1..5CV(W)=0

LOGON
END
```

Example 2

This program scans ten channels and calculates a cross-channel average.

```
BEGIN"Wind-02"
RA10S
 1CV(W)=0 'Clear 1CV
 1..10V(+=1CV,W) 'Sum 10 voltages into 1CV
 1CV=1CV/10 'Divide by 10 for average
END
```

Derived Quantities

The *DT80* can automatically compute various commonly used **derived quantities** such as differences, rates of change, pulse widths and so on. These are calculated by including the appropriate channel option, as detailed below.

In each case the derived quantity is returned instead of the original reading.

Rates and Integrals

The following derived quantities are calculated based on the current and the previous channel reading.

Channel Option	Description	Formula
DF	Difference	Δx
DT	Time difference	Δt
RC	Rate of change	Δx / Δt
RS	Reading / time difference	x / Δt
IB	Integrate	(x – Δx / 2) Δt

The **DF** channel option returns the **difference** between the current and previous measurements; the **DT** option returns the **time difference** and the **RC** channel option combines the two to return the **rate of change** (Δx / Δt).

For example,

```
RA1S 1V(=1CV) 1CV(DF,"DeltaV") 1CV(DT,"DeltaT") 1CV(RC,"RC~mV/s")
1V 29.4 mV
DeltaV 3.9
DeltaT 00:00:00.992 0
RC 4.0 mV/s
```

The **RS** option is similar to **RC** except that the numerator is the actual reading, not the difference. This is intended for use with channels where the reading is already a difference. In the following example the counter is reset after each reading (using the **R** channel option), so the count reading is actually the number of counts since the last reading, so to calculate counts per second the **RS** option is used:

```
RA10S 3C(R,RS,"~counts/s")
```

Finally, the **IB** option is used to **integrate** a signal. It returns the area under a straight line connecting the current to the previous reading. For example:

```
RA20S 1V(=1CV,W) 2#I(=2CV,W)
  3CV(W,=4CV)=1CV*2CV
  4CV(IB,W,+=5CV)
  5CV("Energy~J")
```

The above example will, every 20 seconds, first measure a voltage and a current and assign them to two channel variables. These are then multiplied to give the instantaneous power in Watts (3CV), then integrated to give the energy used over the 20 second period (4CV). Finally, the energy values are accumulated in channel variable 5CV to give the total energy used.

Note A channel containing one of the rate/integral options should normally appear once only in a schedule, otherwise you may not get the result you expect, eg:

```
RA10S 1V("V") 1V(DF,"deltaV)  'incorrect
```

will generally always report very small difference value because the "previous" sample was taken just moments before, as part of the same schedule iteration. If you want to return both the current value and the delta, use a channel variable, eg:

```
RA10S 1V(=1CV,"V") 1CV(DF,"deltaV~mV")
```

which will return the expected result because each channel is only being evaluated once per schedule iteration.

Similarly, the energy example described above would not work properly if you tried to use:

```
3CV(W)=1CV*2CV  3CV(IB,W,+=5CV)   'incorrect
```

because the **3CV** channel (which uses the **IB** option) appears twice in the schedule.

Edge Timing

A number of channel options are provided for reporting details relating to the timing of digital transitions (edges).

As with the rate and integral options, these derived quantities are calculated based solely on the current and the previous channel readings.

The **TOR** and **TOF** options return the absolute date/time at which a last rising or falling edge occurred. If no edge has occurred since the last reading then a "zero" date/time value (normally presented as `00:00:00.000,01/01/1989`) is returned. For example:

```
RA1-E 1DS(TOF,"-Edge at")  RB1+E 1DS(W)
-Edge at 12:42:05.000,25/12/2006
```

```
-Edge at 12:44:37.060,25/12/2006
```

In the above example the A schedule runs when a falling edge on digital input 1 occurs, and reports the time at which the edge occurred. The B schedule runs on rising edges and evaluates (but does not return) the **1DS** channel. This is necessary because otherwise the **1DS** channel in schedule A would not see any change (since every time schedule A runs, digital input 1 will be low).

The following options report the time interval between two edges:

- If a rising edge has occurred since the last reading on a channel, and another rising edge has occurred some time previously, then the **TRR** option returns the time interval between the two edges, otherwise it returns zero.
- If a rising edge has occurred since the last reading on a channel, and a falling edge has occurred some time previously, then the **TFR** option returns the time interval between the two edges, otherwise it returns zero.
- If a falling edge has occurred since the last reading on a channel, and another falling edge has occurred some time previously, then the **TFF** option returns the time interval between the two edges, otherwise it returns zero.
- If a falling edge has occurred since the last reading on a channel, and a rising edge has occurred some time previously, then the **TRF** option returns the time interval between the two edges, otherwise it returns zero.

The following example reports the period on each rising edge:

```
RA1+E 1DS(TRR,"Period")  RB1-E 1DS(W)
Period 00:00:17.006

Period 00:00:14.000
```

Statistical Channel Options

Overview

It is often convenient to sample channels frequently and a return and/or log a **statistical summary** at longer intervals (see Statistical Report Schedules (P50)). Statistical channels are sampled during the period between report times (at a rate governed by the statistical schedule, **RS**), and the statistical summary is generated and returned at report time, ie. when the regular schedule runs.

Channels that require statistical sampling must include a channel option to indicate the statistical information to generate. Here's a summary of the statistical channel options — see also the Statistical (P40) category in the *Table 3: DT80 Channel Options* (P41) table:

Channel Option	Description	Appended to Units
AV	Average	(Ave)
SD	Standard deviation	(SD)
MX	Maximum	(Max)
MN	Minimum	(Min)
TMX	Time of maximum	(Tmx)
TMN	Time of minimum	(Tmn)
DMX	Date of maximum	(Dmx)
DMN	Date of minimum	(Dmn)
IMX	Instant of maximum (combines DMX and TMX)	(Imx)
IMN	Instant of minimum (combines DMN and TMN)	(Imn)
INT	Integral	(Int)
NUM	Number of samples	(Num)

The statistical option is defined by including it as a channel option in parentheses after the channel type. For example:

```
RA1M 3TT(AV)(NUM)
3TT 103.7 degC (Ave)
3TT 42 (Num)

3TT 110.2 degC (Ave)
3TT 60 (Num)
```

In this case you will see the **Sample** LED flash once per second (which is the default rate for **RS**), but data will only be returned once per minute. These data consist of average of the samples taken since the A schedule last ran, and the number of samples (which will normally be 60). Note that a tag, eg (Ave), is attached to the units to indicate the statistical function that has been applied.

Note There may be fewer than expected samples in the first sample period after starting a schedule. This is because, by default, schedule execution is synchronised to midnight (see *Time Triggers — Synchronizing to Midnight* (P52)) so a one minute schedule will always execute on minute boundaries.

If insufficient statistical samples have been taken at the time when the report schedule runs then an error message returned and the data value is flagged as "not yet set" (displayed as -9000000000). The **SD** and **INT** options require a minimum of two samples, the others require at least one sample. To minimise the chance of this condition occurring:

- switch off the synchronise to midnight flag (**/S**), or
- ensure that the reporting schedule period is substantially longer than the statistical schedule period.

Statistical Functions

Average (AV)

The average or mean is the sum of all the channel readings divided by the number of readings. It is very useful in reducing sensor noise.

Standard Deviation (SD)

Standard deviation is a measure of the variability of the data about the average or mean. The variation may be due to electrical noise or process changes. The units of standard deviation are the same as the channel reading.

Maximum and Minimum (MX and MN)

The maximum and minimum of a set of channel readings can be reported with the **MX** and **MN** channel options.

The time at which these occurred can be reported with the **TMX** and **TMN** options, the date with **DMX** and **DMN**, and the combined date/time ("instant") with the **IMX** and **IMN** channel options.

For example:

```
RS1M  RA30M 1TK(AV)  RB1D 1TK(MX)(TMX)(MN)(TMN)
1TK 24.2 degC

1TK 21.9 degC

1TK 19.0 degC
1TK 33.9 degC (Max)
1TK 15:10:00.000 (Tmx)
1TK 12.9 degC (Min)
1TK 04:33:00.000 (Tmn)
```

The above job measures the temperature once a minute (**RS1M**). Every 30 minutes the average for the 30 minute period is returned by the A schedule. Once a day (at midnight), the daily min/max temperatures are returned, along with the times at which they occurred.

Integration (INT)

The integration channel option returns the integral (area under the curve) with respect to time in seconds using a trapezoidal approximation. The units of an integration are those of the original reading multiplied by seconds.

In the following example a sensor returns a voltage that is proportional to the flow rate (0-1000mV = 0-0.2 l/s):

```
BEGIN
RS100T
S5=0,0.2,0,1000"litres"
1CV=0
RA2S 1V(S5,INT,+=1CV,W) 1CV("Fuel Used",FF3)
END
Fuel used 0.012 litres

Fuel used 0.104 litres
```

Every 100ms, the voltage output from the sensor is measured, scaled by span **S5** (yielding a value in litre/s) and the integral is progressively accumulated (yielding a value in litres). This is then accumulated in **1CV** (yielding the total number of litres used since the schedule started), which is reported every 2 seconds.

Note the differences between the **INT** and **IB** options (both of which calculate integrals):

- The **IB** option uses two points only (the current value and the previous value) and calculates the area under the curve using a single trapezoid. It does not involve the statistical schedule.
- **INT** is a statistical option. It calculates the integral using a trapezoid for each sample point measured by the statistical schedule.

Multi Value Statistical Options

The statistical options described here are special in that they return **multiple values**. A channel can only have one return value, so these options work by setting channel variables.

These channel options do not affect the usual reporting or logging of the channel's readings.

Histogram (*Hx:y:m..nCV*)

The *DT80* can be used to generate a histogram (frequency distribution) of channel samples by applying the histogram channel option, which instructs the *DT80* to

- divide the measured data range into a number of intervals called **classes**
- count the number of readings that occur in each class during the histogram period
- load each class count into a separate channel variable.

Then use another schedule to read, log and clear the channel variables.

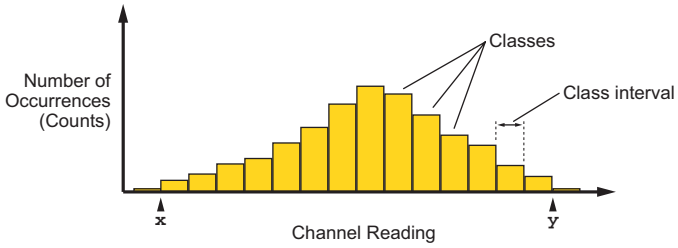


Figure 20: Histogram

In addition, the *DT80* automatically counts the number of under-range, over-range and total readings, and stores these in three separate channel variables.

The format of the histogram channel option is:

H*x*:*y*:*m*..*n*CV

where:

- *x* and *y* are the lowest and highest channel readings of interest, as shown in the above diagram
- *m* and *n* denote the range of channel variables (*m***CV** to *n***CV** inclusive) to use for storing count values.

The channel variables are set as follows:

Channel Variable	Function
*m***CV**	number of readings in the lowest class
...	
*(n-3)***CV**	number of readings in the highest class
*(n-2)***CV**	number of readings under range (<*x*)
*(n-1)***CV**	number of readings over range (>*y*)
*n***CV**	total number of readings including those out of range

The range *x..y* will therefore be broken up into (*n* – *m* – 3) classes.

Example — Histogram

To create a histogram of a temperature channel over five classes requires eight channel variables:

```
BEGIN"HISTO"
11..18CV=0
RA1S 1TT(H25.0:35.0:11..18CV)
RBX 16CV("Under") 11..15CV 17CV("Over") 18CV("Total")
RCX 11..18CV=0
END
```

The A schedule will report the temperature once a second. It will also accumulate a histogram with five temperature classes and intervals of 2°C:

- **11CV** counts readings in the first class (25.0 to 26.999°C interval)
- **12CV** counts readings in the second class (27.0 to 28.999°C interval)

- **13CV** counts readings in the third class (29.0 to 30.999°C interval)
- **14CV** counts readings in the fourth class (31.0 to 32.999°C interval)
- **15CV** counts readings in the fifth class (33.0 to 34.999°C interval)

The following three CVs will also be updated:

- **16CV** is the number of under-range samples (<25°C)
- **17CV** is the number of over-range samples (>35°C)
- **18CV** is the total number of samples (sum of 11..17CV)

The B schedule will, when polled, report the current histogram data. Polling the C schedule will clear the histogram data. A typical histogram report would look like:

```
XB
Under 7
11CV 19
12CV 33
13CV 102
14CV 71
15CV 22
Over 2
Total 246
```

Rainflow Cycle Counting

Rainflow cycle counting (also called **rainflow analysis**) is an internationally-accepted method of fatigue cycle counting used for monitoring long-term accumulative structural fatigue damage. The process reduces large quantities of cyclic data — collected from sensors attached to the structure over a long period of time — into relatively simple histograms.

As a structure deflects due to repetitive external influences, measurements produce arbitrary peak and valley sequences that form closed loops or cycles. Each loop or cycle has a size (the difference between peak and valley magnitudes), and rainflow analysis accumulates a profile of the number of cycles versus cycle size into a histogram.

A minimum cycle size can be defined that sets a noise rejection level, and cycle sizes below this level are rejected as noise and are not counted.

The *DT80* implements the ASTM E 1049-85 standard: Standard Practices for Cycle Counting in Fatigue Analysis.

Real-time rainflow analysis can be carried out using the *DT80*'s **RAINFLOW** channel option, which instructs the *DT80* to monitor attached strain gauges at regular intervals and reduce the resulting large quantity of data into simple cycle histograms.

The *DT80* can also produce a formatted report of the accumulated cycle histograms — see Reporting Rainflow Data *(P69)*.

Although the rainflow cycle counting has been optimized for welded steel structures, it can be used to record arbitrary waveforms from other sources — temperature cycles in a furnace or electrical signals, for example.

Collecting Rainflow Data

Rainflow analysis is defined by the **RAINFLOW** channel option. Although this is generally used for channels measuring strain gauge inputs, you can also use it for any type of sensor that is monitoring a process that produces cycles of peaks and valleys with hysteresis.

The overall range of cycle sizes is divided into a number of smaller cycle size **classes** and, as the analysis proceeds, the number of cycles of each size class is counted. These counts are accumulated into the *DT80*'s 32-bit signed **Integer Variables** (channel type *n***IV**).

(These integer variables are only for use with rainflow analysis.)

The **RAINFLOW** channel option requires a maximum cycle size to be specified, a noise rejection level, and a range of sequential integer variables or channel variables that can be used for accumulating the cycle size counts and other information. It has the form:

RAINFLOW:*a*:*b*:*c*..*d***IV**

where:

- *a* is the **maximum cycle size** expressed in the channel type units (for example, ppm)
- *b* is the **minimum cycle size** for noise rejection expressed as a percentage of *a*
- *c* and *d* denote the range of integer variables (*c***IV** to *d***IV** inclusive) to use for storing count values.

Therefore the range of cycle sizes is from zero to the maximum cycle size defined (*a*), and cycle sizes smaller than *b*% of *a* are rejected and not counted. For example, the channel option

(RAINFLOW:1000:5:*c*..*d***IV)**

sets the cycle size range to 0–1000 units, and cycle sizes less than 50 (5% of 1000) units are rejected as noise.

The number of variables allocated for the rainflow analysis must be set to the number of cycle size classes required over the cycle size range, plus seven (7) additional variables for summary data. For example, if you require 10 cycle size classes over the cycle size range then 17 variables will be needed. The variables can begin at any number in their range of 1 to 500 (*c*), and are used sequentially to the last variable number (*d*).

The use of variables in the allocated variable range is summarized in the following table. The first column shows how variables are used within the allocated range, and the last column shows how 20 variables are used. The last 7 variables contain various summary data.

	c..*d*IV	IV Contents	Example: 21..40IV
	c+0	Contains the count of cycles for the first cycle size class	21IV
	c+1	Contains the count of cycles for the second cycle size class	22IV
	c+2	Contains the count of cycles for the third cycle size class	23IV
	⇓	⇓	⇓
	d–7	Contains the count of cycles for the last cycle size class	33IV
Summary data	*d*–6	Contains the count of cycles that over-ranged the maximum cycle size	34IV
	d–5	Contains the count of all cycles	35IV
	d–4	Contains the maximum buffered cycles 0..100 (or 99999 if the buffer has overflowed and buffered half-cycles have been lost)	36IV
	d–3	Contains the minimum valley encountered	37IV
	d–2	Contains the maximum peak encountered	38IV
	d–1	Contains the total number of good points	39IV
	d–0	Contains the total number of "in error" points (out of range, for example)	40IV

In practice, some cycles do not close immediately and are buffered until a closure is detected. Variable *d*–4 contains a count of these unclosed or "half cycles".

Note: The rainflow channel option can be used on a maximum of 16 channels.

Rainflow cycle data is collected at a rate dependent on the frequency of influences deforming the structure under test. These might be quite slow events (such as waves crashing against a sea wall), or quite fast (such as a high speed boat hull travelling through waves).

Place the channel being sampled for rainflow in a schedule that's triggered fast enough to take sufficient readings during a cycle to adequately characterise the loop closures. For example, the schedule

```
RA50T 3BGI(RAINFLOW:a:b:c..dIV,W)
```

measures the input every 50ms (20 times/sec), and counts loop closures. The **W** channel option declares this as a working channel (does not return or log the individual samples of strain-stress.

Reporting Rainflow Data

Rainflow data is collected over long periods of time using the **RAINFLOW** channel option. Then, periodically, the rainflow cycle histogram can be retrieved by a computer, using the **RAINFLOW** command.

To report the rainflow cycle histogram, send the original rainflow channel option exactly as originally defined for the channel, but as a command. That is, send the command:

```
RAINFLOW:a:b:c..dIV
```

The *DT80* returns a tabular report as illustrated below:

```
RAINFLOW:72:5:1..27IV
Rainflow ( 5% rejection)   01/01/2000 00:03:43
n    IV/CV  Range       Mean        Cycles
=============================================
  1      1          0.0         0.0          0
  2      2          3.6        11.4         27
  3      3          7.2        11.3          6
  4      4         10.8        12.4          6
  5      5         14.4        11.9          6
  6      6         18.0        12.8          9
  7      7         21.6        12.3          2
  8      8         25.2         0.0          0
  9      9         28.8         0.0          0
 10     10         32.4         0.0          0
 11     11         36.0        18.0          1
 12     12         39.6         0.0          0
 13     13         43.2         0.0          0
 14     14         46.8         0.0          0
 15     15         50.4         0.0          0
 16     16         54.0         0.0          0
 17     17         57.6         0.0          0
 18     18         61.2         0.0          0
 19     19         64.8         0.0          0
 20     20         68.4         0.0          0
 21     21  >=     72.0         0.0          0
```

```
================================================
Total cycles                                    58
Peak/Valley mean                              12.6
Max Peak                                        71
Min Valley                                      -1
Max buffered cycles                             11
Valid input points %                        100.00
```

The rainflow report provides a complete summary of the rainflow data for the collection period. The cycle size range for each class, the number of cycles in each class, and the mean for each class is shown, as well as the summary data.

Although the rainflow report cannot be logged in the *DT80*, the primary cycle count data used to make up the rainflow report can. For example, the program

```
BEGIN"Rainflow"
RA50T 2BGI(RAINFLOW:72:5:1..27IV,W)
RB7D 1..27IV
LOGONB
END
```

logs the histogram data every 7 days. Reports can be created manually after download of the primary cycle count data.

Example — Rainflow Cycle Counting

Capture raw strain gauge data and perform rainflow cycle analysis using the program

```
BEGIN
RA50T 1BGI(RAINFLOW:1000:5:101..127IV,W)
END
```

This instructs the *DT80* to

- collect current-excited bridge data (**1BGI**) every 50ms (**RA50T**) and carry out rainflow analysis over the range of zero to **1000** ppm
- apply a **5**% rejection (that is, cycles smaller than 50ppm are rejected)
- accumulate cycles into histogram variables 101 through 127 (**101..127**); this gives 20 cycle size classes for cycle counts, and 7 others for summary information.

The matching rainflow report command

```
RAINFLOW:1000:5:101..127IV
```

can then be used to return a summary report.

Part F — Alarms

Alarm Concepts

DT80 **alarms** allow decisions to be made based on the magnitude of *DT80* input channels, channel variables, timers, the clock/calendar, internal channels, system variables and so on. The decision is a **true** or **false** result of an alarm **condition test**. The true/false result is also known as the alarm **state**.

The *DT80* can be instructed to carry out actions when an alarm tests true. These actions can be setting the *DT80*'s digital state outputs, issuing messages, or executing commands to change the *DT80*'s operation.

Alarm transitions can also be logged to the *DT80*'s internal file system for later analysis.

There are two types of alarms:

- single shot alarms (**ALARM** command) act once on the transition of the condition test from false to true
- repeating alarms (**IF** and **DO** commands) act repeatedly each time the enclosing schedule runs, while the condition tests true

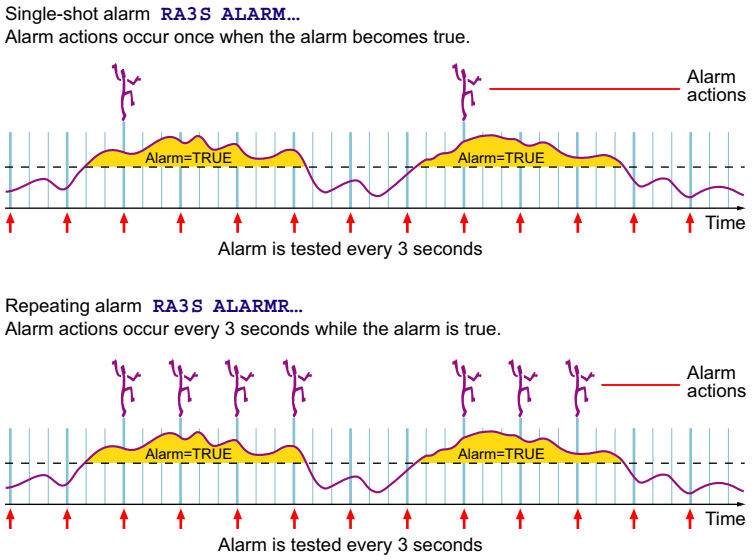


Figure 21: Comparing single-shot and repeating alarms (3-second schedule example)

Alarm commands can be included in any report schedule, and are processed in sequence with other schedule processes such as reading input channels and performing calculations.

Alarm Commands

The *DT80* provides three main alarm commands, each with a similar basic syntax:

ALARM*n* **(** *test* **)** *digitalAction* **"***actionText***" {** *actionProcesses* **}**

IF*n* **(** *test* **)** *digitalAction* **"***actionText***" {** *actionProcesses* **}** (**ALARMR** is also accepted as a synonym for **IF**.)

DO*n* **"***actionText***" {** *actionProcesses* **}**

where:

- ***n*** is the **alarm number**, used to distinguish logged alarms (optional)
- ***test*** is the **alarm condition** to test
- ***digitalAction*** is one or two digital output or CV channels which will follow the alarm state (optional)
- **"*actionText*"** is a text string to output if the alarm condition tests true (optional)
- **{*actionProcesses*}** is a set of channel definitions and/or commands to be executed if the alarm condition tests true (optional)

These are explained further in the following sections.

Note that the **DO** command is the same as **IF** except that the alarm condition is assumed to be always true.

Alarm Number

Alarm commands can optionally be given a number (a positive integer), which is used to identify the alarm when an alarm transition is logged.

For example:

```
RA1S ALARM1(2TT<15)"Too cold" ALARM2(2TT>30)"Too hot" LOGON
```

will test a temperature once per second. If the temperature dips below 15°C an alarm record indicating that alarm #1 has triggered will be logged to the *DT80*'s internal file system. See *Logging and Retrieving Data* *(P80)* for more details about the logging of data and alarms.

Alarm numbers need not be allocated in order, and they need not be unique (although normally they would be) – whatever you select as the alarm number will be what is logged if the alarm is triggered.

Note that unnumbered alarms, eg:

```
IF(1CV>10){9CV=9CV+1}
```

are not logged, even if logging is enabled for the schedule

Alarm Condition

The alarm condition compares a channel value to one or two **setpoints**. The state of the alarm is then set to true or false based on this test. (Note that in the case of the **DO** command, no alarm condition is specified – the state of the alarm is always true.)

Four different types of test are supported:

Condition	Alarm is true if
(*chan***<***setpoint***)**	channel value is less than *setpoint*
(*chan***>***setpoint***)**	channel value is greater than or equal to *setpoint*
(*chan***><***setpoint1,setpoint2***)**	channel value is greater than or equal to *setpoint1* AND less than *setpoint2* (that is, between the two setpoints)
(*chan***<>***setpoint1,setpoint2***)**	channel value is less than *setpoint1* OR greater than or equal to *setpoint2* (that is, outside the setpoint range)

where:

- ***chan*** is a standard channel definition, which will be evaluated (ie. measured) in the usual way. One set of channel options may be included if required.
- ***setpoint*** is a constant (eg. **2.77**) or channel variable specifier (eg. **13CV**)

In addition, a **time specifier** may be appended to the above. If a time specifier is present, the alarm will only be set true after the condition has been continuously true for the specified time.

A time specifier has one of the following forms:

Time Specifier	Condition must be continuously true for
/*n***S**	*n* seconds
/*n***M**	*n* minutes
/*n***H**	*n* hours
/*n***D**	*n* days

Some sample alarm conditions are shown below:

Condition	Alarm is true if
(2R(II)>51.5)	channel 2R value is greater than or equal to 51.5 ohms
(3+V(AV)<-200)	channel 3+V (averaged over schedule interval) is less than -200mV
(REFT<>-10,45)	temperature inside *DT80* is outside the range -10°C - +45°C
(T><9:00,17:30)	time is between 9am and 5:30pm
(32SV>10000)	10000 or more data records have been logged for schedule A
(1CV<2CV)	1CV is less than 2CV
(4DS<0.5)	digital input 4DS is low

Note that the channel that is tested in an alarm condition is always treated as a working channel, ie. its value is neither logged, returned nor displayed. If you want to record or output the channel's value each time it is tested then you need to assign the channel to a channel variable, then test the CV in the alarm condition, eg:

```
RA1S 2R(II,=1CV) ALARM(1CV>51.5)"R limit exceeded" LOGON
```

which will return and log the resistance measurement once per second, and return/log a message each time it exceeds the specified setpoint.

Complex Conditions

As indicated above, only relatively simple condition tests can be included in an alarm command. There are two ways to perform a more complex test.

Boolean Expressions

An arbitrarily complicated **boolean expression** can be constructed and assigned to a CV, which can in turn then be tested in an alarm command. For example:

```
RA5S
 1V(=1CV) 2V(=2CV) 3V(=3CV)
 9CV=((1CV>2.2)AND(1CV/2CV<=0.9)AND(3CV>=(1CV+2CV)))OR(3CV=0.0)
 ALARM(9CV>0.5)"Condition red"
```

Note that the syntax for boolean expressions is quite different to that used in alarm conditions.

In the above example 9CV will get the value 1.0 if the expression evaluates to true, otherwise 0.0

You can also use the result of a boolean expression in an **arithmetic expression**, making use of the fact that the value of the boolean expression is always 1.0 or 0.0. For example,

```
12CV=4CV*(1CV>2.5)+5CV*(1CV<+2.5)
```

will set 12CV to the value of 4CV if 1CV is greater than 2.5, otherwise it will set it to the value of 5CV.

See *Expressions* (P62) for more information about using expressions.

Combining Alarms

An alternative method of building up a complex alarm condition is to chain a number of consecutive alarm commands together. They are combined using logical operators (**AND**, **OR** or **XOR**), which replace the *digitalAction*, *actionText* and *actionProcesses* of all except the last alarm. The actions associated with the combined test are attached to the last alarm. Any alarm delay period is also associated with the last alarm.

For example, the combined alarm

```
ALARM(1*TK>100)OR
ALARM(1+TK>100)OR
ALARM(1-TK>100)AND
ALARM3(T>10:00:00)"Temp Error"{1DBO=12}
```

produces a single alarm output based on several temperature tests and a time test. The combined alarm becomes true when any one of **1*TK**, **1+TK** or **1-TK** exceeds 100°C after 10:00:00 am.

Alarm Digital Action Channels

One or two comma-separated **digital action channels** can be declared for each alarm. These channels will then mimic the state of the alarm. That is, these outputs are set to their default state if the alarm tests false, and are set to there non default state if the alarm tests true:

Digital action channels can be:

- digital outputs (*n***DSO**)
- **Attn** LED output (**1WARN**)
- latching relay output (**1RELAY**)
- channel variables (*n***CV**)

These will be set according to the following table:

Digital Action Channel	Value if Alarm FALSE	Value if Alarm TRUE
1..4DSO	1 (high/off)	0 (low/on)
5..8DSO	0 (low)	1 (high)
1WARN	0 (LED off)	1 (LED on)
1RELAY	0 (relay open)	1 (relay closed)
***n*CV**	0.0	1.0

Typically, the digital state outputs are used to annunciate the *DT80* alarm by switching devices such as relays, sirens and lights, or to directly control actuators and similar equipment.

For example,

```
RA1M ALARM(4TK<-1)4DSO"Heater on^M"
```

will check the temperature every minute. If it drops below -1°C a message will be output and digital output **4D** will go low, energising an external heater relay. The digital output will remain low (heater on) until the temperature (measured every minute) is no longer below -1°C.

Similarly,

```
RB10S ALARM(2+TC>2100)2CV,1RELAY"ReactorScram"
```

checks a temperature every 10s; if it exceeds 2100°C then the relay will close and 2CV will be set to 1.0 until the temperature drops back below the setpoint. Once this occurs the relay will open and 2CV will be set back to 0.0.

Alarm Action Text

Action text may be included in an alarm command. This text string is automatically returned to the host computer and/or logged to the internal file system:

- once whenever the state of a single-shot alarm (**ALARM**) goes from false to true, or
- repeatedly at the controlling schedule's rate while a repeating alarm (**IF** or **DO**) remains true.

Up to 200 characters of action text can be included in each alarm.

Note The action text may be truncated when it is logged. A fixed amount of space is allocated in the log file for storing the alarm text, which is set by the **ALARMS:W***n* (alarm width) schedule option; see *Schedule Options* (P43). If the default setting of **60** is used, each logged alarm string will be truncated to 60 characters.

Setting the alarm message switch (P132) to **/Z** stops the return of the action text to the host – in a similar way to the **NR** (no return) channel option.

Destination for Text

Action text is normally enclosed in **"**double quotes**"**, in which case it is returned to the host computer using the currently active communication port.

If, however, the action text is instead enclosed in **'**single quotes**'**, the *actionText* is sent exclusively to the RS232 Host Port on the logger. This is useful for communicating with modems when they are in command mode and when the host port is used for other purposes.

Substitution Characters

Special substitution characters can be placed into *actionText*. These instruct the *DT80* to dynamically insert the following information when the alarm returns and/or logs its action text:

Characters	Function	Example
!	Substitutes *DT80* serial number followed by a colon (:) and the alarm number	080035:8
?C or **?c**	Substitutes channel ID	2PT385
?N or **?n**	Substitutes user channel name	Boiler
?U or **?u**	Substitutes user channel units	degC
?V or **?v** or **?**	Substitutes the data value when the alarm tested true	100.1
@	Substitutes the time that the action text was returned (in P39 and P40 format)	12:13:14.634
#	Substitutes the date that the action text was returned (in P31 format)	11/2/2001
?R or **?r**	Substitutes relation	>50.0
??	Substitutes question mark	?
!!	Substitutes exclamation mark	!
@@	Substitutes @ symbol	@
##	Substitutes # symbol	#
?*n*	Substitutes the current value of the specified channel variable, where *n* is the number of the channel variable (1 to 800). For example, **?3** instructs the *DT80* to substitute the contents of 3CV into the alarm action text. You can also specify the format and number of decimal places (P41). For example: **?3F** inserts the value of 3CV in fixed-point format **?3E** inserts the value of 3CV in exponential format **?3M4** inserts the value of 3CV in mixed format with 4 significant digits	

Special Characters

Special characters may be inserted in alarm strings using control character (eg. **^M**) or backslash (eg. **\013**) notation. See *ASCII-Decimal Tables* (P192) for more information.

For example,

```
ALARM(3TT>120)"\192 hautes temp\233ratures!! ?v \176C^M^J"
```

will return/log the following string when the specified temperature is exceeded:

```
À hautes températures! 129.4 °C
```

In this example the **\192** and **\233** insert the accented characters, **\176** inserts the degree symbol, and **^M^J** adds a carriage return/line feed pair. Note also the **!!** to generate a single exclamation mark, and the **?v** substitution string, which is replaced by the channel value.

If the software used to enter the program text supports it, you could alternatively have entered the special characters directly,

ie.

```
ALARM(3TT>120)"À hautes températures!! ?v °C^M^J"
```

Be aware that many extended ASCII character codes display differently on the *DT80*'s LCD compared with the host computer. See *ASCII-Decimal* Tables (P192).

Alarm Records

If an alarm is triggered while fixed format mode (**/H**) is selected, a fixed format **alarm record** will be returned. This has a similar format to a fixed format data record (see *Format of Returned Data* (P23))

For example, the job:

```
BEGIN"B1" RB1S ALARM8(1V(S1)>1.5)"OverPressure ?vMPa^M^J" END
```

would, when triggered, return text similar to the following if normal (**/h**) mode was selected:

```
OverPressure 1.563MPa
```

In fixed format (**/H**) mode, however, it would return an alarm record:

```
A,080035,"B1",2006/04/16,14:32:01,0.254870,8;B,1,"OverPressure 1.563MPa^M^J";0078;3D95
```

An alarm record consists of:

- the usual fixed format header and trailer (serial number, job name, timestamp, error check fields)
- the alarm number (8)
- the schedule (B)
- the transition type (1 – false to true)
- the alarm text string (if any). Note that control characters (ASCII code < 32) are not output; they are left in the string in **^***x* notation.

Alarm records may also be **logged** to the *DT80*'s internal file system. This will occur if an alarm number is provided (eg. **ALARM7**), and logging is enabled for the enclosing schedule.

As with data, when logged alarm records are **unloaded**, they will be returned as fixed format records, as illustrated above.

Other Alarm Transitions

By default, an alarm record is only logged when an alarm is triggered, ie. its state changes from false to true. However, by setting parameter **P9=3** (see *Parameters* (P129)), the *DT80* will also log a record when the alarm goes inactive (true to false).

Also note that numbered **IF** and **DO** commands (which are seldom used) will log an alarm record every time their schedule executes, while their condition is true.

The following table summarises what is logged, and what is returned for the various types of alarm command.

Alarm Command	State/Transition	Parameter 9 (default P9=1)	Logged Transition Type, Alarm Text	Returned Alarm Text
ALARM(*test***)"***actionText***"**	False to true ↑	x	—	*actionText*
	Continuing true	x	—	—
	True to false ↓	x	—	—
ALARM*n***(***test***)"***actionText***"**	False to true ↑	P9=1 or 3	1, *actionText*	*actionText*
		P9=0 or 2	—	*actionText*
	Continuing true	x	—	—
	True to false ↓	P9=0 or 1	—	—
		P9=2 or 3	3, "ALARM*n* FALSE"	—
IF(*test***)"***actionText***"** (or **ALARMR**)	False to true ↑	x	—	*actionText*
	Continuing true	x	—	*actionText*
	True to false ↓	x	—	—
IF*n***(***test***)"***actionText***"** (or **ALARMR**)	False to true ↑	P9=1 or 3	1, *actionText*	*actionText*
		P9=0 or 2	—	*actionText*
	Continuing true	x	2, *actionText*	*actionText*
	True to false ↓	P9=0 or 1	—	—
		P9=2 or 3	3, "ALARM*n* FALSE"	—
DO"*actionText***"**	x	x	—	*actionText*
DO*n***"***actionText***"**	x	x	2, *actionText*	*actionText*

Examples

Text Labels

The **DO** command in conjunction with alarm text provides a simple way to output a text string in a schedule, eg:

```
RA5M DO"Boiler 1^M^J" 1TK 2TK DO"Boiler 2^M^J" 3TK 4TK DO"^M^J"
```

will include a heading before each group of measurements:

```
Boiler 1
1TK 239.4 degC
2TK 99.9 degC
Boiler 2
3TK 212.4 degC
4TK 90.9 degC
```

Modem Commands

The schedule command

```
RC2+E DELAY=500 DO"AT&F E0 Q1 S0=2 ^M"
```

instructs the *DT80* to output a modem initialization string 500ms after an external event occurs (**2+E**); for example, when the modem powers up.

Alarm Action Processes

Action processes can be any *DT80* functions to be executed when an alarm is true. These functions can be reading input channels, setting output channels, calculations, setting parameters and switches, and so on.

In addition, action processes are a very powerful programming facility for the *DT80*. Use them to perform a wide range of program-related functions such as re-programming on events, adaptive schedules (see examples below), programmed calibration cycles, management of digital state outputs, and management of the Serial Channel.

Action processes are also useful with unconditional alarm commands (**DO** commands) as a means of executing a *DT80* command (as opposed to a channel) within a schedule. See *Executing Commands in Schedules* (P52) for more details.

Action processes are placed within braces **{ }** as the last element in an alarm command. Each "process" is either:

- a channel definition (eg. **1+V(=1CV)** or **3DSO=0**), or
- a command (eg. **XC** or **P12=5** or **SATTN** or **LOGONA**), or
- a schedule trigger re-definition (eg. **RA100T**)

Alarm commands cannot be included.

Any number of processes may be included, but they must all be on the same line. Processes can be separated by semi-colon (**;**) or space characters.

Order of Execution

When an alarm is triggered, things happen in the following sequence:

1. Digital action channels (if any) are set to the required value
2. Alarm text (if any) is generated and returned/logged
3. Any channels in the action process list are evaluated, left to right. All channels in an alarm process list are treated as working channels – they are neither returned, logged nor displayed.
4. Any commands or schedule trigger re-definitions are queued for execution, working left to right.
5. Any further channels or alarm commands in the current schedule are executed
6. Any queued commands, including the ones generated by the alarm, are executed.

For example, the job

```
BEGIN
RA5S ALARM1(3TK>30){XB 1DSO=0 SATTN} 4V(NR)
RC1S 1V
RBX 1SERIAL("{boo!}")
LOGON
END
```

will perform as follows:

1. Schedules A & C will become due at the same time, because A's scan rate is an exact multiple of C's. A will run first, because, as noted in *Triggering and Schedule Order* (P51), A comes before C in the priority order.
2. Channel **3TK** exceeds 30 so the alarm is triggered. The alarm is numbered and logging is enabled so an alarm record will be logged, although the alarm text field will be an empty string.
3. Channel **1DSO** is evaluated – output **1D** is set low.

4. The command string **XB;SATTN;** is queued for execution.
5. Channel **4V** is evaluated. It's value is logged and displayed but not returned.
6. Schedule A is now finished; schedule C is selected to run next. It does not actually run, however, because there are queued commands to execute.
7. The **XB** command is now executed. This causes the B schedule to become due.
8. The **SATTN** command is executed, which turns on the **Attn** LED
9. There are no more queued commands so the C schedule can now run. Channel **1V** is evaluated and logged/displayed/returned.
10. Schedule B is also due, so it now runs. The **1SERIAL** channel is evaluated, which causes a string to be sent out the serial sensor port.
11. There is nothing further to do so the logger idles until schedule C next becomes due.

In most applications the ordering is not particularly important as all of the alarm actions occur within a very short space of time. However, it can cause surprises in some circumstances, as illustrated below.

Trap – Commands Don't Affect Channels in Same Schedule

Any commands executed in an action process list will not take effect until after all channels have been processed. For example, if you wanted some measurements to be returned in fixed format mode and some in free format, you might try:

```
RA1S DO{/H/R} 1V  DO{/h/R} 2V     ' does not work!
```

but in fact both channels will be returned in free format mode.

To achieve the desired result you need to do something like:

```
RA1S 1V DO{/H/R XB}
RBX 2V DO{/h/R}
```

In this example **1V** will be returned in free format mode, then we switch to fixed format mode, then we issue the command to poll schedule B. Schedule B will then do the same thing: return **2V** in fixed format mode, then switch back to free format mode so that the next time schedule A runs it will return its value in the correct format.

Note In the *DT80*, commands have higher priority than schedules. If there are any queued commands outstanding, they will be executed ahead of any schedules that happen to be due. However, once a schedule starts executing, it always runs to completion – any queued commands will be held off until the schedule completes.

Trap – Don't Use DELAY Between Commands

The **DELAY=***ms* function is a channel, not a command. It therefore cannot be used to insert a delay between two commands. For example, if you wanted to light the **Attn** LED for 5 seconds to indicate that a measurement was about to be taken, you might try

```
RA20M DO{SATTN; DELAY=5000; CATTN; XB} RBX 1V   ' does not work!
```

but this is no good because the **DELAY**, being a channel, is executed first, then **SATTN;CATTN;XB** in quick succession.

The **PAUSE** *ms* command does the same thing as **DELAY** except that it is a command, so you can use:

```
RA20M DO{SATTN; PAUSE 5000; CATTN; XB} RBX 1V  ' OK
```

(The semicolons between commands are optional in most cases. They are included in the above example because they make the program a little more readable, especially when commands with space-separated parameters are used.)

Note also that a simpler way to implement the above functionality would be to not use commands at all, eg:

```
RA20M 1WARN=1 DELAY=5000 1WARN=0 1V
```

or, even better:

```
RA20M 1WARN(5000,R)=1 1V
```

Examples

Controlling a System

Alarm action processes can be used to control a system or process. This is often preferable to the method used in the example in *Alarm Digital Action Channels* *(P73)* because it allows some hysteresis to be included.

For example,

```
RA1S
  ALARM(1TK<74.75)"Heater ON"{1DSO(W)=0}
  ALARM(1TK>75.25)"Heater OFF"{1DSO(W)=1}
```

is a simple heater control for a water bath. The two alarms work to hold the temperature at 75°C ± 0.25°C.

Adaptive Scheduling

Adaptive scheduling is the dynamic adjustment of the acquisition of data about a system or process as the system or process changes.

As the examples below show, adaptive scheduling can reduce total data volume while giving greater time resolution when required.

The schedule:

```
RA15M
  1V("Wind speed",S1,=1CV)
  ALARM(1CV>5.25){RA2M}
  ALARM(1CV<4.75){RA15M}
```

measures wind speed

- every 2 minutes if wind speed is greater than 5m/s, or
- every 15 minutes if wind speed is less that 5m/s

Note the deliberate 0.5m/s hysteresis to prevent oscillation around the switchover point. If the measured wind speed exceeds 5.25m/s, schedule A's trigger is re-defined to run every 2 minutes. When it drops below 4.75m/s it is reset back to every 15 minutes.

The following job:

```
RC30M 1TK("Oven Temp")
RD1M  ALARM(5TK>120){GA}  ALARM(5TK<110){HA}
LOGONC HC
```

continuously monitors the temperature of an oven and logs the temperature whenever it exceeds 120°C.

Initially the logging schedule (C) is halted (**HC**). Schedule D checks the temperature every minute, and when it exceeds 120°C schedule C is started (**GC**), and it is stopped again once it goes below 110°C.

Using an Alarm to Poll a Schedule

As mentioned above, if any channels are included in an action process list then they cannot be logged, returned or displayed. This limits the types of channels that can usefully be included in an action process list to:

- output channels (eg. **2DSO=0**)
- calculations (eg. **1CV=1CV+1**)
- channels that assign to a CV (eg. **2*V(=2CV)**)

If you need to conditionally take measurements and log/return them, you will need to set up a separate schedule and then use the alarm to poll it.

For example, the job:

```
BEGIN
1..3CV(W)=0
RA1S
 1CV(W)=1CV+1
 ALARM(1CV>3CV){XB 2CV(W)=2CV+1 3CV(W)=2^3CV}
RBX LOGONB
 1..5TK
END
```

logs data at increasing intervals as the experiment proceeds. The program calculates the next log point as an incrementing power of 2 seconds — that is, it logs the temperatures at t = 0, 1, 2, 4, 8, 16,… seconds. The following table lists the values of the three CVs at the point at which the **ALARM** statement is executed.

Time (s)	1CV	2CV	3CV	
0	1	0	0	Alarm active – B schedule polled
1	2	1	2	Alarm active – B schedule polled
2	3	2	4	
3	4	2	4	Alarm active – B schedule polled
4	5	3	8	
5	6	3	8	
6	7	3	8	
7	8	3	8	Alarm active – B schedule polled
8	9	4	16	
				etc.

(Remember that **(1CV>3CV)** means 1CV is greater than or equal to 3CV.)

The following example will log all voltage readings that exceed 200mV:

```
BEGIN
RA1S IF(2V(=1CV)>200){X}
RX 1CV("Vout~mV")
LOGONX
```

```
END
```

Note that assigning to a CV in this way and then reporting the CV value is preferable to including **2V** in both schedules.

Executing Commands in Schedules

The following will output a directory listing every time a positive edge is received on digital input 7:

```
RA7+E DO{DIR"B:"}
```

Selecting a Job to Run

The following schedule

```
RA1SERIAL""
 1SERIAL("%1d",=1CV)
 IF(1CV><0.5,1.5){RUNJOB"MIX"}
 IF(1CV><1.5,2.5){RUNJOB"CHURN"}
 IF(1CV><2.5,3.5){RUNJOB"GRIND"}
```

will run when a character is received on the serial sensor port. If the character is **1**, **2** or **3** then the indicated job will be loaded and run, replacing the current job.

Automatic Data Archive

The schedule command

```
RE1D DO{MOVEDATA"Job1"A}
```

instructs the *DT80* to — every midnight (**1D** trigger) — move all logged data for schedule A of Job1 to an archive file on the USB memory device.

Polling Alarm Inputs

The current values of the channels being tested in alarm conditions can be polled (requested) by the host computer at any time. There are three commands for polling alarm data:

Command	Function	
?*n*	returns the current input value of alarm *n*	For numbered alarms only
?*x*	returns the current input values of all alarms in schedule *x*, where *x* = **A**, **B**,…**K**, **X**	For numbered and un-numbered alarms
?ALL	returns the current input values of all alarms in all schedules	

The output of each of these commands consists of:

- the alarm number (A*n*). When un-numbered alarms are polled, the alarm number is returned as A0
- the alarm condition (so that un-numbered alarms can be distinguished)
- the current value of the channel being tested in the alarm condition

For example

```
BEGIN
RA2S
 ALARM4(2R>50)"High R"
 ALARM(1CV<>-10,10){2CV=2CV+1}
 IF(2CV>4){1V(=9CV)}
 ALARM5(9CV<100){RA100T}
END

?ALL
A4 4R>50 1300.6
A0 1CV<>-10,10 99.0
A0 2CV>4 102.0
A5 9CV<100 0.0
```

If the *DT80* is set to formatted mode (**/H**) then formatted mode records containing the same information are returned.

Part G — Logging and Retrieving Data

Logging Data

By default, the *DT80* returns measurement data to a host computer in real time. However, the *DT80* can also automatically record each reading taken for some or all of a schedule's channels. These data are stored in the *DT80*'s internal memory, and can be retrieved at a later date, using a USB memory device or via one of the communications ports.

Each reading is automatically timestamped.

Logged data is retained in the internal memory until it is explicitly cleared, even if the *DT80* is reset or loses power.

Enabling and Disabling Data Logging

LOGON and LOGOFF Commands

By default, data logging is disabled when a schedule is entered. The following commands switch logging on or off. They may be entered as part of a job, or they may be sent at any time after a job has started running:

Command	Function
LOGON	Enables logging (data and alarms) for all schedules.
LOGOFF	Disables logging (data and alarms) for all schedules.
LOGON*sched*	Enables logging for schedule *sched* (data and alarms)
LOGOFF*sched*	Disables logging for schedule *sched* (data and alarms)

For example the following job defines a schedule and enables logging:

```
BEGIN"LUMPY" RA2M 2V 3V LOGONA END
```

This will create a store file with the default size. Every two minutes, two voltages will be measured and the results will be stored, along with the time at which the measurements were taken.

Disabling Data Logging for Specific Channels

If logging is enabled for a schedule then by default all channels defined therein will be logged. To disable logging for specific channels:

- use the **NL** (no log) channel option, or
- use the **W** channel option (working channel; do not log, return or display)

How Data and Alarms are Stored

The DT80 File System

The *DT80*'s internal flash memory is organised as a DOS-compatible file system, which uses files and directories in a similar way to a desktop computer. When a USB memory device is inserted it is treated in a similar way. Note that:

- the USB memory device, if present, is referred to as drive **A:**
- the *DT80*'s internal file system is referred to as drive **B:**

(The **DIR** and **DIRTREE** commands can be used to explore the contents of either drive. For example **DIRTREE"B:"** will list the names of all files and directories on the internal file system.)

The standard internal file system has a capacity of 64Mbyte. The *DT80* stores approximately 90,000 data values per megabyte of memory, so the internal memory can hold approximately 5,000,000 data values.

Store Files

When a job is first entered, a directory (folder) is created on the internal file system: **B:\JOBS***jobname*. This directory contains files which record the job's program text and other details about the job.

If logging is then enabled for one of the job's schedules, a data storage sub-directory for that schedule is created: **B:\JOBS***jobname**sched*.

Finally, a **store file** is created in the schedules data storage directory. Note the following important points:

- A store file is a **pre-allocated, fixed size** file. The size of the file (as returned by the **DIR** command, for example) does not change are data is stored.
- A store file contains two fixed size sections – one for **data** (measured channel values), one for **alarms** (text strings which are logged when a particular condition is true). The sizes of these sections is configurable on a per-schedule basis.
- A store file is a **binary** file. The data contained in it are not directly human-readable. See *Retrieving Logged Data* (P84).
- A store file has a name of the form **DATA**_*sched*.**DBD**.

For example, the job:

```
BEGIN"BUMPY"  RA2M 2V 3V   RB1S 2DS   RK20S 1V   LOGONA LOGONK  END
```

would create the following store files:

```
B:\JOBS\BUMPY\A\DATA_A.DBD
B:\JOBS\BUMPY\K\DATA_K.DBD
```

Note that initially no file would be created for schedule B because logging is not enabled for that schedule. If at some later time you entered the **LOGONB** command, then a store file would be created.

In the above example, the store files would all have the same, default size (approx. 1Mbyte).

How Much Data Can I Store?

Each time a schedule executes (assuming logging is enabled for the schedule), it writes one **data record** to its store file. A data record consists of the values of all channels defined in the schedule, other than those for which logging has been disabled (using the **NL** or **W** channel options).

As a rule of thumb, one data record uses **10 + (10 x numberOfLoggedChannels)** bytes, assuming "normal" channel types (time/date channels and $ strings require more space)

So for the schedule:

```
RA1S 1V 2CV(NL) 3TK
```

each data record will use 30 bytes, so the default 1Mbyte allocation for data is enough for 1,048,576 / 30 = 34,952 data records. The store file will therefore contain the most recent 9 hours or so of readings, assuming a 1 second scan rate.

How Many Alarms Can I Store?

Normally, one alarm record is logged each time a numbered alarm is triggered, ie. its state goes from false to true. However, as discussed in *Other Alarm Transitions* (P75), the true-to-false transition may optionally also be logged, and numbered **IF** and **DO** alarm commands may log a record each time their schedule executes while their condition is true.

As a rule of thumb, one alarm record uses **12 + alarmWidth** bytes, where **alarmWidth** is set using the **ALARMS:W***n* schedule option; see *Schedule Options* (P43). So assuming the default setting of **60** is used, each alarm record will use 72 bytes. The default 100kbyte allocation will therefore store 102,400 / 72 = 1,422 alarm records.

Logging Options

Various data logging parameters can be changed by means of **schedule options**. These options are inserted in a schedule definition just before the schedule trigger. See *Schedule Options* (P43) for details.

Schedule options can be used to specify:

- the logging destination. Using the **"A:"** schedule option it is possible to configure a schedule to data directly to a USB memory device. In this case the store file will be placed in the **A:\SN***serial-num***\JOBS***jobname******sched sub-directory.
- the space allocated for data. This can be specified in bytes, records, or (for time-based schedules) as a time period ("I want to store 30 days of data")
- the space allocated for alarms
- whether new data/alarm records are permitted to overwrite old records (**OV**) or whether logging should stop when the store file is full (**NOV**).

For example, the following schedule definition:

```
RA(DATA:30D)1M 1V 2V LOGONA
```

will allocate a store file with space for 43,200 data records (30x24x60). No space is allocated for alarms because no alarms are defined in this schedule.

Note It is normally better to always log data to the internal file system. Logging directly to a USB device is possible, but is subject to the following caveats:

- The logging performance is significantly slower than for the internal drive.
- There is the potential for data corruption if the USB device is removed during a write operation. Be sure to always halt logging and use the **REMOVEMEDIA** command (see *Using a USB Memory Device* (P90)) to shut down the device prior to removing it.

- The *DT80*'s USB socket is designed for easy access, and will not necessarily retain a USB device securely over a long period, particularly if the *DT80* is wall mounted or subject to vibration.

Factors Which May Prevent Logging

Insufficient Space to Create Store File

When logging is first enabled for a schedule, the *DT80* creates a store file of the required size. If, however, there is insufficient free space on the selected logging drive then an error will be reported. A message will also be displayed on the LCD, and the **Attn** LED will start flashing. The schedule will still execute – channels will be measured and values returned – but no data will be logged.

If space is later made available – for example by deleting other jobs or their data from the *DT80*, or by inserting a larger capacity USB device (if the schedule is set up to log directly to USB device) – then logging will automatically resume and the **Attn** LED will stop flashing.

Note that it is possible that some of a job's schedules will be able to create their store files, and therefore start logging, while some will not. The **Attn** LED will continue to flash whilst there are any schedules for which logging is enabled but which cannot create a store file.

To determine how much space is available on the internal file system for creating new store files, see *Checking Logging Status* (P82).

Store File Full

Normally, when a store file fills up it will automatically begin overwriting the oldest logged data. However, in some circumstances the older data may be more valuable than the newest. In these cases you would use the **NOV** (no overwrite) schedule option. If this option is set then logging for that schedule will stop when the store file becomes full, and the **Attn** LED will start flashing.

Logging will resume (and the **Attn** LED will stop flashing) if you extract the data to an **archive file** using the **MOVEDATA** command (see *Archiving Logged Data* (P86)), or if you delete the logged data using **DELDATA** and/or **DELALARMS** (see *Deleting Logged Data* (P88)).

To determine how many records have been logged to a store file, see *Checking Logging Status* (P82).

Pre-existing Store Files

When a job is entered, the *DT80* checks whether there are any pre-existing store files associated with the job name. For example, if you enter a job called "FIDO" (using **BEGIN"FIDO"** ...) then the *DT80* will check to see if there are any existing store files under the **B:\JOBS\FIDO** directory.

If there are existing store files, the *DT80* then checks to see whether the existing store files were created by the same job as the one being entered (this information is encoded within the store file). Note that to be considered the same, the new job's program text must be identical to that used to create the store files.

If the new job matches then logging will commence and data will be **appended** to the existing store files.

If, however, the job being entered is not the same as the job used to create the store files then the new job will not be loaded and an error message will be displayed, eg:

```
Cannot log: job 'FIDO' has existing data/alarms
```

To get around this you need to either:

- rename the new job, or
- remove the existing data in the store files, using **MOVEDATA**, or **DELDATA** and/or **DELALARMS**.

This check ensures that all data in a store file is consistent.

Store Medium Absent

This is applicable only to schedules which are set up to log directly to a USB memory device. If the USB device is removed while the job is running then logging will be suspended for the affected schedules, and the **Attn** LED will start flashing.

When a USB device is inserted, the *DT80* will attempt to resume logging. This may involve creating new store files if they don't already exist on the USB device.

Checking Logging Status

A number of commands can be used while a job is running to monitor the data logging status.

Schedule LOGON/LOGOFF Status

The **STATUS** command (P138) can be used to confirm which schedules currently have logging enabled. The relevant information is in the fifth line of the **STATUS** output; you can return just this line using

```
STATUS5
A C E,none Scan Schedules LOGON,LOGOFF
```

In the above example, all three schedules in the job (A, C and E) have logging enabled.

(Note that even if a schedule is reported as having logging enabled, it may be prevented from physically logging data due to one of the reasons described in *Factors Which May Prevent Logging* (P82).

Free Space for Creating New Store Files

To determine how much space is available on the internal file system for creating store files you can use the **DIR"B:"** command (or **DIR"A:"** for a USB device). This will list the various files stored in the root directory, and will then show the remaining free space, eg:

```
3 File(s)    42902196 Bytes free
```

Alternatively, system variable 1SV (P34) will return the current free space, in kbytes (1kbyte = 1024 bytes), on the internal file system (or 3SV for a USB device), eg:

```
1SV
1SV 41896.0
```

Number of Records Logged

To determine how many records have been logged to a store file, you can use the **DIRJOB***jobspec* command. As discussed in *Job Commands* (P56), *jobspec* can take one of three forms:

- **DIRJOB** – list details for current job, or
- **DIRJOB"***jobname***"** – list details for specified job, or
- **DIRJOB*** - list details for all jobs stored on *DT80* internal file system

DIRJOB displays a report similar to the following:

```
Job   *GOOSE
S  SchedID  Log  Data Recs   Capacity  First               Last                Alarm Recs   Capacity  First               Last
=============================================================================================================================
A  rabbit   Off         28      54290  23/08/2005 12:39:35  23/08/2005 13:40:02          2       5128  23/08/2005 12:39:48  23/08/2005 12:39:57
B  hare     On         400        400  23/08/2005 13:09:35  23/08/2005 13:50:12          0          0
```

In this example, 28 data records and 2 alarm records have been logged for schedule A, and the total capacity of this schedule's store file is 54290 data records plus 5128 alarm records. Schedule B's store file has no space allocated for alarms (probably because no numbered alarms were defined in the schedule). Its data space is full – earlier records will already have been overwritten.

If fixed format mode has been selected (**/H**) the information for a job is returned as a single fixed format record.

The number of logged data and alarm records for each schedule of the current job are also available in system variables 30SV – 53SV (see *System Variables* (P34)), eg.

```
32SV
28.0
```

Halt and Go During Data Logging

While data logging is in progress, data from each report schedule is progressively stored into its respective store file. Whenever a schedule is halted (using a **H** or **H***sched* command), a **discontinuity record** is written to the file. This is a special kind of data record that indicates that execution of the schedule was interrupted.

Retrieving Logged Data

Logged data can be retrieved from the *DT80* at any time. This can be done in one of two ways:

- The *DT80* can output fixed format (comma separated) data records via the active communications port. This is referred to as **unloading** the data.
- The data in a store file can be copied or moved to an **archive file**. An archive file has the same format as a store file, except that any empty space is removed. The archive file can then be transferred to a host PC using a USB memory device or via FTP (File Transfer Protocol). Once copied to a PC the archive file(s) can then be opened by applications such as **DeView**.

Unloading Data and Alarms

Unload Commands

Logged data and alarm records are unloaded using the **U** and **A** commands respectively. These commands have the same syntax:

Command	Function
U*storefile-spec*	Unload all logged data from the store file(s) specified by *storefile-spec*
U*storefile-spec***(***from***)**	Unload logged data from the store file(s) specified by *storefile-spec*, starting at timestamp *from*
U*storefile-spec***(***from***)(***to***)**	Unload logged data from the store file(s) specified by *storefile-spec*, starting from timestamp *from* up to but not including timestamp *to*
U*storefile-spec***[***fromISO***]**	Unload logged data from the store file(s) specified by *storefile-spec*, starting at timestamp *fromISO*
U*storefile-spec***[***fromISO***][***toISO***]**	Unload logged data from the store file(s) specified by *storefile-spec*, starting from timestamp *fromISO* up to but not including timestamp *toISO*
Q	Terminate an unload

(substitute **A** instead of **U** to unload alarms)

where:

- ***storefile-spec*** is either:
  - **"***filename***"** – unload data from the specified store file (This form is often used to unload archive files, see *Archiving Logged Data* (P86)), or
  - *schedule* – unload data for the specified schedule within the current job, or
  - **"***jobname***"***schedule* – unload data for the specified schedule within the specified job, or
  - **"***jobname***"** – unload data for all schedules within the specified job, or
  - nothing – unload data for all schedules within the current job
- ***from*** is either:
  - **BEGIN** (unload starting with the earliest logged record), or
  - *time* – time in the format specified by parameter P39 (unload records logged at or after this time today). Note that the time fields are optional, zero is assumed for fields that are omitted.
  - *time***,***date* – time and date in P39 and P31 format (unload records logged at or after this time and date)
- ***to*** is either:
  - **END** (unload up until the latest logged record), or
  - *time* – time in the format specified by parameter P39 (unload records logged prior to this time today). Note that the time fields are optional, zero is assumed for fields that are omitted.
  - *time***,***date* – time and date P39 and P31 format (unload records logged prior to this time and date)
  - nothing (unload data for all schedules)
- ***fromISO*** and ***toISO*** are date/time strings in **ISO format** (*yyyy***/***mm***/***dd***,***hh***:***mm***:***ss***,0.***uuuuuu* where **0.***uuuuuu* is the fractional seconds. Note that the time fields (*hh*, *mm*, *ss* and **0.***uuuuuu*) are optional, zero is assumed for fields that are omitted.)

Note that if *from* and *to* are identical then *to* is "rounded up" to the start of the next time period (second, minute, hour or day, depending on how many time components were specified. For example if hours and minutes are specified then *to* is rounded up to the start of the next minute.

Examples

Some sample unload commands are shown below:

Command	Data to Unload
U	all logged data for the current job
UX	data for schedule X only
A"ZOOM"	all alarms for job "ZOOM"
U"CABBAGE"B	data for schedule B of job "CABBAGE"
U"B:\JOBS\X1\C\2006-10-03T17-55-01.DBD"	all data from the specified store file
U(15)	data logged since 15:00 today
A(0,27/9/2005)	all alarms logged since 00:00, 27-Sep-2005 (assumes P31=1)
UC(BEGIN)(12:15,25/12/2006)	data for schedule C logged up until 12:15, 15-Dec-2006
U[2006/1/1,9:00][2006/1/31,9:00]	all data logged between the indicated times
U(13)(13)	all data logged between 13:00 and 14:00 today (same as **U(13)(14)**)
U()()	all data logged today

Format of Unloaded Data

Logged data and alarms are returned as **fixed format records**.

For the job:

BEGIN"MANGO" RA1S 1*TK 4R 3V(NL) RB5S 1CV=1CV+1 LOGON END

a typical unload sequence might be as follows:

```
U
D,080043,"MANGO",2006/02/06,10:26:13,0.010864,1;A,0,23.4136,1300.22;0068;1D59
D,080043,"MANGO",2006/02/06,10:26:14,0.000366,1;A,0,23.3562,1300.30;0068;0FA2
D,080043,"MANGO",2006/02/06,10:26:15,0.003662,1;A,0,23.2430,1300.30;0068;49A2
D,080043,"MANGO",2006/02/06,10:26:16,0.008666,1;A,0,23.1898,1300.29;0068;AAF5
D,080043,"MANGO",2006/02/06,10:26:17,0.013916,1;A,0,23.1430,1300.28;0068;E222
D,080043,"MANGO",2006/02/06,10:26:18,0.002685,1;A,0,25.2273,1300.29;0068;2ACB
D,080043,"MANGO",2006/02/06,10:26:19,0.007934,1;A,0,27.6343,1300.30;0068;F795
D,080043,"MANGO",2006/02/06,10:26:20,0.014892,1;A,0,28.7219,1300.29;0068;1CAB
D,080043,"MANGO",2006/02/06,10:26:21,0.002075,1;A,0,29.3870,1300.28;0068;7A9E
D,080043,"MANGO",2006/02/06,10:26:22,0.007324,1;A,0,29.8504,1300.28;0068;28FB
D,080043,"MANGO",2006/02/06,10:26:23,0.014160,1;A,0,30.3531,1300.29;0068;0E0A
D,080043,"MANGO",2006/02/06,10:26:24,0.001342,1;A,0,30.6677,1300.29;0068;3052
D,080043,"MANGO",2006/02/06,10:26:25,0.006713,1;A,0,30.8642,1300.29;0068;F7A1
D,080043,"MANGO",2006/02/06,10:26:26,0.011840,1;A,0,30.9801,1300.29;0068;A13C
D,080043,"MANGO",2006/02/06,10:26:27,0.002075,1;A,0,30.5055,1300.29;0068;EF01
D,080043,"MANGO",2006/02/06,10:26:28,0.006225,1;A,0,29.4510,1300.29;0068;1A1D
D,080043,"MANGO",2006/02/06,10:26:29,0.012695,1;A,0,28.4525,1300.29;0068;3217
D,080043,"MANGO",2006/02/06,10:26:30,0.000488,1;A,0,27.8787,1300.29;0068;F1B1
D,080043,"MANGO",2006/02/06,10:26:31,0.006713,1;A,0,27.4039,1300.26;0068;1C9B
D,080043,"MANGO",2006/02/06,10:26:32,0.010742,1;A,0,26.9638,1200.23;0068;A3A6
D,080043,"MANGO",2006/02/06,10:26:33,0.000366,1;A,0,26.5611,1200.25;0068;FBF4
D,080043,"MANGO",2006/02/06,10:26:34,0.005981,1;A,0,26.2456,1300.28;0068;EC99
D,080043,"MANGO",2006/02/06,10:26:35,0.010253,1;A,0,25.8993,1300.28;0068;93EC
D,080043,"MANGO",2006/02/06,10:26:36,0.000366,1;A,0,25.6119,1200.24;0068;543A
D,080043,"MANGO",2006/02/06,10:26:37,0.003662,1;A,0,25.3895,1300.28;0068;B387
D,080043,"MANGO",2006/02/06,10:26:38,0.009277,1;A,0,25.2659,1300.27;0068;B0B1
D,080043,"MANGO",2006/02/06,10:26:39,0.015747,1;A,0,25.0792,1300.27;0068;3126
D,080043,"MANGO",2006/02/06,10:26:40,0.003173,1;A,0,25.0043,1300.27;0068;48B1
D,080043,"MANGO",2006/02/06,10:26:41,0.009643,1;A,0,24.8593,1300.26;0068;7DAF
D,080043,"MANGO",2006/02/06,10:26:42,0.013793,1;A,0,24.6740,1300.26;0068;50AA
D,080043,"MANGO",2006/02/06,10:26:42,0.442871,4;A,0,0.00000,0.00000;0068;7B42
D,080043,"MANGO",2006/02/06,10:27:00,0.065551,1;A,0,23.6096,1300.45;0068;1EF2
D,080043,"MANGO",2006/02/06,10:27:01,0.016723,1;A,0,23.5897,1300.47;0068;D656
D,080043,"MANGO",2006/02/06,10:27:02,0.003906,1;A,0,23.5380,1300.46;0068;914A
D,080043,"MANGO",2006/02/06,10:27:03,0.008422,1;A,0,23.4914,1300.45;0068;E60A
D,080043,"MANGO",2006/02/06,10:27:04,0.015136,1;A,0,23.4313,1300.46;0068;D312
D,080043,"MANGO",2006/02/06,10:27:04,0.710083,4;A,0,0.00000,0.00000;0068;9AD5
D,080043,"MANGO",2006/02/06,10:27:06,0.814697,5;37,ABCDEF01
D,080043,"MANGO",2006/02/06,10:26:15,0.140380,1;B,0,6.00000;0060;461F
D,080043,"MANGO",2006/02/06,10:26:20,0.152954,1;B,0,7.00000;0060;C7E1
D,080043,"MANGO",2006/02/06,10:26:25,0.144042,1;B,0,8.00000;0060;4E70
D,080043,"MANGO",2006/02/06,10:26:30,0.136840,1;B,0,9.00000;0060;1FB2
D,080043,"MANGO",2006/02/06,10:26:35,0.145874,1;B,0,10.0000;0060;079B
D,080043,"MANGO",2006/02/06,10:26:40,0.145385,1;B,0,11.0000;0060;850D
D,080043,"MANGO",2006/02/06,10:26:42,0.453735,4;B,0,0.00000;0060;6D3E
D,080043,"MANGO",2006/02/06,10:27:00,0.766723,1;B,0,12.0000;0060;A402
```

```
D,080043,"MANGO",2006/02/06,10:27:04,0.720947,4;B,0,0.00000;0060;368B
D,080043,"MANGO",2006/02/06,10:27:06,0.928955,5;9,ABCDEF01
D,080043,"MANGO",2006/02/06,10:27:06,0.933959,3;46,ABCDEF01
```

Note the following points:

- Data records are unloaded schedule by schedule, in the order X, A, B, … K. Within each schedule, records are unloaded in chronological order. So in the above example you see all schedule A records (1s intervals) followed by all schedule B records (5s intervals).
- The `D` at the start of each records indicates that this is a data record, while the digit immediately before the first semicolon (`;`) specifies the type of data record:
  - `1` indicates a **normal data record**. Fields after the semicolon consist of the schedule name, channel offset (normally always 0), and the data values, one for each channel in the schedule that has logging enabled. Thus each schedule A record has two data values (the **3V** channel is not logged, due to the **NL** option), while schedule B records have one.
  - `4` indicates a **discontinuity record**, which is inserted if a schedule is halted. In the above example, both schedules were halted at 10:26:42, then resumed at 10:27:00, then halted again at 10:27:04. A discontinuity record has the same format as a data record except that the data values are always zero.
  - `5` indicates an **end of schedule record**. This has one integer data item, which is the number of data records (including discontinuity records) that were unloaded for the schedule (37 records for schedule A in this example, 9 records for schedule B). The `ABCDEF01` string does not mean anything.
  - `3` indicates an **end of unload record**. Its data item indicates the total number of data records unloaded (37+9 = 46 in the above example).
- All channel values are returned to 6 significant digits, regardless of any formatting channel options (eg. **FF3**). Exponential format may be used (eg. `2.60930e+10` when appropriate).
- Overrange channel values are returned as `99999.9` when unloaded.

Other Considerations

Operation During Unload

Unloading data is a **background** operation, so schedules continue to run and commands can be executed during the unload. Note however that any data that is logged during the unload will not be unloaded.

During an unload, the **/r** (return), **/e** (echo), **/m** (error messages) and **/z** (alarm messages) switches are disabled to prevent transmissions from these sources being inserted into the unload data stream. The *DT80* automatically sets these switches to their previous state on completion of the unload.

To terminate an unload that is in progress, use the **Q** command.

Unload Does Not Remove Data

Unloading data does not delete the data from the logger. If you want to delete data from store files on the *DT80* after completing an unload you need to do this explicitly, using the **DELDATA** or **DELALARMS** commands; see *Deleting Logged Data* *(P88)*.

Archiving Logged Data

Archive Files

A store file can be **archived** at any time. This process involves the *DT80* copying all of the logged data and alarms in a store file to a new file – an **archive file**. The *DT80* can then optionally delete the data from the original store file. The new archive file can be either on the internal file system or on a removable USB memory device.

Creating an archive file does not affect schedule execution, and logging (if it is enabled) continues to the original store file.

An archive file can be thought of as containing a "snapshot" of the store fie at a particular time. When you create an archive file, the *DT80* automatically assigns a name based on the time that it was created, eg: **2005-09-27T04-15-20.DBD**.

There are two ways of getting archive files onto the host PC:

- create the files on a USB memory device and use it to transport the files to the PC
- create the files on the internal file system, then connect to the *DT80*'s inbuilt FTP server and download the files via an Ethernet or PPP link.

Once the archive files have been transported to a PC they can be opened using an application such as **DeView**.

Archive Commands

The following commands can be used to create archive files:

Command	Function
COPYDATA*storefile-spec*	copies logged data to a new archive file on the USB memory device
MOVEDATA*storefile-spec*	copies logged data to a new archive file on the USB memory device, then deletes the data from the original store file
ARCHIVE*storefile-spec*	copies logged data to a new archive file in the same directory as the original store file

where:

- ***storefile-spec*** is either:
  - nothing – copy data for all schedules within the current job, or
  - *schedule* – copy data for the specified schedule within the current job, or
  - **"***jobname***"** – copy data for all schedules within the specified job, or
  - **"***jobname***"***schedule* – copy data for the specified schedule within the specified job, or
  - ***** – copy data for all schedules within all jobs, or
  - **"***filename***"** – copy data from the specified store file

When an archive file is created on the USB memory device, it will be placed in the **A:\SN***serial-num***\JOBS***jobname******sched* directory.

Archive commands provide feedback via a progress bar on the LCD and a message (assuming messages are enabled, ie. **/M**) is returned, eg:

```
COPYDATA
Copying: source: "B:\JOBS\DTCAN169\E\DATA_E.DBD" -> dest:"A:\SN000043\JOBS\DTCAN169\E\2006-02-07T13-36-15.DBD"
Copying: source: "B:\JOBS\DTCAN169\F\DATA_F.DBD" -> dest:"A:\SN000043\JOBS\DTCAN169\F\2006-02-07T13-36-15.DBD"
Done
```

Examples

Some sample archive commands are shown below:

Command	Function
COPYDATA	copy all logged data for the current job to USB memory device.
ARCHIVEJ	copy all logged data for schedule J to a file
MOVEDATA"ZOOM"A	copy data for schedule A of job "ZOOM" to USB memory device then delete data from original storefile
MOVEDATA*	copy all data for all jobs to USB memory device, then delete from original store files.

Using Archive Files

Capturing Pre-Trigger Data

Archive files are mostly used as a means for getting data out of the logger using a USB memory device.

However they can also be used for other applications such as capturing **pre-trigger data** leading up to some event. For example:

```
BEGIN"SPARROW"
RA(DATA:200R:OV)1S 2V
 1CV(W)=1CV+2CV IF(1CV>100){1..2CV=0 ARCHIVEA}
RB1-E 2CV=1
LOGONA
END
```

In this example, a small store file is declared for schedule A (capacity 200 records). The channel of interest (**2V**) is measured and logged once per second, with old values being overwritten. When a negative going trigger pulse is detected on digital input **1D**, the job will log a further 100 samples then execute the **ARCHIVEA** command. The end result is that each archive file that is created will contain 100 samples taken just prior to the trigger event, and 100 samples taken immediately after.

Note If data are being logged at a relatively fast rate, the earliest samples may be overwritten before they can be copied into the archive file, which may result in fewer than expected pre-trigger records being present in the archive file. In the above example there might be 198 samples in the archive (98 pre-trigger, 100 post-trigger)

If you need to have exactly the right number of samples then you should halt sampling for the duration of the archive operation, ie:

```
1CV(W)=1CV+2CV IF(1CV>100){1..2CV=0 HA ARCHIVEA GA}
```

Automatic Data Collection Using ONINSERT

An ONINSERT job *(P57)* is a convenient way to automatically "milk" data from a *DT80*. For example, if you create a file containing the text:

MOVEDATA
and save it to a file **ONINSERT.DXC** in the root directory of the USB memory device, then whenever the memory device is plugged into a *DT80* it will extract all data for the current job (since the last extraction) to an archive file on the USB device.

Managing Logged Data

Deleting Logged Data

The **DELDATA***jobspec* and **DELALARMS***jobspec* commands are used to delete all logged data/alarms for a given job. As discussed in *Job Commands* (P56), *jobspec* can take one of three forms:

- **DELDATA** – delete logged data for current job (all schedules), or
- **DELDATA"***jobname***"** – delete logged data for specified job, or
- **DELDATA*** – delete logged data for all jobs stored on *DT80* internal file system.

(Substitute **DELALARMS** to delete logged alarm records.)

These commands clear all data within the relevant store file, but they do not delete the store file itself. Furthermore, the physical size of the store file (as reported by **DIR** or **DIRTREE**) does not change as a result of deleting the data therein. However the number of logged records reported by **DIRJOB** will now be zero.

Note that deleting logged data or alarms does not prevent further data and alarms being logged.

Note also that if a schedule has the **NOV** (no overwrite) option specified and its store file is full, then using the **DELDATA** command will allow it to automatically resume logging.

Deleting Store Files

Deleting Jobs

A job's store files are physically deleted when the job is deleted using **DELJOB** (see *Deleting Jobs* (P56)). However, note that as a safeguard against accidental deletion, you first need to "empty" the store files using **DELDATA** and/or **DELALARMS**.

Therefore, to delete all trace of job "XOOM" (which comprises two schedules, A and F), you could send the following sequence:

```
DELDATA"XOOM"
Deleting Job XOOM Data - A:deleted, F:deleted
DELALARMS"XOOM"
Deleting Job XOOM Alarms - A:deleted, F:deleted
DELJOB"XOOM"
Deleting Job XOOM - done
```

Formatting Storage Media

A somewhat more drastic method of deleting logged data is to **format** the storage media (ie. internal file system or removable USB memory device) on which it resides. Two commands are available:

Command	Function
FORMAT"A:"	delete all files from USB memory device
FORMAT"B:"	delete all files from internal file system

Warning These commands will immediately delete ALL files on the selected medium:

- For the internal drive (**B:**) this includes all store files for all jobs, all archive files, all job program text and also the event and error diagnostic logs (P137)
- For the USB drive (**A:**) this includes all store files and archive files, including any files copied from other *DT80*s, and any other non-*DT80* related files that happen to be stored on the USB memory device.

Do not use these commands unless you are sure there is nothing valuable stored on the drive.

Note that **profile settings** (P133) and any configured **ONRESET** job (P57) are automatically "backed up" to a special area of internal memory. A **FORMAT"B:"** operation will remove the "working" copies of these files, however they will be automatically restored on the next hard reset. The only way to remove profile settings is to use the **PROFILE** command (or **DELUSERINI** to delete all profile settings), and the only way to remove an ONRESET job is to use **DELONRESET**. (The **FACTORYDEFAULTS** command can also be used to remove both profile settings and the ONRESET job.)

The *DT80* File System

Internal File System (B:)

The *DT80* uses a Windows compatible FAT16/FAT32 file system for storing logged information (data and alarms) and system information. The internal file system uses flash memory, so all files will be preserved even if the *DT80* is reset or loses all power. The size of this file system for a standard *DT80* is 64Mbyte.

The internal file system is used to store information such as:

- program text for stored jobs (eg. **B:\JOBS\MYJOB\PROGRAM.DXC**)
- internal details about a job (**B:\JOBS\MYJOB\STATUS14**)
- store files (eg. **B:\JOBS\MYJOB\A\DATA_A.DBD**)
- archive files (eg. **B:\JOBS\MYJOB\A\2006-04-01T09-00-02.DBD**)
- current profile settings working copy (**B:\INI\USER.INI**)
- current ONRESET job working copy (**B:\ONRESET.DXC**)
- system event and/or error logs (**B:\EVENTS\EVENT.LOG** and **ERROR.LOG**)
- file system recovery information (**B:\FAILSAFE**), which helps prevent any corruption that might otherwise occur if power is unexpectedly removed.

Note Path and file names shown above are for illustrative purposes only, and may change in future firmware revisions.

A typical directory listing (see *File Commands* (P91)) of the internal file system might look like:

```
DIRTREE"B:"
Volume in drive B has no label.

 2006/02/06 12:13     133120 <RO>       -  FAILSAFE
 2006/02/06 12:13                <DIR>  -  EVENTS
 2006/02/08 12:17       1501            -    EVENT.LOG
 2006/02/06 12:36                <DIR>  -  INI
 2006/02/08 10:56        199            -    USER.INI
 2006/02/07 10:50        213            -    USER.BAK
 2006/02/06 12:14                <DIR>  -  JOBS
 2006/02/06 15:17                <DIR>  -    SPARROW
 2006/02/06 15:17                <DIR>  -      A
 2006/02/06 15:56       4208            -        DATA_A.DBD
 2006/02/06 15:19       3028            -        2006-02-06T15-19-36.DBD
 2006/02/06 15:56       4208            -        2006-02-06T15-56-54.DBD
 2006/02/06 16:33       4208            -        2006-02-06T16-33-56.DBD
 2006/02/07 13:02        138            -      STATUS14
 2006/02/07 13:02        121            -      PROGRAM.DXC
 2006/02/06 16:01                <DIR>  -    UNTITLED
 2006/02/08 10:53         95            -      STATUS14
 2006/02/08 10:53         41            -      PROGRAM.DXC
 2006/02/07 13:03                <DIR>  -    DTCAN169
 2006/02/07 13:03                <DIR>  -      A
 2006/02/07 13:03    1051584            -        DATA_A.DBD
 2006/02/07 13:03                <DIR>  -      D
 2006/02/07 13:03    1049104            -        DATA_D.DBD
 2006/02/07 13:03       3466            -      STATUS14
 2006/02/07 13:03      17523            -      PROGRAM.DXC
           25 File(s)    63074304 Bytes free
```

In this example there are three jobs stored on the *DT80*: SPARROW (one schedule; three archive files have been created at various times; in the case of the first one the store file was not yet full at the time it was created, hence its size is smaller than the others), UNTITLED (no schedules have logging enabled, hence no store files) and DTCAN169 (two schedules with logged data).

External USB Devices (A:)

An external USB memory device plugged into the *DT80* can be used to store:

- store files (eg. **A:\SN081234\JOBS\MYJOB\A\DATA_A.DBD**) for schedules that are configured to log directly to the USB device (using the **"A:"** schedule option)
- archive files (eg. **A:\SN081234\JOBS\MYJOB\A\2006-04-01T09-00-02.DBD**)
- an ONINSERT job (**A:\SN081234\ONINSERT.DXC**) which will run when the memory device is inserted into this *DT80*

- an ONINSERT job (**A:\ONINSERT.DXC**) which will run when the memory device is inserted into any *DT80*
- other files which have been manually copied from the *DT80*, eg event logs
- other files not related to the *DT80*

Notice that *DT80* related files are always stored in a subtree whose name is based on the *DT80* serial number. This allows data from a number of different *DT80*s to be collected on the one USB device.

A typical directory listing (see *File Commands* (P91)) of a USB device might look like:

```
DIRTREE"A:"
Volume in drive A is JD1

    2006/02/06 12:40                <DIR> -  SN080043
    2006/02/06 12:40                <DIR> -    JOBS
    2006/02/06 12:40                <DIR> -      UNTITLED
    2006/02/06 12:40                <DIR> -        A
    2006/02/06 12:40    1048784           -          DATA_A.DBD
    2006/02/07 13:36                <DIR> -      DTCAN169
    2006/02/07 13:36                <DIR> -        A
    2006/02/07 13:36       8928           -          2006-02-07T13-36-15.DBD
    2006/02/07 13:36                <DIR> -        D
    2006/02/07 13:36       4788           -          2006-02-07T13-36-15.DBD
    2006/01/27 10:46                <DIR> -  SN080122
    2006/01/27 10:46        202           -    ONINSERT.DXC
    2006/02/07 17:46                <DIR> -    JOBS
    2006/02/07 17:46                <DIR> -      XAM
    2006/02/07 17:46                <DIR> -        A
    2006/02/07 17:46    1048784           -          DATA_A.DBD
    2006/02/07 17:46                <DIR> -        B
    2006/02/07 17:46    1048784           -          DATA_B.DBD

              18 File(s)    125306880 Bytes free
```

In this case the memory device has been used in two different *DT80*s. Serial number `080043` has logged some data directly to the device as part of job `UNTITLED`, while archive files have been created for job `DTCAN169` – probably using **COPYDATA**. Serial number `080122` has logged data directly to store files on the device, and it also has an ONINSERT job defined, which will run whenever the memory device is plugged into *DT80* serial number 080122.

Supported USB Device Types

For an external USB memory device to be recognised by the *DT80*, the device must:

- draw no more than 100mA from the USB bus, and
- support the standard USB "mass storage" device class interface (this includes most USB "memory sticks", MP3 players and USB hard disks, but does not include devices such as USB printers, modems and so on), and
- have the primary disk partition formatted using a FAT16 or FAT32 file system.

USB memory devices are nearly always shipped pre-formatted using a FAT16/FAT32 file system.

If a memory device is inserted that is not properly formatted, the *DT80* will display:

```
USB device unrecognised
```

on the LCD.

The **FORMAT"A:"** command (see *Formatting Storage Media* (P88)) can be used to re-format the device. This will delete all data from the device. Alternatively the device can be formatted in a Windows based computer.

Using a USB Memory Device

Startup

When a USB memory device is first plugged in, the *DT80* needs to read various information from the device before it can be used. This process can take several seconds (possibly a minute or more for large media), but it is a background operation so sampling and logging can continue. The *DT80* displays

```
Reading USB device
```

on the LCD while this operation is in progress.

Once the USB device is ready, it can be accessed in the same way as the internal drive.

Removal

Important The USB device must not be removed while it is being accessed. Doing so may result in data corruption.

To safely remove a USB device, you should always first issue the **REMOVEMEDIA** command. This command is also one of the default options on the front panel function menu (P95). This command will:

1. suspend logging for any schedules that are configured to log directly to the USB memory device. This will cause an error message to be returned, and the **Attn** LED will start flashing.
2. make sure that all required information has been fully copied to the device and all files are closed
3. shut down the device. If the device has an indicator light, it should now be off.

If you change your mind and want to keep using the device, you will now need to remove it and then re-insert it.

File Commands

The *DT80* provides a number of general purpose file manipulation commands. These will work both for files stored on the internal file system (**B:**) and an external USB memory device (**A:**), if one is present.

Command	Function
COPY"*source*""*dest*"	reads the file *source* and creates a copy called *dest*
DIR"*path*"	lists the contents of directory *path*. If *path* is not specified, **B:** is assumed.
DIRTREE"*path*"	lists the contents of directory *path* and all sub-directories. If *path* is not specified, **B:** is assumed.
TYPE"*source*"	displays the contents of file *source*. Use only with text files.
DEL"*source*"	deletes the file *source*. Use with care!
DELTREE"*path*"	deletes the directory *path* and all subdirectories, which is a quick way to remove a job and all of its data. Use with care!
FORMAT"*drive*"	deletes all files from *drive* and re-creates the file system. Use with great care!

Note These commands are for advanced users only. In most cases the standard *DT80* commands (eg **DIRJOB**, **DELDATA** and so on) are preferred. Furthermore, note that file names or locations may be subject to change in future firmware versions.

Note also that the *DT80* does not have a concept of a "current directory". Therefore all parameters specified above must specify the full path, including drive letter. Note also that spaces may optionally be inserted between parameters.

For example, if you want to copy the *DT80*'s system event log to a USB memory device for later analysis you could use:

```
COPY "B:\EVENTS\EVENT.LOG" "A:\EVENT_20060219.LOG"
Done
```

Important The **DeTransfer** program, which is often used to supervise the *DT80*, has a number of special commands that begin with a **** (backslash) character. These are interpreted by **DeTransfer** and not sent to the *DT80*. In order to send a **** character from **DeTransfer**, you need to enter a double backslash (****). For example, the above example would be entered into **DeTransfer** as follows:

```
COPY "B:\\EVENTS\\EVENT.LOG" "A:\\EVENT_20060219.LOG"
```

This rule applies to **DeTransfer** only; it does not apply to the "Text" window in **DeLogger**, for example.

Data Recovery

Prevention

If you accidentally remove a USB device while it is being accessed, then it is possible that the file system on the USB device may be corrupted. (The same applies if you remove a USB memory device from a Windows computer without selecting the "Safely Remove Hardware" option.)

The internal file system or USB memory device may also be corrupted if the logger suddenly loses all power while it is writing to the disk.

To minimise the chance of data loss due to these causes, remember to:

- ensure that the battery link is in place. This will allow the *DT80* to keep operating normally for a period of time, in the event of an external power failure
- always use the **REMOVEMEDIA** command before removing the memory device if there are schedules logging directly to the USB device

Note that the *DT80* provides some protection against gradual power failure (eg. the internal battery becoming discharged). If it detects that the supply voltage is becoming critically low, the *DT80* will automatically close all store files and force the unit into low power **sleep** mode *(P145)*. The *DT80* will remain asleep until the power supply recovers to an adequate level.

Recovery

Media Removal

If a USB device was accidentally removed while it was being logged to, you should plug it into a Windows computer. Before attempting to open any of the files, run a Windows file system check utility (eg. type **chkdsk** *drive***:** at a command prompt, where *drive* is the drive letter assigned to the USB device). This will detect and if possible repair any inconsistencies in the file system structure.

Note that even if there are no file system errors, there may still be corruption in one or more store files (**DATA_***x***.DBD**) if they were being actively logged to at the time of the media removal. Copy the files to the host PC and verify that they can be opened successfully and all the expected data is there.

If the files appear to be damaged (eg. they don't open correctly in **DeView**) then the original files on the USB device should be deleted before inserting the device back into the *DT80* and re-enabling logging. This avoids having the *DT80* attempt to log data to a damaged file, possibly causing more damage in the process.

Power Failure

If external power was lost and the battery link was not present or the internal battery was flat, then the internal file system may be corrupted (if it was being written to at the time of the failure).

When power is restored, the *DT80* will attempt to automatically repair the file system (using the **B:\FAILSAFE** file) if necessary.

Before re-enabling logging, it would be prudent to manually copy to a USB device the store files for the job that was active at the time of the failure, eg:

```
COPY "B:\JOBS\JOE\A\DATA_A.DBD" "A:\JOE_SAVE\DATA_A.DBD"
```

and verify that they can be opened on the host PC.

If a store file is damaged, contact dataTaker for assistance. We may be able to recover data from the file. But note that prevention is always better than cure.

Part H — *DT80* Front Panel

The *DT80* front panel has a 2 line by 16 character back-lit liquid crystal display, 6 keys and 3 status indicator lights. The display provides information about *dataTaker* data logger status, channel data, alarms and store operation. In addition the display will indicate conditions that require attention and USB memory device status.

The *DT80* from cannot be programmed from the front panel. However, pre-defined commands can be issued by selecting from the function list via the front panel.

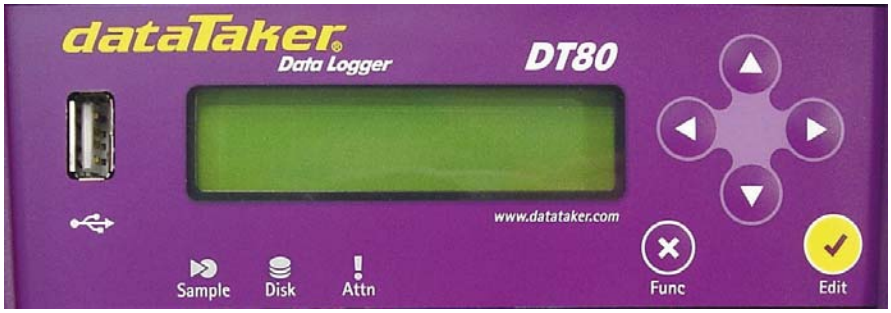


Figure 22 DT80 Display

Display

(Not applicable to DT81)

The display normally shows the current value for all channels and alarms for the current job. Each channel or alarm is shown one at a time. The actual channel or alarm shown is selected by pressing the up and down directional keys on the front panel. In addition it shows several status values that can also be selected via the up and down directional keys. The channels and alarms are arranged in the same order that they are defined in for the current job.

Displaying Channels and Alarms

When channel data is displayed, the top line of the display shows the channel identification. The default is the channel number and type. If a channel identification text has been entered as a channel option, then the first 16 characters of that text is displayed.

When alarms are displayed the top line of the display identifies the alarm and the state of the alarm – ON or OFF. If the alarm channel definition includes identification text, then this is displayed when the alarm is not true. If the alarm contains action text, this is displayed when the alarm is true. Alarms must be numbered to be displayed.

The bottom line on the display shows the most recent reading as a numeric value or bar graph. If the channel or alarm has not yet been sampled, the display shows " – – –.

For example, assume that the following job has been defined:

```
BEGIN"MYJOB"
RA1M 1TK("Boiler Temp",FF0)
 2LM35
 ALARM4(3V>2000)"Over voltage"
 1CV(W)=1CV+1
 ALARM7(4TT("Oven OK")>107)"Oven Over Temp"
END
```

The following "screens" will then be available. These can be scrolled through using the up and down arrows on the keypad.

Display screen	Comments
DT80 V5.02 MYJOB	The default "sign-on" screen indicates the *DT80*'s firmware version number and the name of the currently loaded job (**No current job** is displayed if there isn't one)
Date: 23/10/2005 Time: 16:44:02	Current date and time (format can be changed using P31 and P39)
Battery: 90% ↓ -290mA 6.2V	Internal battery status. This shows the approximate battery charge as a percentage, a charge (↑) or discharge (↓) indicator, battery current (negative=discharging) and the battery terminal voltage. **NC** is displayed if the internal battery is not connected.

Boiler Temp 97 °C	First user channel (user defined channel name)
Channel 2LM35 17.9 °C	Second user channel (default channel name)
Alarm4 OFF 1356.3 mV	Alarm #4 state
Oven Over Temp 117.2 °C	Alarm #7 state (alarm text replaces channel name when alarm is active)

Note that channel **1CV** is not displayed because it is defined as a working **(W)** channel. Working channels are neither logged, returned nor displayed.

Bar Graph

The channel value can be shown as a bar graph instead of a numeric value by using the **BG** channel option. The **BG** option allows the values to be set that represent the left and right side of the graph scale. The channel label can be used to set the graph scale labels. For example:

Display screen	Comments
E--Fuel Level--F ▮▮▮▮▮▮▮	**4V("E--Fuel Level—F",BG:10:900)** displays zero scale (no bars) if measured voltage < 10mV; displays full scale if voltage > 900mV

Controlling what is shown on the display

All defined channels and alarms will be shown on the display, **except for:**

- channels which specify the **ND** (no display) channel option
- working channels (**W** channel option)
- un-numbered **ALARM** of **IF** channels
- channels used as the condition in an **ALARM** of **IF**

Enable/Disable status screens

Status screens can be enabled or disabled for display by P19. Each bit in this parameter value represents a status screen. A "1" enables and "0" disables. The bit mapping is:

Bit Number	Decimal Value	Status Screen
0	1	Sign-on
1	2	Date / Time
2	4	Battery Status
3	8	Reserved
4	16	Reserved
5	32	Reserved
6	64	Reserved
7	128	Reserved

To make screens available set **P19** to the sum of the decimal values following the required screens, e.g. for Battery Condition and Current Job screens only set P19=5 (i.e. 1 + 4). By default P19=255 and all screens are available. If P19=0 and there are no channels or alarms to display then the sign-on screen is displayed.

Transient Messages

The display may also show temporary status screens, such as.

Display screen	Comments
Reading USB device	When a USB device is inserted the *DT80* needs to read certain system information from it before it can be used.

USB device unrecognised	This indicates that the *DT80* does not recognise the device as a valid USB mass storage device.
Processing ONINSERT.DXC	If the USB memory device contains a file called **ONINSERT.DXC** then it will be automatically loaded and run by the *DT80*
Copying Data ▮▮▮▮▮▮▮	This indicates progress during a **COPYDATA** operation.

Display Backlight

The display backlight stays on when external power is supplied. If the *DT80* is running from internal battery then the backlight will only stay on for 30 seconds after the last key press. The actual period that the backlight stays on for after a key press is controlled by P17 in seconds.

User Defined Functions

(Not applicable to DT81)

The user can define named macros called functions. These functions can be executed by the user via the LCD and keypad of the *DT80*.

Functions can be very useful. For example the functions can be used to completely reprogram the *DT80*, with a different program assigned to each function. The functions can also be assigned by ALARMS.

The FUNCTION command

Functions are defined with the use of the **FUNCTION** command. The user may define up to 10 functions. A function is deleted when the user provides no label or command text following the = (equal) character. The syntax is as follows:

FUNCTION*n*="*label*"{*commands*}

where

Parameter	Meaning
n	This is the ID or slot number of the function to be redefined. It must be an integer in the range from 1 to 10 inclusive.
"*label*"	This is optional. It is a label that effectively names the function. This label will be displayed on the LCD when the user scrolls through the function list. It can be no more that 16 characters long. If this option is not supplied then up to 13 characters of command text will become the label text as displayed on the LCD.
commands	This can be a list of any white space separated *DT80* commands, enclosed in braces. These commands will be executed when the user selects the associated function from the function list.

Examples:

Command	Description
FUNCTION1="Start"{G}	This function would display **'Start'** and when selected would issue the G or go command to the logger which would start all schedules
FUNCTION2="Stop"{H}	This function would display **'Stop'** and when selected would issue the H or halt command to the logger to stop all schedules
FUNCTION3="Init"{1V(S1,=10CV)}	This would display **'Init'** and when selected would store the current vale of 1V scaled to channel variable 10CV
FUNCTION4="Clear"{1..20CV(W)=0}	This function would display '**Clear**' and would set channel variables 1-20 back to zero.

Selecting Functions

Pressing the **Cancel/Func** key will cause the function list to be shown on the display. Once function is shown at a time, and only those functions which have been defined are shown. The up and down direction keys can be pressed to scroll through the list of functions. Once the desired function is visible on the display it can be executed by pressing the **OK/Edit** key. If you wish to exit the function list without executing any function then press the **Cancel/Func** key to cancel the function selection process.

After selecting the function to execute the display will indicate that the function selected has been initiated.

Default Functions

Following reset, the *DT80* automatically defines two commonly used functions:

```
FUNCTION9="Remove USB"{REMOVEMEDIA}
FUNCTION10="Copy logged data"{COPYDATA}
```

These can be redefined or removed if desired.

Displaying Currently Defined Functions

To display the current function definitions you can use:

- **FUNCTION***n* to display the definition for one function, or
- **FUNCTION** to display the definitions for all 10 functions.

Keypad operation

(Not applicable to DT81)

Direction Keys

The up and down direction keys allow scrolling through the available channels, alarms and status screens on the display. When the function list is shown, then the up and down direction keys allows scrolling through the list of available of functions.

The left and right direction keys are not presently used. They will be used in a later version of the firmware that supports editing of values.

OK (Edit) Key

The **OK/Edit** key is used to select a function to execute when the function list is displayed. The edit function will be used in a later firmware version that supports editing of values.

Cancel (Function) Key

The **Cancel/Func** key is used to enter the function list display. It can be pressed again to exit the function list without selecting a function.

Special Key Sequences

Entering Bootstrap Mode

Holding down the **OK/Edit** key during the logger reset or power-up sequence will force the logger into bootstrap mode. This would only be required if there is a corruption of the firmware in the logger.

Status Indicator Lights

Sample Indicator

The **Sample** indicator is illuminated whenever any channel in the current job is sampled. This includes all analog, digital and internal channels.

Disk Indicator

The **Disk** indicator is illuminated whenever the internal disk is reading or writing. For example, the disk indicator will illuminate when writing data to the internal data store or when unloading data from the data store.

Power Indicator (DT81)

This indicator flashes every 3 seconds while the DT81 is awake. A long "LED on" time followed by a short "LED off" time indicates that the DT81 is externally powered; a short "on" time followed by a long "off" time indicates battery power.

Attn Indicator

This LED is used to:

- warn that an unexpected *DT80* reset has occurred (flashing)
- warn that logging has been partially or fully suspended (flashing)
- indicate a warning state under the control of a user program (continuously on)

Unexpected Reset

A message such as the following may be displayed following *DT80* reset, in conjunction with a flashing **Attn** LED. Press any key to clear the message and the flashing LED.

Display	Comments
DT80 restarted Power loss	The *DT80* lost power, both external and the internal battery. This message may also be displayed if the hardware reset button (accessed using a paper clip) is pressed.
DT80 restarted Safe mode	A "triple-push" reset was performed (by pressing the hardware reset button three times within 10s), which temporarily restores factory settings
DT80 restarted SW exception	This indicates a possible problem with the *DT80* firmware. Contact dataTaker Support if you see this message.
DT80 restarted All mem cleared	The *DT80* lost power, and all internal RAM has been cleared, probably due to the internal Lithium memory backup battery being flat. Programs and logged data will not be affected but you will need to reset the *DT80*'s time/date.

Logging Suspended

If data for one or more schedules cannot be logged for some reason then the *DT80* will continue to run the job but it will flash the **Attn** LED and display a message such as the following. Pressing a key will clear the message from the display, but the **Attn** LED will keep flashing until space is made available (eg. by deleting other jobs' data) or the job is stopped.

Display	Comments
Cannot log No space	There is insufficient space on the specified logging drive (internal or USB device) to allow a data file of the requested size to be created.
Cannot log Data full	One or more schedules have been set to "no-overwrite" mode (**NOV** schedule option), and the allocated space is now full
Cannot log No USB device	The **"A:"** schedule option (log directly to USB device) has been specified, but no USB device is inserted.
DT80 restarted All mem cleared	The *DT80* lost power, and all internal RAM has been cleared, probably due to the internal Lithium memory backup battery being flat. Programs and logged data will not be affected but you will need to reset the *DT80*'s time/date.

User Control

You can also turn the **Attn** LED on or off using the **SATTN** (Set Attention) and **CATTN** (Clear Attention) commands.

Alternatively, the **1WARN** channel type (which works in the same way as a digital output channel) may be used.

For example:

```
RA1S ALARM1(3TT>500)"Meltdown"{SATTN}
```

will cause the LED to come on and stay on if the alarm is triggered, and

```
RA1S 1CV=(1CV+1)%10 IF(1CV<.5){1WARN(R,200)=1}
```

will give a 200ms flash every tenth time the schedule is scanned.

Part I — Web Interface

What is the Web Interface?

The *DT80* provides an embedded **web interface** that provides simple, intuitive access to the logger's operations using a standard web browser. You can view current sensor readings, job status and access data and other files stored in the file system.

The standard web interface is built into the logger's firmware. It does not need to be installed on the logger or on your PC. Simply use your existing browser to browse to the logger's IP address and the home page will be displayed.

The interface is standards compliant which helps to make the interface accessible from a wide variety of web browsers. This makes it possible to view the interface on a wide range of devices such as desktop PCs, PDA devices and mobile phones.

For advanced users, the web interface can also be customised by developing new web pages and loading them onto the *DT80*'s internal file system.

Note It is not currently possible to program or configure the *DT80* using the web interface – for that you still need the standard text-based command interface. The web interface does however provide a very convenient way to view data and status information. In this respect it performs a similar role to the DT80's LCD display, but with far greater capabilities.

Browser Requirements

The web interface uses a minimum of browser functions to provide its interface. The browser must however support XHTML 1.0, CSS 1 and JavaScript to fully support the DT80's web interface.

The following browsers have been used and found to be compatible

- Internet Explorer 6.x, Pocket Internet Explorer (on Windows Mobile 5 and Windows Mobile 2003)
- Mozilla 1.x
- Firefox 1.x
- Opera 8.x, Opera Mobile and Opera Mini
- Safari 1.x
- Mobile Explorer
- Palm Web Browser 2.x (Palm Garnet OS)
- Sony PlayStation Portable Web browser
- Blazer

Connecting to the Web Interface

In order to access the *DT80* web interface, a **TCP/IP** connection between the *DT80* and the your PC is required. You can connect the *DT80*'s Ethernet port to a local area network, or directly to the PC using an Ethernet cross-over cable, or you can set up a serial connection to the host RS232 port using the PPP protocol. See *Ethernet Communications* (P122) and *PPP Communications* (P127) for more details.

Once a TCP/IP connection has been established, all you need to do is type the *DT80*'s IP address into the address field of your browser. The web interface home page should then be displayed.

Navigating the Web Interface

The built-in web interface consists of the five pages – **Home**, **Details**, **Status**, **Admin** and **Help**. The design of the web interface follows a tab based metaphor where each tab represents a HTML page.

Figure 23: Navigation tabs

To navigate the web interface, simply click on the desired tab heading. The Home page is displayed by default when the web interface is first loaded.

Home Page

The **Home** page displays the data logger's model, firmware version, serial number, the current job name and the current date and time.

Figure 24: Home page

The page does not provide any functionality other than the display of data logger general information.

Channels Page

The **Channels** page displays a table listing all channel entries defined for the current job. This table shows the most recent measurement for each channel, along with the time that the measurement was taken.

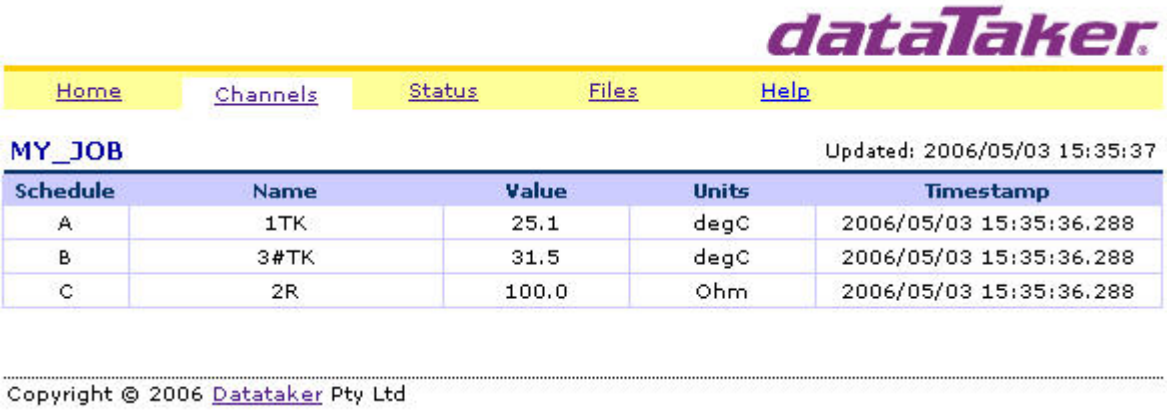


Figure 25: Channels Page

The Channels page is updated every 30 seconds. The time at which the table was last updated is displayed on the top-right corner of the channel listings table.

Status Page

The **Status** page displays status information for each defined schedule in the current job. The following information is displayed for each schedule:

- Schedule Name
- Schedule Trigger
- Schedule Status – whether it is active or halted.
- Schedule Logging State – enabled or disabled
- The number of data records / alarms logged
- The capacity of the schedule's store file
- The timestamp for the first and last data / alarm records stored

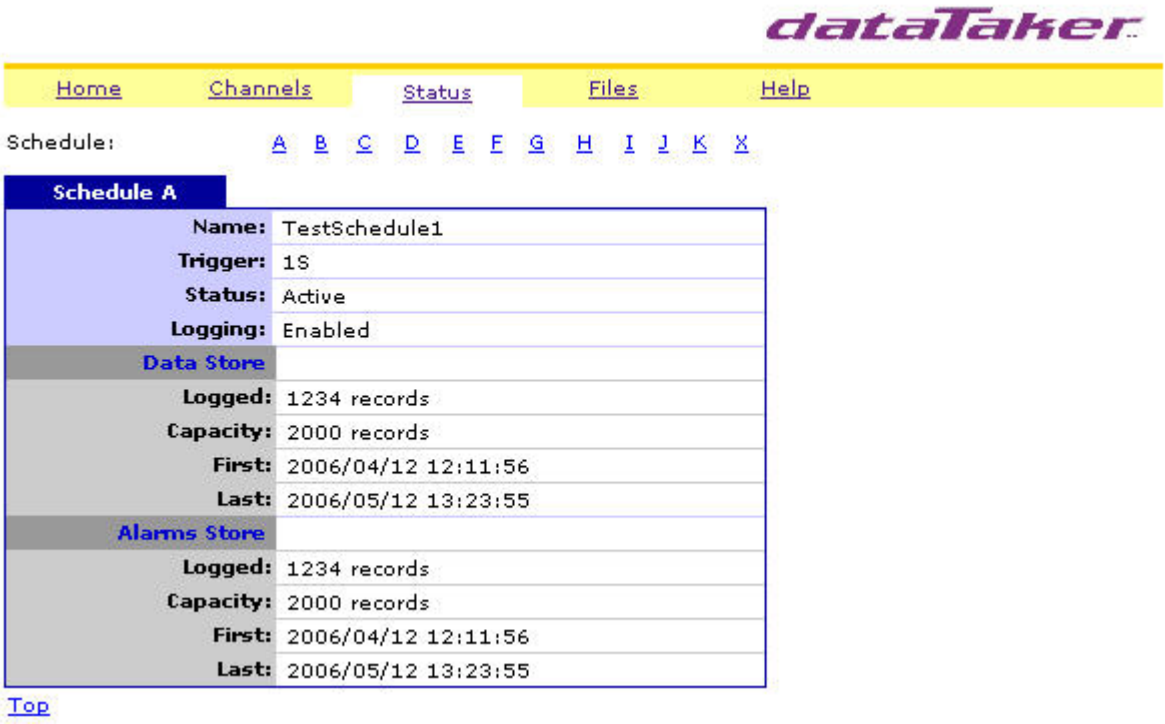


Figure 26: Status Page

All schedules configured for the current job will be displayed when this page is loaded.

The schedule navigation links at the top of the page (A – X) allow you to jump directly to a particular schedule.

Finally, the Top link displayed below each schedule's data provides a quick way to scroll back to the top of the page.

Files Page

The **Files** page provides the ability to view files stored on the *DT80's* file system. Direct links are provided for the system event and error logs (see *Event Logs* (P137)), and the remainder of the file system can be browsed using FTP.

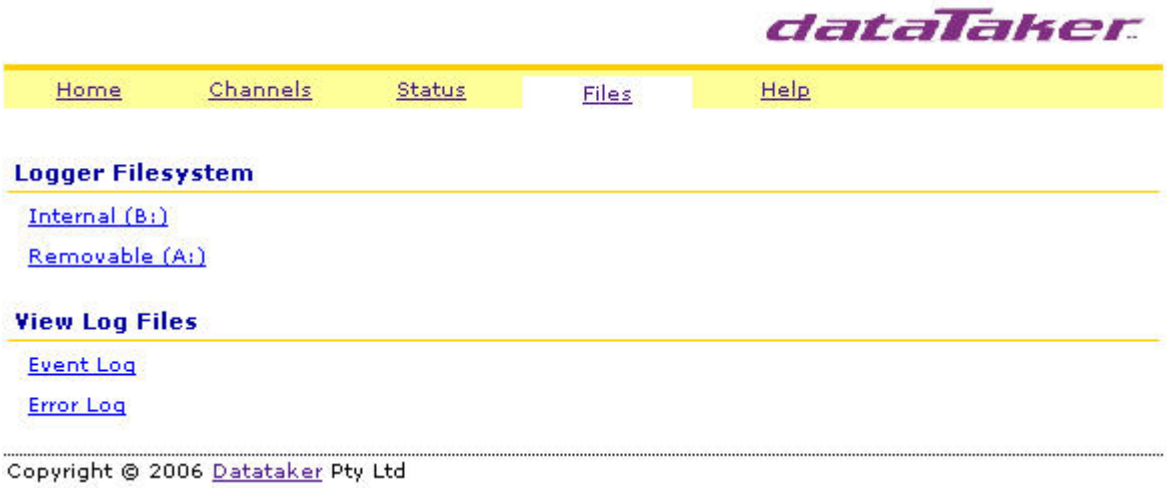


Figure 27: Files Page

To view a log file, simply click on the desired log file link. The log file will then be displayed. Click on the Back link to return to the Files page.

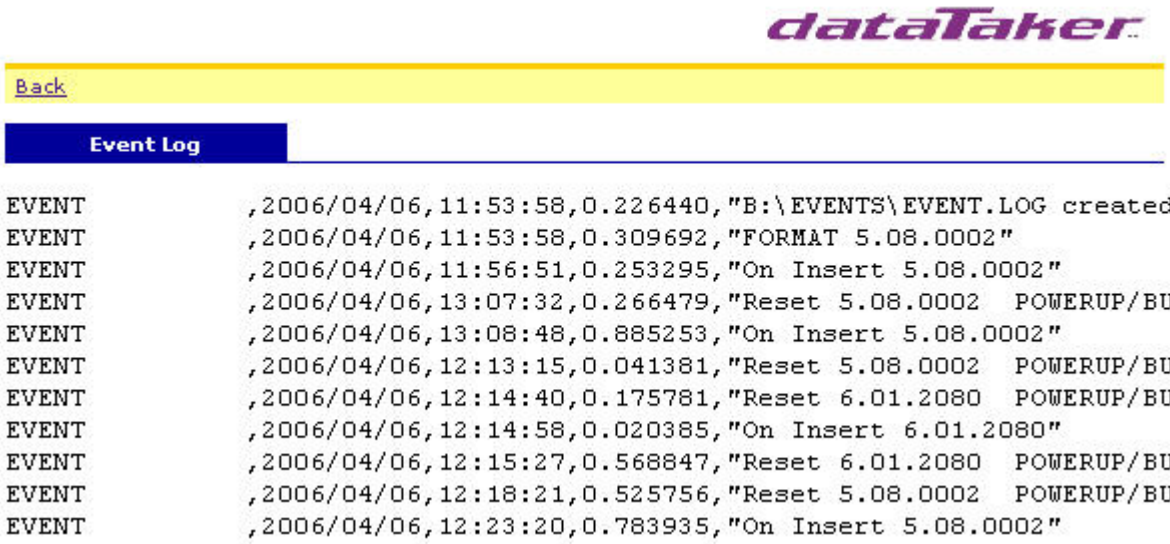


Figure 28: Event Log Page

To access files stored on the *DT80*'s internal file system, or in a connected USB memory device, click on the desired FTP link. This will then display the directory listing of the drive. Clicking on a file link will then initiate an FTP download of the selected file.

Using these links is equivalent to typing an **ftp://** URL into the browser as described in *Using the DT80 FTP Server* *(P126)*. Note however that the web interface links provide read-only (anonymous) access only.

Click on the web browser's Back button to navigate back to the Files page.

Help Page

The Help page provides troubleshooting and help information, and a link to the Technical Support web page on dataTaker's website.

There is also a link to the *DT80* User's Manual (PDF format), which is normally pre-loaded onto the *DT80*'s internal file system in the **B:\manual** subdirectory. If the link does not work, verify that the directory and file are present. (Be aware that if the **FORMAT"B:"** command is ever issued, it will delete the manual, along with everything else on **B:**.) If required, the manual (and any other files that you would like to store on the logger) can be loaded back onto the logger using FTP.

Customising the Web Interface

This section describes some of the technical features of the *DT80* web interface. These allow advanced users to replace the built-in web interface with a customised web user interface.

Web Application Programming Interface (API)

The *DT80* provides an application programming interface (API) so that you can build custom web pages that can view and display data from the logger. The API consists of a set of **server-side include** (SSI) directives. SSI directives are placed in HTML pages, and evaluated on the logger when the HTML page is requested. HTML pages that contain SSI directives are known as **SHTML** pages, and typically have a **.shtml** file extension.

When an SHTML page is requested, it is scanned by the web server (logger) for these directives. Once found, the logger interprets the directive and performs the required action. The output is then sent as part of the response back to the web browser.

Server-Side Include (SSI) Directives

An **SSI directive** consists of a special sequence of characters which is placed within an HTML page. The format is as follows:

<!--#*directive attribute***="***value***"** *attribute***="***value***"** ... **-->**

where:

- **<!--#** and **-->** are the opening and closing identifiers that must be specified when applying an SSI directive.
- *directive* is the name of the directive to be executed.
- *attribute* is the name of an **attribute**, and *value* is the value it is set to. Each SSI directive has a set of valid attributes that can be specified to control the operation of the directive. One or more attribute-value pairs can be specified.

For example

```
<!--# echo var = "1CV" -->
```
inserts the SSI directive named **echo**, which contains one attribute **var** whose value is set to **1CV**.

DT80 *SSI Directives*

The *DT80* web server supports five SSI directives, which are summarised in the table below. Each directive requires the indicated attribute to be set. In addition, one or more optional attributes may be included.

Directive	Required Attribute	Function
echo	**var**	Inserts the current value of the indicated variable
channeltable	**schedule**	Inserts an HTML table containing, for each channel, its schedule, name, most recent value, units and timestamp – similar to the **Channels** page
measure	**channel**	Samples the indicated channel (certain channel types only) and inserts the measured value plus timestamp
reading	**channel**	Inserts the most recent value of the indicated channel
include	**file** or	Inserts the contents of the indicated text file, specified as a path relative to the document root (**DOC_ROOT** profile setting
	virtual	Inserts the contents of the indicated text file, specified as an absolute path

Optional Attributes

One optional attribute is supported, which may be applied to any of the above directives in addition to their standard attribute:

Directive	Attribute	Function
any	**cond**	Evaluate the directive if and only if the indicated condition is true

The following sections discuss each directive in more detail.

#echo Directive

This directive inserts a specific piece of information into the HTML page.

SSI Directive	Description
<!--#echo var = "D" -->	Inserts the current date. eg. 2006/05/02.
<!--#echo var = "T" -->	Inserts the current time. eg. 11:45:23.
<!--#echo var = "dtmodel" -->	Inserts the model number of the logger. eg. DT80.
<!--#echo var = "*n*CV(FF*d*)" -->	Inserts the value of channel variable *n*CV. The **(FF*d*)** part is optional, and specifies the number of decimal places to display (default is one decimal place) eg. 23.4.
<!--#echo var = "*n*SV(FF*d*)" -->	Inserts the value of system variable *n*SV. The **(FF*d*)** part is optional, and specifies the number of decimal places to display (default is no decimal places) eg. 23200
<!--#echo var = "JobName" -->	Inserts the name of the current running job (or no current job if none). eg. MYJOB.
<!--#echo var = "SchName(*s*)" -->	Inserts the schedule name associated with schedule *s*. eg. SchWebA.
<!--#echo var = "SchTrigger(*s*)" -->	Inserts the trigger string for schedule *s*. eg. 1S.
<!--#echo var = "SchStatus(*s*)" -->	Inserts the run status (active/halted) for schedule *s*. eg. active.
<!--#echo var = "SchLogState(*s*)" -->	Inserts the logging state (enabled/disabled) for schedule *s*. eg. disabled.
<!--#echo var = "SchDataStoreSize(*s*)" -->	Inserts the number of the logged data records for schedule *s*. eg. 2001.
<!--#echo var = "SchAlarmStoreSize(*s*)" -->	Inserts the number of the logged alarms for schedule *s*. eg. 2301.
<!--#echo var = "SchDataStartTime(*s*)" -->	Inserts the timestamp of the earliest logged data record for schedule *s*. eg. 2006/05/02 14:15:12.
<!--#echo var = "SchAlarmStartTime(*s*)" -->	Inserts the timestamp of the earliest logged alarm for schedule *s*. eg. 2006/05/02 15:15:12.
<!--#echo var = "SchDataEndTime(*s*)" -->	Inserts the timestamp of the latest data record for schedule *s*. eg. 2006/05/02 11:15:12.

<!--#echo var = "SchAlarmEndTime(s)" -->	Inserts the timestamp of the latest alarm for schedule *s*. eg. 2006/05/02 14:13:12.

#channeltable Directive

This directive inserts a table of channel values.

SSI Directive	Description
<!--#channeltable schedule = "" -->	Inserts an HTML table containing a header row, plus a row for each defined channel in the current job, excluding working channels and immediate channels. Each row contains the following columns: • schedule identifier (A – K, X) • channel name • most recent value of the channel • units string • time at which most recent measurement was taken
<!--#channeltable schedule = "*s*" -->	As above, but only channels belonging to schedule *s* are included.

Sample HTML output generated by the **#channeltable** directive is as follows:

```
<table class="jdt" cellspacing="0">
 <colgroup>
        <col class="sid"/>
        <col class="sun"/>
        <col class="chv"/>
        <col class="dfu"/>
        <col class="tsp"/>
 </colgroup>
 <thead>
  <tr>
    <th>Schedule</th>
    <th>Name</th>
    <th>Value</th>
    <th>Units</th>
    <th>Timestamp</th>
  </tr>
 </thead>
 <tbody>
  <tr>
    <td>A</td>
    <td>1V</td>
    <td>1.234</td>
    <td>mV</td>
    <td>2006/04/07 12:12:11</td>
  </tr>
  <tr>
    <td>A</td>
    <td>Geyser Temp</td>
    <td>122.8</td>
    <td>degC</td>
    <td>2006/04/07 12:12:11</td>
  </tr>
  <tr>
    <td>C</td>
    <td>Gravy Press</td>
    <td>69.9</td>
    <td>MPa</td>
    <td>2006/04/05 12:42:01</td>
  </tr>
 </tbody>
</table>
```

Note that the CSS (Cascading Style Sheet) class ids in the table and colgroup tags have been used to style the table. To

change the style (colours, spacing, etc.), create another CSS file, reusing the same class ids. By default, the table will be displayed without any styles applied.

#measure Directive

This directive is used to perform a input channel measurement. This is executed in the Immediate schedule and the data gathered is not logged.

SSI Directive	Description
<!--#measure channel = "*chan-def***" -->**	Evaluates the specified *DT80* channel definition, eg. **2R(4W)**, as an immediate channel, waits for it to complete, then inserts an HTML fragment containing the measured value and timestamp eg: <pre> 23.9 2006/05/07 12:13:12 </pre>

#reading Directive

This directive is used to return the most recent reading for the specified channel. The channel is assumed to have been already defined in a schedule in the current job.

SSI Directive	Description
<!--#reading channel = "*chan-def***" -->**	Inserts the most recent reading for the specified *DT80* channel definition. eg. 44.0

#include Directive

This directive is used to insert the contents of another file into a HTML page.

SSI Directive	Description
<!--#include file = "*rel-file***" -->**	Inserts the contents of the file *rel-file* (specified as a relative path, eg. **footer.htm**) into the current HTML page.
<!--#include virtual = "*abs-file***" -->**	Inserts the contents of the file *abs-file* (specified as an absolute path, eg. **b:\events\event.log**) into the current HTML page.

cond Attribute

This attribute may be included by any SSI directive, in addition to its normal attribute. It specifies a **condition** – if the condition is met then the SSI directive is processed, otherwise it is ignored. This provides a way to conditionally include web page elements based on the status of the *DT80*.

condition value	Description
cond = SchDefined(*s***)**	Process directive if the specified schedule has been defined
cond = DataStored(*s***)**	Process directive if any data have been logged for the specified schedule
cond = AlarmsStored(*s***)**	Process directive if any alarms have been logged for the specified schedule

For example:

```
<!--#include file = "schedA.shm" cond = "SchDefined(A)" -->
```

will only include the indicated file if schedule A is defined in the current job.

Building A Custom Web Page

This section provides a brief overview of the process of setting up a custom web page for the *DT80*. It is assumed that the reader has a good working knowledge of HTML and the *DT80*.

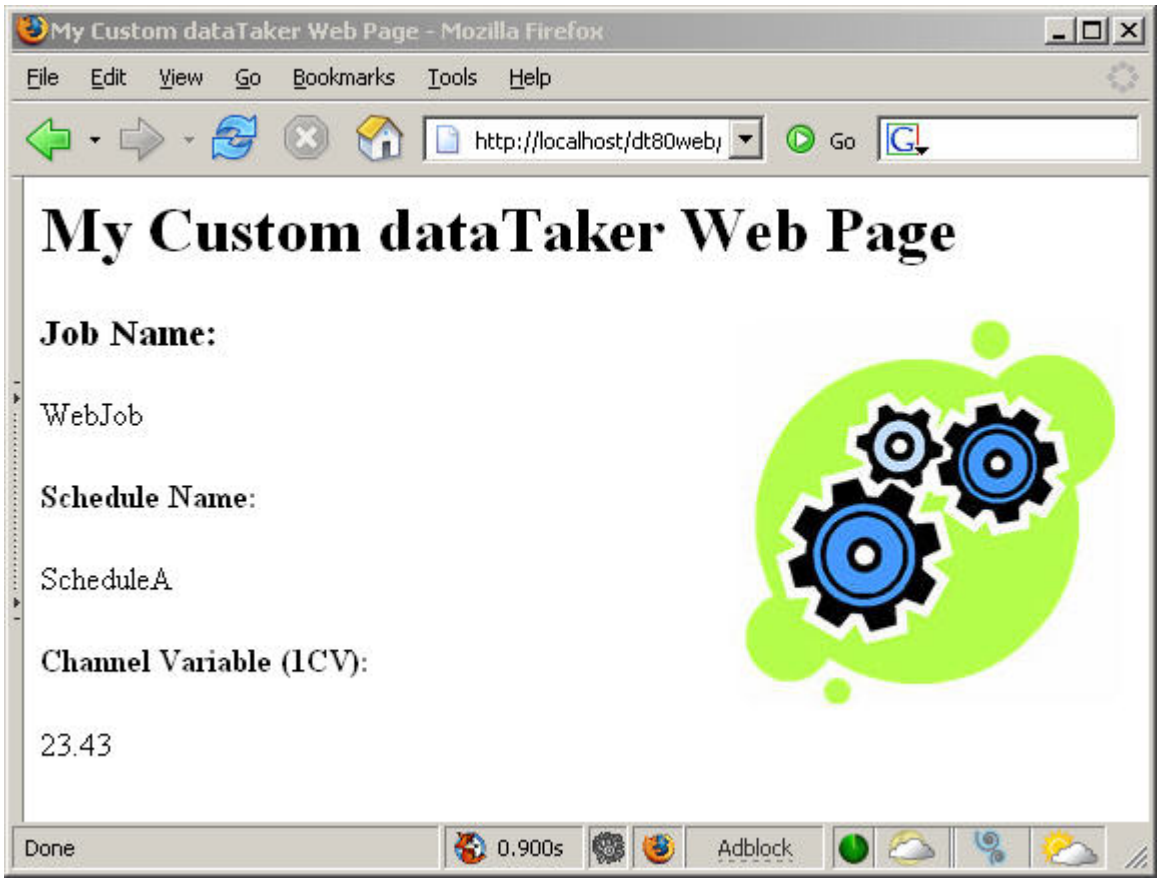
All custom pages will need to be loaded into a directory of your choice on the *DT80*'s internal file system (**B:**).

Creating the SHTML Page

An SHTML page can be created with any HTML or text editor. Let's create a page that is titled "My Custom dataTaker Web Page" that will contain an image and display the following logger data:-

- The current job name
- The name of schedule A for the current job
- The value from channel variable 1

This is what the SHTML page will look like when viewed in a web browser:-



The SHTML mark-up for this page is as follows:

```
<html>
<head>
<title>My Custom dataTaker Web Page</title>
</head>
<body>
<h1>My Custom dataTaker Web Page</h1>
<img src="custom1.jpg" align="right"/>
<h3>Job Name:</h3>
  <!--#echo var = "JobName" -->
<h4>Schedule Name:</h4>
  <!--#echo var = "SchName(A)" -->
<h4>Channel Variable (1CV):</h4>
  <!--#echo var = "1CV" -->
</body>
</html>
```

Notice the SSI directives (red).

Note The SHTML page filename must be saved with the following extensions:- **.shtml**, **.shm** or **.sht**. The SHTML page will not be rendered correctly if these extensions are not used.

Custom Home Page

It is recommended that a central (Home) page is used when building a custom web interface. The Home page can be an HTML or SHTML page. Set the Home page filename to either **index** or **default** (with the appropriate file extension)

The *DT80* web server will automatically looks for a file with one of these names if the user types just the IP address into their browser (ie. no filename specified).

Storing the Custom Web Page

To test the custom web pages they need to be loaded onto the *DT80*'s internal file system.

Connect to the logger's FTP server, specifying the configured username and password – files cannot be written to the file system using the default anonymous username. Then upload the files to a directory on the internal drive, eg. **B:\custom_www**.

Note Be aware that any web pages loaded onto the internal file system will be deleted if the **FORMAT"B:"** command is ever

used.

Profile Settings

The following profile settings *(P133)* are used to configure the *DT80* web server. These settings should be placed in the **HTTP_SERVER** section.

Key name	Default value	description
DOC_ROOT	C:\www\	The location of the web pages on the logger. The default setting points to the built-in pages located on the *DT80*'s ROM disk (**C:**).
PORT	80	The port number used for processing HTTP. Set to zero to disable the web server.

For example, to set the web server to look for web pages in the **b:/myweb** directory you need to enter the following commands:

```
PROFILE "HTTP_SERVER" "DOC_ROOT"="B:\\myweb\\"
SINGLEPUSH
```

The double backslash is necessary because the *DT80* interprets a sequences starting with a backslash, within a quoted string, as a special character. Remember also that if **DeTransfer** is used to enter these commands then it too requires any backslashes to be "escaped" by entering a double backslash. So the above string would need to be entered into the DeTransfer send window as **"B:\\\\myweb\\\\"**.

Customising the Built-in Web Interface

As an alternative to creating the web interface from scratch, you can also use the built-in web pages as a starting point and customise them as required. These pages are accessible on the logger's built-in "ROM disk" in the directory **C:\WWW**.

Note, however, that these files have been compressed and are difficult to read. The equivalent uncompressed files are available on the dataTaker resource CD in the "DT80 Firmware" directory, and are included in firmware packages downloaded from the dataTaker web site.

These files can then be loaded into a directory on the *DT80*'s **B:** drive, and the **DOC_ROOT** profile setting changed to point to them. This should provide identical functionality to the built-in interface. You can now customise one or more of the pages as required.

Part J — Modbus Interface

About Modbus

Modbus is a simple communications protocol which is widely used in **SCADA** (supervisory control and data acquisition) systems. Modbus provides an efficient and standardised 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 can operate using a broad range of communications media. These fall into two main categories:

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

The *DT80* is capable of operating as a Modbus server; that is, it can act like an RTU or PLC device. This allows the *DT80* to be easily integrated into any Modbus-based SCADA system. No special drivers are required for the client system.

A Modbus client system can directly read or write any *DT80* channel variable (CV) or digital I/O channel.

Note that the *DT80* does not currently implement the master side of the Modbus protocol, ie. it cannot operate as a Modbus client.

In general terms, the procedure for setting up the *DT80* in a Modbus environment is:

1. Establish a physical connection (TCP/IP or serial) between the Modbus client system and the *DT80.*
2. Load a job onto the *DT80* that scans the required channels at the required rates. The job should also load the measured values into channel variables.
3. Configure the client system to poll the Modbus addresses corresponding to the *DT80* CVs and digital I/Os of interest.

Note Even if there is no job loaded, the *DT80*'s Modbus server is still active and the client can query or set any CV or digital channel.

Connecting to a Modbus Network

The *DT80* supports both TCP/IP and serial Modbus networks. You can connect the *DT80* to one or more of the following:

- a TCP/IP-based Modbus network, using the *DT80* Ethernet interface.
- a TCP/IP-based Modbus network, using a PPP connection to the *DT80* host RS232 port.
- a point-to-point or multi-drop serial Modbus network (RS485, RS422 or RS232), using the serial sensor port (not available on DT81). Note that this will prevent the serial sensor port being used for controlling other serial sensors.
- a point-to-point serial Modbus network, using the host RS232 port. Note that this will prevent the host port being used for the *DT80* command interface – you will need to use the USB or Ethernet interface to send commands to the *DT80.*

TCP/IP Connection

Up to five Modbus client systems 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. See *Ethernet Communications* (P122) and *PPP Communications* (P127) for more details.

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). If required, this port number may be changed using the following *DT80* command:

```
PROFILE "MODBUS_SERVER" "TCPIP_PORT"="port"
```

where *port* is the desired port number (1-65535). As with any startup profile setting, it is necessary to reset the *DT80* (eg. using **SINGLEPUSH**) in order for the setting to take effect.

To disable the *DT80*'s TCP/IP Modbus server, set the port number to zero, ie.

```
PROFILE "MODBUS_SERVER" "TCPIP_PORT"="0"
```

Serial Connection

A serial Modbus network has one client (master) system connected to one or more server (slave) devices. Serial networks using the RS485 or RS422 standards support multi-drop, ie. 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. (Slave addresses are not required on a TCP/IP Modbus network, because the slaves are identified by their IP address.)

The *DT80* can be connected to a serial Modbus network using either the serial sensor port (see *Serial Channel* (P166)), or

the host RS232 port. It can even be connected to two separate serial networks, using both ports.

By default, Modbus is disabled for both the host and serial sensor ports. Unlike TCP/IP, if a serial port is configured for Modbus then it cannot be used for its normal purpose. For example, if the serial sensor port is used for Modbus then it cannot be used for controlling serial sensor devices, ie. any **1SERIAL** channels will be inoperative. Similarly, if the host port is configured for Modbus then it will no longer be possible to send *DT80* commands via the host RS232 port.

Slave Address

To enable Modbus operation on the serial sensor port, all that is required is to set the *DT80*'s slave address. This is done using the following profile setting:

```
PROFILE "MODBUS_SERVER" "SERSEN_ADDRESS"="addr"
```

where *addr* is the desired address (1-247). Setting the address to zero (which is the default) will disable Modbus on the serial sensor port.

In the same way, Modbus can be enabled on the host RS232 port using:

```
PROFILE "MODBUS_SERVER" "HOST_ADDRESS"="addr"
```

Again, the default setting is zero, which means "disabled".

Serial Parameters

If Modbus is enabled on a serial port, its serial parameters will be set to the standard Modbus default settings, which are **RS232,19200,8,E,1** (RS232, 19200 baud, 8 data bits, even parity, 1 stop bit).

To change these settings, use the **PS=** or **PH=** commands. For example, if you are using the serial sensor port to connect to an RS485 network running at 9600 baud you should include the following command in your *DT80* job:

```
PS=RS485,9600
```

Note that the host RS232 port only supports RS232; the serial sensor port supports RS232, RS422 or RS485.

Note also that any host port parameters specified in the **HOST_PORT** section of the startup profile will be ignored if the host port is configured for Modbus. The only way to override the default Modbus settings for the host port is to use the **PH=** command.

Modbus Registers

The Modbus Data Model

The Modbus protocol defines a simple data model. It specifies that any Modbus slave device contains the following resources:

- an array of single bit **coils** (digital outputs). When setting up a Modbus client application, a particular coil is normally referenced using a 5-digit number in the range 00001-09999, or a 6-digit number 000001-065536 (depending on the Modbus client implementation). In this manual the 5-digit notation will be used.
- an array of single bit **discrete inputs** (digital inputs), numbered 10001-19999, or 100001-165536
- an array of 16-bit **input registers**, numbered 30001-39999, or 300001-365536
- an array of 16-bit **output registers** (a.k.a **holding registers**), numbered 40001-49999, or 400001-465536

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. Some Modbus client applications reference registers using these raw protocol addresses. The documentation for the particular package should make clear which convention it uses.

The 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.

Accessing DT80 *Channels via Modbus*

The *DT80* maps blocks of Modbus registers onto certain *DT80* **channels** (channel variables and digital channels), as specified in the following tables. The Modbus client can therefore directly access any of these *DT80* channels by transmitting a request to read or write the associated Modbus register(s).

The first table shows the action taken by the *DT80* in response to requests by the client system to read particular Modbus registers:

Register number as specified in Modbus client application	Type of register to read	Action taken by *DT80*
00001-00800 10001-10800	coil 1-800 discrete 1-800	returns current state of channel variable 1..800CV (0 if CV value is 0.0, otherwise 1)
30001-30800 40001-40800	input reg 1-800 output reg 1-800	returns current value of channel variable 1..800CV
08001-08009	coil 8001-8009	returns current state of digital output 1..8DSO or 1RELAY
18001-18008	discrete 8001-8008	returns current state of digital input 1..8DS
38001-38009	input reg 8001-8009	returns current state of digital input 1..8DS as a numeric value (0 or 1)
48001-48008	output reg 8001-8009	returns current state of digital output 1..8DSO or 1RELAY as a numeric value (0 or 1)

The next table shows the action taken by the *DT80* in response to a write request:

Register number as specified in Modbus client application	Type of register to write	Action taken by *DT80*
00001-00800	coil 1-800	sets channel variable 1..800CV to 0.0 or 1.0
40001-40800	output reg 1-800	sets channel variable 1..800CV to the specified value
08001-08009	coil 8001-8009	sets digital output 1..8DSO or 1RELAY to the specified value
48001-48008	output reg 8001-8009	sets digital output 1..8DSO or 1RELAY to the specified value (0 if the specified value is 0, otherwise 1)

If the Modbus client attempts to access any register outside the ranges specified above then the *DT80* will return a Modbus error response and ignore the request.

Note that for the DT81, Modbus registers x8001-x8004 correspond to channels 1..4DS and 1..4DSO. Accessing registers x8005-x8008 will not cause an error, but it will not do anything either, because these channels are not present on the DT81. Register x8009 corresponds to the 1RELAY channel, as with the DT80.

Data Types

Modbus input and output registers are 16 bits wide. The Modbus standard does not, however, define how these bits are to interpreted, other than stating that the most significant byte of a register value is transmitted first ("big endian" format).

For data values that cannot be represented by a 16-bit integer value (-32768 to 32767), there are a number of options:

- the register can be treated as an unsigned 16-bit integer (0-65535)
- the value can be scaled, typically by a power of ten, to give the required precision or range. For example a scaling factor of 100 would permit values in the range -327.68 to 327.67 to be returned.
- multiple registers can be combined to return a single larger value, eg. a pair of registers could return a 32-bit quantity.

Clearly, both the slave device and the client system must agree on how a given Modbus register is to be interpreted. It is no good if the device encodes the value 40000 as an unsigned 16-bit number (9C40 hexadecimal) but then the client interprets it as a signed number and displays it as -25536.

The only solution is to explicitly configure the required data types on both the slave and the client.

The SETMODBUS Command

By default, all CV values are transferred to and from the *DT80* as signed 16-bit integers, with no scaling factor. The **SETMODBUS** command is used to specify alternative data types and scaling factors.

The format of the command is as follows:

SETMODBUS *channels format scaling*

where:

- *channels* specifies a single channel variable, or a range (eg. **1CV** or **20..29CV**)
- *scaling* is an optional floating point scaling factor by which the channel value will be multiplied before being returned. Conversely, when the client writes a value, it will be divided by the scaling factor before being written to the CV.
- *format* is an optional code that specifies the data type, as follows:

Format code	Data type	Comments
MBI	signed 16-bit integer	Default setting. Returns -32768 or 32767 if the scaled return value is outside the valid range.
MBU	unsigned 16-bit integer	Returns 0 or 65535 if the scaled return value is outside the valid range.
MBL	signed 32-bit integer	Upper 16 bits of *n*CV are returned in Modbus register *n*. Lower 16 bits are returned in register *n*+1. Returns -2,147,483,648 or 2,147,483,647 if the scaled return value is outside the valid range.
MBF	32-bit floating point	Returned as a single precision IEEE-754 floating point number. Lower 16 bits of *n*CV are returned in Modbus register *n*. Upper 16 bits are returned in register *n*+1.

If *format* and *scaling* are not specified, the current settings for the indicated range of CVs are displayed.

Any number of these **SETMODBUS** commands can be issued (typically at the start of the *DT80* job) to configure the required channels.

Example

This example illustrates some of the technicalities relating to Modbus transfers.

Consider the following job:

```
BEGIN"PERCY"
SETMODBUS 7CV MBF
SETMODBUS 9..10CV MBU 100
SETMODBUS 11CV MBL
7CV=23.91  8CV=42  9CV=490.22  10CV=921.0  11CV=75535.9
END
```

If a Modbus client then requests input registers 30007-30012 it will receive the following raw data:

Register	Value (hex / decimal)	Comments
30007	47AE	The 32-bit value 41BF47AE is the IEEE754 representation of 23.91
30008	41BF	
30009	BF7E / 49022	490.22 multiplied by scaling factor (100)
30010	FFFF / 65535	overflow: 921.0 x 100 = 92100 is too big for an unsigned 16-bit integer
30011	0001 / 1	The 32-bit value 00012710 equals 75536 decimal
30012	2710 / 10000	

To make sense of this, the client software must support the 32-bit data formats used by the *DT80*, and it must be told the data type of each register (or register pair).

Note Be aware that for 32-bit data types, the word order (ie. whether the upper or lower 16 bits comes first) is not proscribed by the Modbus standard, so naturally both orderings are widely used. Most client software can be configured to support either ordering.

Note that in this example 7CV and 11CV are spread across two registers apiece (30007-8 and 30011-12 respectively, which makes channel variables 8CV and 12CV effectively inaccessible via Modbus. They can still be used in the program (eg. setting **8CV=42** in the above program is perfectly OK and won't interfere with 7CV); it's just that they cannot (easily) be accessed by a Modbus client.

Note also that some Modbus clients require that all 32-bit register pairs start on an odd-numbered register. That is, you can return a 32-bit value using registers 30001-30002, but not using registers 30002-30003.

Putting It All Together

In this example, a greenhouse is monitored by a DT80. Three thermocouples on analog inputs **1-3** monitor the temperature at various points. The DT80 is required to log the temperatures every 15s and switch on an extractor fan (by setting digital output **1D** low) if any of the temperatures exceed a programmable setpoint. A sensor attached to the fan produces a voltage proportional to fan speed (1.25mV/rpm) and this is fed into analog input **4**. Digital output **2D** is connected to a watering system valve.

The DT80 is connected to an Ethernet network. In a central office an operator runs a Modbus-capable SCADA package. She wants to be able to:

- check current temperatures
- set the current extractor fan setpoint
- check the status of the extractor fan (control output state and fan RPM)
- switch the watering system on or off
- check how many data and alarm records have been logged

The following sections discuss how this might be achieved.

DT80 Configuration

It is assumed that the *DT80*'s Ethernet connection has already been set up. This can be verified by entering its IP address into DeTransfer or a web browser and checking that the *DT80*'s command or web interface is accessible. If you cannot

connect to these services then you probably won't be able to connect to the Modbus server either. See *Ethernet Communications* (P122) for more information.

Once TCP/IP connectivity has been established, a suitable *DT80* job can be entered:

```
BEGIN"GERANIUM"
SETMODBUS 1..4CV MBI 10 ' temperatures & setpoint
SETMODBUS 5..7CV MBL 1  ' logged data,alarm recs (32 bit)
SETMODBUS 9CV    MBI 1  ' fan RPM
4CV(W)=30.0 ' default setpoint
' update every 15 sec
RA(DATA:30D,ALARMS:500KB)15S
 1TK(NR,=1CV)
 2TK(NR,=2CV)
 3TK(NR,=3CV)
 ' set 10CV=1 if at least one temp over limit
 ' set 11CV=1 if all temps under limit
 ' allow +/- 1 degC hysteresis
 10CV(W)=(1CV>4CV+1)OR(2CV>4CV+1)OR(3CV>4CV+1)
 11CV(W)=(1CV<4CV-1)AND(2CV<4CV-1)AND(3CV<4CV-1)
 ALARM1(10CV>.5)"Temp>?4 Fan ON^M"{1DSO=0}
 ALARM2(11CV>.5)"Fan OFF^M"{1DSO=1}
 32SV(W,=5CV)  ' num logged data recs for sched A
 33SV(W,=7CV)  ' num logged alarm recs for sched A
 4V(W,0.8,=9CV)  ' fan speed 1.25mV/rpm
LOGONA
END
```

This job sets up the following CVs for access by the Modbus client:

- **1CV**, **2CV** and **3CV** contain the three measured temperatures. A scaling factor of 10.0 is applied so that it can be returned with one decimal place (range -3276.8 to +3276.7)
- **4CV** is designed to be accessed as an output register, ie. the Modbus client writes to it. This channel variable is used as the temperature setpoint, so it is scaled in the same way as the other temperature CVs.
- **5CV** and **7CV** return the number of logged data and alarm records for the main A schedule. In this example the store file size is relatively large (30 days data @ 15s scans), so the number of logged records may exceed the capacity of a single 16-bit register. They are therefore defined as 32-bit long integers (**MBL**). (Channel variables **6CV** and **8CV** have been skipped because their associated Modbus registers are used for returning the 32-bit 5CV and 7CV values.)
- **9CV** returns the fan speed in RPM. A standard 16-bit integer is OK here. (The **SETMODBUS** command for this CV could have been omitted because the values it is setting are the default values.)

Notice that all Modbus-accessible CVs have been grouped into one contiguous block. This is not essential, but it will improve the efficiency of the Modbus link because the client can request all relevant CVs in one command.

The digital channels of interest are **1DSO** (fan control) and **2DSO** (watering system). Since the Modbus client can access these channels directly, there is no need to include them in the *DT80* job.

One final point is that because all relevant data are returned by Modbus, there is no point having the *DT80* return real time data via its standard command interface – it would just clutter the screen if the operator ever needed to connect to the *DT80* using Detransfer. All channels are therefore set to "working" (**W**) or "no return" (**NR**) channels.

Modbus Client Configuration

It is now necessary to configure the SCADA software package to suit the *DT80* channel usage described above. This is highly application dependent but in very general terms the steps involved will typically include:

- configuring communications details
- designing a mimic screen incorporating the required measurement and control fields
- associating each element of the mimic with the correct Modbus register address

So in this case the first step might be to select the software package's "generic PLC" device driver and then create a "DT80" device instance. As a minimum, the *DT80*'s IP address would need to be entered here. If the driver or application provides a "test" facility, you may at this point be able to try manually reading and writing specific Modbus registers.

The mimic screen for this application might consist of:

- three thermometer displays showing the measured temperatures
- an entry field where the operator can set the desired setpoint
- two numeric indicators showing number of logged data and alarm records
- a fan icon with on/off and RPM indicators
- a button to turn the watering system on or off.

The final step would typically then be to edit the properties of each control and indicator to specify their behaviour. This would generally mean specifying:

- which slave device to use. This may involve selecting the *DT80*'s "device instance" from a list of connected Modbus slave devices.
- the Modbus register number
- the data type (signed or unsigned or floating point, 16 or 32 bits, byte/word ordering)
- the scaling factor. This is often presented as a **span**, similar to a *DT80* span *(P58)*. That is, you specify the "device range" (min/max returned value) and the corresponding "display range" (min/max value to display) – which then creates a linear scaling curve.
- the scan rate (how often to read/write the value from/to the *DT80*)
- other details such as units.

So the setup for this application might be something like:

Mimic Element	Modbus Reg	Data Type	Device Range	Display Range	Units	Comments
"temp1"	30001	signed 16-bit	-3276 to 3276	-327.6 to 327.6	degC	1CV (input reg)
"temp2"	30002	signed 16-bit	-3276 to 3276	-327.6 to 327.6	degC	2CV (input reg)
"temp3"	30003	signed 16-bit	-3276 to 3276	-327.6 to 327.6	degC	3CV (input reg)
"setpt"	40004	signed 16-bit	-3276 to 3276	-327.6 to 327.6	degC	4CV (output reg)
"num_data"	30005	signed 32-bit, MSW first	-1 to 1000000	-1 to 1000000	recs	5CV (input reg)
"num_alm"	30007	signed 32-bit, MSW first	-1 to 1000000	-1 to 1000000	recs	7CV (input reg)
"fan_rpm"	30009	signed 16-bit	0 to 32767	0 to 32767	rpm	9CV (input reg)
"fan_state"	08001	n/a	n/a	n/a	on/off	1DSO (coil)
"water"	08002	n/a	n/a	n/a	on/off	2DSO (coil)

Notice how input register numbers are prefixed by "3", output registers by "4" and coils (output bits) by "0".

Part K — Communications

The *DT80* provides four types of interfaces through which it can communicate with a host computer:

- a **command interface**, to which a host computer sends ASCII commands and from which it receives ASCII data and other responses. All of the various commands and responses described in this manual are sent to the *DT80* via its command interface.
- a **web server**. This allows the *DT80* to be monitored remotely using a standard web browser, such as Microsoft Internet Explorer. See *Web Interface* (P98).
- an **FTP server**. The File Transfer Protocol is a standard TCP/IP based protocol, which allows files to be efficiently transferred between the *DT80*'s file system and that of a host computer. See *Using the DT80 FTP Server* (P126).
- a **Modbus server**. This allows the *DT80* to be monitored and controlled by a Modbus client system (typically a SCADA package). See *Modbus* Interface (P107).

These "conceptual" interfaces obviously all require some kind of physical data connection between the *DT80* and the host computer(s).

The remainder of this section will deal with:

- selecting the most appropriate physical communications link, and how to setup and use it.
- specific details on how the command interface and FTP interface operate.

The Command Interface

Physical Interfaces

The *DT80* has a number of different **physical comms interfaces**, any of which can be used as the command interface. You can therefore send commands from a computer to the *DT80* via:

- a direct **USB** connection (P114)
- a direct **RS232** connection, using a crossover ("null modem") cable (P118
- a fixed or dial-up **modem**, which is connected to the *DT80*'s RS232 port (P118).
- a **TCP/IP** network, which is connected to the *DT80*'s **Ethernet** port (P122).
- a **TCP/IP** network, which is connected to the *DT80*'s RS232 port (using **PPP**) (P122).

Arbitration

The *DT80* can have more than one type of communications link connected at the same time. In this situation, the *DT80* automatically switches between each link as required, responding back through the link from which the most recent communication was received.

You can therefore switch to a new comms interface at any time, simply by sending a *DT80* command (or just a carriage return character) via that interface.

Broadcasting Data

Up to ten different computers can be simultaneously "connected" to the *DT80* using TCP/IP. These can be thought of as ten separate comms interfaces.

Although there can only be one active comms interface at any one time, it is possible for the *DT80* to "broadcast" data to a number of computers simultaneously.

All output text generated by the *DT80* (command echoes, messages, returned/unloaded data etc.) is sent to:

- the active comms interface (which may be RS232, USB or TCP/IP), **and**
- all currently open TCP/IP connections (if any)

In this way a number of computers can be connected to the *DT80* via a TCP/IP network and passively "listen" to the stream of returned data generated by the *DT80*.

Command Interface Operation

Characters received via the command interface are **buffered** until a carriage return (CR) character is received. This buffer can hold 255 characters, so this is the maximum line length that can be sent to the *DT80*.

Once a CR is received, the *DT80* will:

1. **wait** until any currently executing schedule completes
2. **echo** the received command line (after converting it to uppercase). Command echo can be disabled using the **/e**

switch command.

3. **process** the command.
4. output the *DT80*> **prompt**, to indicate it is ready for the next command.

Note that it is not necessary to wait until a command completes before sending the next command, as the *DT80* provides additional buffering for subsequent command lines. Each type of comms channel provides some form of automatic **flow control** to ensure that these buffers do not overflow when a large number of command lines are sent at once.

Detecting *DT80* Presence

Host software can detect the presence of a *DT80* by sending a DEL character (ASCII 127). If this character is received at any time, the *DT80* will respond with << followed by CR LF. The DEL character is always recognised and responded to, even if a password has been applied (see below).

Password Protection

To reduce the possibility of unauthorised access to the *DT80*'s command interface, you can configure a **password**, so that communication is only possible after the password is entered.

Setting and Removing the Command Interface Password

Set a password by sending
PASSWORD="*password***"**
to the *DT80*. *password* can be any text string of up to 10 case-sensitive characters.

Remove a password by sending
PASSWORD=""

Note The password is cleared if the *DT80* performs a hardware or **SINGLEPUSH** reset.

Accessing Password-Protected Command Interface

To establish communication at anytime, simply send the password followed by a carriage return. If the password is correct, the *DT80* responds with Accepted and opens the comms port.

The ports stay open until you send the **SIGNOFF** command, or while there is comms activity. If there is no communication for a period of time defined by P14 (default is 300 seconds), the ports time out and close.

Is the Command Interface Protected?

Simply send the command
PASSWORD
to determine if a command interface password has been set. The *DT80* responds with 1 if a password has been set, otherwise 0.

USB Communications

The *DT80*'s USB interface operates as a "virtual COM port". With the appropriate drivers installed on the PC, the USB link will, when connected, appear to the PC as an additional COM port.

Installing the USB Driver

When the *DT80* is first connected to a PC via USB, Windows will search for a suitable driver.

If no driver is installed on the PC, the Windows "new hardware wizard" will run. The exact sequence of events that follows varies somewhat depending on the version of Windows.

Normally, Windows will ask where it should look for a driver. Insert the dataTaker resource CD and select the "CDROM drive" as the driver location when prompted.

The New Hardware wizard should then be allowed to run to completion. You should now have an additional COM port available on the PC. We now need to determine what COM port number has been assigned.

When we use DeLogger, the particular COM port in the Connection dialog drop down list can be identified by the **FTDI** manufacturers tag.

Both DeLogger and DeTransfer only show **active** COM ports in their connection dialogs, so it should not be too difficult to work out which one corresponds to the USB connection.

One possible source of confusion might be if you also use a USB to RS232 converter (such as that supplied with other

dataTaker logger models) – it may also be identified as an "FTDI" device. By removing the adapter or the *DT80* connection and observing the COM port lists in DeLogger and DeTransfer it should be possible to work out which COM port has been assigned to which device.

Note The assigned COM port is associated with the particular PC USB port that you are using. If you subsequently connect the *DT80* to a different USB port then it will be assigned a different COM port number.

Using the USB Connection

Once the driver has been successfully installed, the USB connection will operate in a very similar way to an RS232 connection, except that

- it will be faster
- you do not need to set baud rate or flow control options

However, note that:

- it is recommended that the logger not be allowed to go to sleep while the USB cable is connected. By default, low power sleep mode is automatically disabled if a USB cable is connected.
- modems cannot be used on the USB interface

RS-232 Communications

Direct RS-232 Connection

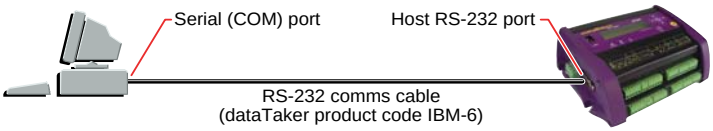


Figure 29: Direct (local) connection

For applications where the *DT80* is to be directly connected to a nearby computer, **USB** is normally the preferred communications medium. However, you may wish to use a direct **RS232** connection if:

- your computer has no available USB ports
- you need the *DT80* to go into low power sleep mode between scans, and you want to continue to receive real-time data returns; or you want to be able to wake the *DT80* by sending a character (See *Sleep Mode* (P118))
- the required cable length is longer than about 5 metres.

To set up a direct RS232 connection you will need a "cross-over", or "null-modem" cable. A suitable 9-pin cable (*dataTaker* product code IBM-6) may be ordered for this purpose. See *Cable Details* (P195) for wiring information.

It is also possible to use a simpler 3-wire cable (**RXD**, **TXD** and **GND**) although this will mean that hardware flow control is not possible.

If your computer has no RS232 ports (as is the case for some recent laptop models) a USB to serial adapter may be used.

Cable Length

Although the RS-232 standard specifies a cable of not more than 4 metres (15 feet), longer cables can be used. It's possible to use RS-232 cable runs of 100 metres or more, but to achieve reasonably error-free communication these generally need to have heavier wires and a slower baud rate may be necessary.

DT80 RS-232 Port

The *DT80* has a 9-pin male connector for RS-232 serial communication to a computer or modem. The pinout is the same as that used on a PC, namely:

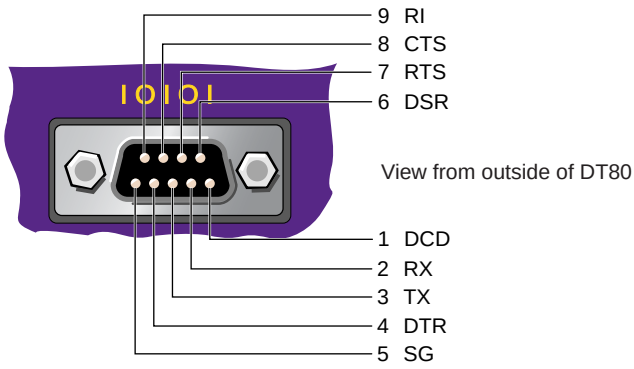


Figure 30: DT80 Host RS-232 port — connector pinout (DTE)

See *RS-232 Standard* (P195) for more details on individual pin functions.

Configuring the Host RS-232 Port

There are three parameters that need to be set for any RS232 port, and the *DT80*'s port is no exception:

- baud rate (data transfer rate in bits per second) *DT80* default is **57600**.
- serial framing format (number of data bits, parity type, number of stop bits) *DT80* default is "**N,8,1**" – no parity, 8 data bits, 1 stop bit.
- flow control (mechanism for one computer to tell the other to stop sending) *DT80* default is **software** flow control (special characters are used to signal "stop" and "go").

It is essential that both ends of an RS232 link be configured identically – same baud rate, framing and flow control.

You can check the *DT80*'s current RS232 parameters using the **PH** command, eg:

```
PH
RS232,57600,N,8,1,SWFC
```

Temporary Settings

The Host RS232 communications parameters can be set by the command

PH=*baud*,*parity*,*databits*,*stopbits*,*flow-control*

where:

Parameter	Settings	Default
baud	is the baud rate at which you want the RS232 port to operate. Use **300**, **600**, **1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600** or **115200**.	**57600**
parity	can be **N** (none), **O** (odd) or **E** (even)	**N**
databits	can be **7** or **8**	**8**
stopbits	can be **1** or **2**	**1**
flow-control	**NOFC** (no flow control) **SWFC** (software flow control) **HWFC** (hardware flow control) **SWHW** (both software and hardware flow control)	**SWFC**

These parameters may be specified in any order and all are optional.

For example, the command

```
PH=115200,HWFC
```

sets the RS232 port to 115200 baud, no parity, 8 data bits, 1 stop bit, and hardware flow control.

These settings will be reset to their defaults by a hard reset (eg. **SINGLEPUSH**).

PROFILE Settings

To make the host port settings "permanent", you can enter them into the **startup profile**, in which case they will be applied after every hard reset (although not in the case of a "triple push" reset, see *Safe Mode* (P136))

For example:

```
PROFILE "HOST_PORT" "BPS"="115200"
PROFILE "HOST_PORT" "FLOW"="HARDWARE"
SINGLEPUSH
```

See *Startup Profile* (P133) for details on the available profile settings. Note that the **SINGLEPUSH** command is used to force a *DT80* reset, which will then cause the profile settings to be loaded.

Flow Control

Flow control is the means by which communicating devices (such as the *DT80* and a host computer) control each other's transmission of characters to avoid data loss. The receiver uses flow control to disable transmissions by the sender if the receiver's input buffer is at risk of overflowing and thereby losing data.

The *DT80* supports all methods of flow control:

- Software flow control (**SWFC**; also known as XON/XOFF flow control, XON/XOFF handshaking, or software handshaking)
- Hardware flow control (**HWFC**; also known as RTS/CTS flow control, RTS/CTS handshaking, or hardware handshaking)
- No flow control (**NOFC**)
- **SWHW** (both software and hardware flow control)

The *DT80* will often be used set to SWFC, which is the default. But there may be times when there is a need to change the *DT80* to HWFC — for example, when using software on the host computer that doesn't support SWFC, or when using devices in the communications link (such as modems, radios or line drivers) that don't support SWFC.

Figure 31 (P117) summarizes the recommended flow control settings for the two general types of RS-232 connections between a *DT80* and its host computer.

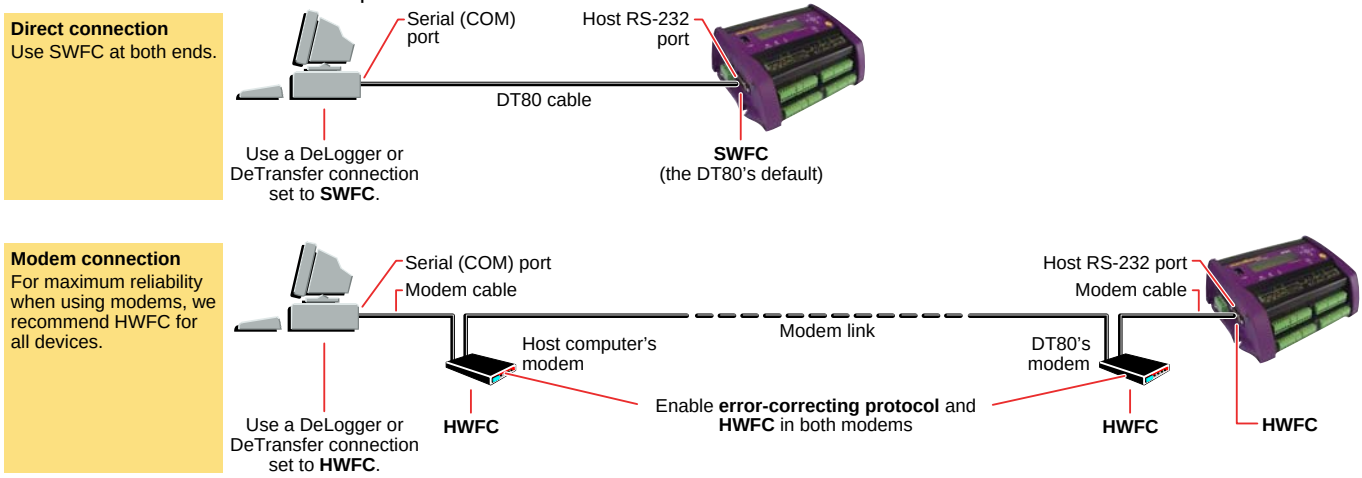


Figure 31: Recommended flow control settings for the two types of connections

Software Flow Control (SWFC)

In SWFC mode, the receiver controls the flow of characters by transmitting

- the XOFF character (ASCII 19 or Control S) to stop the sender from sending further characters
- the XON character (ASCII 17 or Control Q) to allow the sender to resume sending characters.

If the *DT80* receives an XOFF character, it will stop transmission within two character periods. If no XON is received within 30 seconds (see *P26* (P130)) the *DT80* will resume transmitting anyway.

Hardware Flow Control (HWFC)

In HWFC mode, the transmission of characters is managed by the **RTS** (Request To Send) and **CTS** (Clear To Send) lines of the RS-232 serial port of the sender and receiver. The state of these lines determines if transmission by the sender can proceed. The receiver raises the **RTS** line when it is able to receive characters from the sender, and lowers the **RTS** line when not able to receive characters. The **RTS** line of the receiver is connected (by means of the communications cable) to the CTS line of the sender, and the sender only transmits characters when its **CTS** line is high.

The *DT80* communications cable (product code IBM-6) has the RTS/CTS lines connected in this crossover manner — see *Figure 69* (P195).

HWFC is inherently more reliable than SWFC, because the flow control state (send/don't send) is continuously indicated by the hardware signals. SWFC can get into difficulties if line noise causes an XON or XOFF character to be lost, for example.

HWFC is therefore the preferred method. It is not, however, the default because it can only be used if the cable is wired appropriately and the host computer is configured to use HWFC. (In this sense SWFC is a little more "forgiving").

If the RS232 cable is accidentally disconnected, the *DT80* will no longer see **CTS** active, so it will stop transmitting. If, however, this condition persists for more than 30 seconds (see *P26* (P130)), the *DT80* will conclude that the host computer is no longer connected and will stop trying to transmit, until **CTS** again becomes active.

No Flow Control (NOFC)

The *DT80* can also be set to NFC (No Flow Control), in which case there is no control of the sender by the receiver. Use this setting with care, and only where there is no risk of the receiver being over-run by excess data from the sender, otherwise data loss will occur.

SWHW (Both)

The *DT80*'s SWHW setting is provided for backwards compatibility. It enables both software flow control and hardware flow control at the same time.

Sleep Mode

If the *DT80* is in low power sleep mode, it can be woken by sending a character to the RS232 port. Note, however, that the character that was sent will be discarded, if other characters are sent immediately afterwards they may be discarded, too.

It is recommended that the *DT80* be woken by sending a CR character, then waiting at least 500ms before sending any commands. (Note that the "Wakeup Required" option in DeTransfer can be used to automatically prefix all commands by the abovementioned wakeup sequence.)

Note that only the RS232 port can be used to wake the *DT80* in this way, because when the *DT80* goes to sleep any Ethernet or USB connections are terminated. This also means that the RS232 port can be used to monitor real time data returns where the *DT80* is configured to go to sleep between scheduled scans – something which is not possible with the USB or Ethernet port.

Modem Communications

Modem (Remote) RS-232 Connection

Another common way of communicating with the *DT80* is to connect its Host RS-232 port to a wired or wireless modem, which communicates with another modem connected to the host computer at the other end of the comms link. This way, the *DT80* can be across town or across the world from the host computer, and the link can use PSTN (landline), radio, GSM (cellular) or satellite communication. This is known as a **modem connection** to a **remote** *DT80*.

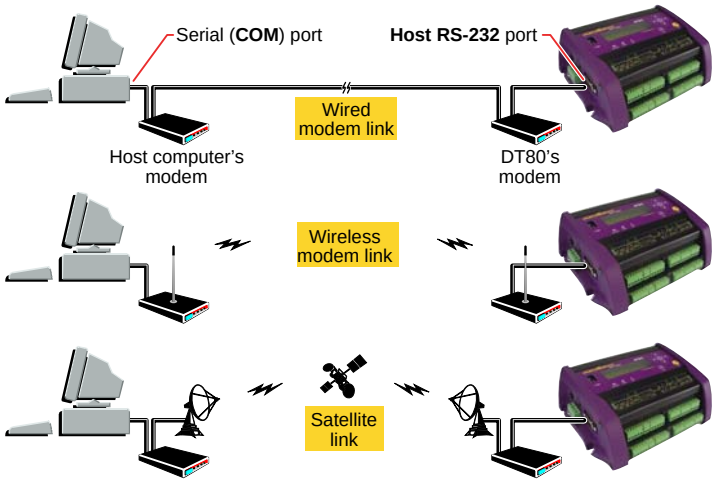


Figure 32: Modem (remote) connections

The *DT80* supervises the modem using standard Hayes-compatible "AT" commands. Certain command strings are configurable, to allow the widest possible range of modems to be supported.

The *DT80* can control the modem's power supply. If this facility is used, the *DT80* can automatically reset the modem if it determines that this is necessary. See *Powering the DT80's Modem* (P120).

Automatic Modem Detection

A *DT80* uses the state of its Host RS-232 port's **DSR** terminal (**DSR** = "Data Set [ie. modem] Ready") to determine the type of device connected to the port as follows:

If the *DT80*'s DSR terminal is not held active by the connected device, the *DT80* assumes that it's connected directly to the host computer and operates accordingly.

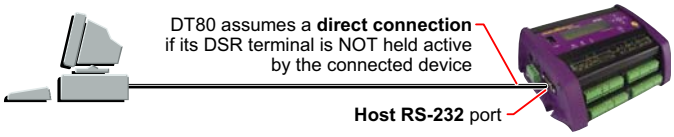


Figure 33: DSR inactive (low)

If the *DT80*'s **DSR** terminal is held active by the connected device, the *DT80* assumes that it's connected to a modem and operates accordingly, initialising the modem, monitoring other Host RS-232 lines to determine when a modem connection to

the host computer has been established, and so on.

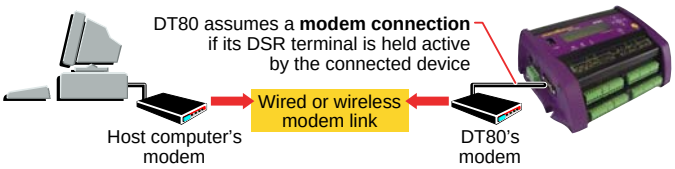


Figure 34: DSR active (high)

DT80-to-Modem Cable

As discussed above, for the *DT80* to recognise that it's connected to a modem and operate accordingly, the *DT80* must see the signal at the **DSR** terminal (pin 6 of the *DT80*'s host RS232 port) as active. There are two ways to ensure this:

- Connect the *DT80*'s Host RS-232 port to a modem using a straight-through (full-parallel) comms cable as shown in Figure 70*(P196).* This will connect the *DT80*'s **DSR** input to the modem's **DSR** output. Nearly all modems will drive the **DSR** signal active while they are switched on.
- If a modem is being used that does not drive its DSR line active when turned on, you can hardwire **DSR** to **DTR** at the *DT80* end of the modem cable. This simulates an active **DSR** terminal, convincing the *DT80* that it's connected to a modem.

Modem Initialisation

Modem Initialisation Conditions

The *DT80* automatically attempts to initialise the device attached to its Host RS-232 port when it first detects the **DSR** signal as active.

It may also re-initialise the device if it detects that the modem has been off-line (ie. not connected to the remote modem) for a long period of time (12 hours, by default – set this using the **MAX_CD_IDLE** profile key). This is done just in case the modem is in an error state or "locked up". (Even if the modem is not in this state, the initialization does no harm.)

Modem Initialisation String

The *DT80* initialises the modem by sending the initialisation string specified by the **INIT** profile key (**HOST_MODEM** section, see *Startup Profile* *(P133)*)

The default setting for this string is `ATE0Q1&D2S0=4&C1&S0`. This will configure the modem as follows:

- **ATE0** – Don't echo commands.
- **ATQ1** – Don't report results of commands (quiet mode).
- **AT&D2** – Disconnect (hang up) if **DTR** signal goes inactive.
- **ATS0=4** – Auto-answer incoming calls after 4 rings.

  The *DT80* does not issue any commands to the modem to answer a call. Therefore, if dial-in functionality is required, the modem must be set to auto-answer incoming calls. This setting may be removed if you do not need dial-in functionality, in which case the modem will ignore incoming calls.
- **AT&C1** – Set **CD** signal active only while connected to remote modem.
- **AT&S0** – Set **DSR** signal active at all times.

Additional Settings

It is recommended that an error-correcting protocol (eg. V42 or MNP2/3/4) is used between the modems, along with local RTS/CTS flow control. In most cases these will be enabled by default if the modem supports them.

Note that to be effective, these settings need to be set on both the *DT80*'s and the host computer's modems.

For the *DT80*'s modem, check the modem documentation for the proper commands. Typically, the `AT\N3` command is used to enable error correction (V42/MNP), and `AT\Q3` is used to enable RTS/CTS flow control. The required commands can be added to the initialisation string if they are not enabled by default.

For a Windows based host computer, check the modem's Advanced Settings panel, which is accessed from the **Phone and Modem Options**.

Modem Automatic Baud Rate Selection

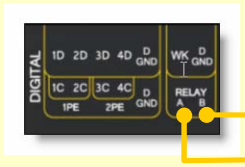
Modems compatible with the AT command set automatically set their local baud rate to match that of the host when the first AT command is received. Therefore, when the *DT80* sends the initialization string to the modem, the modem will automatically adjust its own baud rate to match that of the *DT80*.

In other words, the modem's local baud rate does not have to be set manually.

Powering the *DT80*'s Modem

If required, the *DT80* can control power to the modem, so that it can be powered down when not in use. For example:

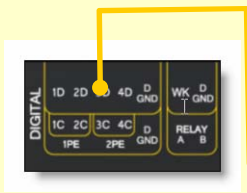
DT80 controls the modem's power supply by means of the RELAY connection to



+Power
-Supply
-Modem
+Power

DT80 controls the modem's power supply by means of the RELAY connected the *DT80* The modem would be controlled by **1RELAY=1** to turn power on to the modem whilst **1RELAY=0** would turn power off.

DT80 controls the modem's power supply by means of a relay connected to the digital channels 1~4



+Power
-Supply
-Modem
+Power

DT80 controls the modem's power supply by means of a relay connected to one of the *DT80*'s digital output channels D1 to D4. In this case the modem would be controlled by **3DSO=0** would turn power on to the modem whilst **3DO=1** would turn power off.

Figure 35: The DT80 can control the modem's supply

Automatic Modem Power-Down Reset

The *DT80* provides an additional feature where the modem can be automatically reset (by removing and re-applying power) if it appears to be unresponsive – that is, it has been off-line (ie. not connected to the remote modem) for a long period of time (12 hours, by default – set this using the **MAX_CD_IDLE** profile key).

To enable this feature, send one of the following **PROFILE** commands:

- If the modem is powered from one of the *DT80*'s digital output channels *n* (where *n* = **1** to **4**), send the command
  PROFILE "HOST_MODEM" "EXT_POWER_SWITCH"="*n***"**
- If the modem is powered via the *DT80*'s relay channel, send the command
  PROFILE "HOST_MODEM" "EXT_POWER_SWITCH"="-1"
- If the modem is not powered by either of the above, send the command
  PROFILE "HOST_MODEM" "EXT_POWER_SWITCH"="0"
  to disable the feature.

Then carry out a hard reset of the *DT80* (**SINGLEPUSH**) to load the startup profile.

From then on, the *DT80* automatically power-down-resets the modem if it detects it to be unresponsive — see also Modem Initialisation *(P119)*.

Modem Communications Operation

Dialling In

As the modem is initialised to automatically answer incoming calls, the *DT80* does not have to monitor the **RI** signal at its Host RS-232 port or request the modem to answer the call. But the *DT80* does have to monitor the **CD** signal to determine when a call has been established.

The *DT80* does not communicate via the RS232 host port unless it determines that a call has been established between itself and a host. When a modem is attached (**DSR** active), the *DT80* monitors the **CD** signal to determine when it can transmit data and status information, and receive commands:

- When **CD** is active the *DT80* accepts commands, and returns data and status information – exactly as it would for a direct connection.
- When **CD** is inactive the *DT80* ignores any received characters and does not transmit data or status information.

This behaviour ensures that any rubbish characters received outside of a call are ignored, and that the *DT80* does not send characters to the modem that the modem may interpret as commands to switch into a different operating state.

Dialling Out

The *DT80* can also initiate an outgoing modem call, which would typically be done in response to an alarm.

SETDIALOUTNUMBER Command

Send the command
SETDIALOUTNUMBER"*digits***"**
to the *DT80* to specify the telephone number to be dialled by the **DIAL** command to establish a connection to the host computer.

DIAL Command

The **DIAL** command causes the *DT80* to instruct its modem to dial out to the telephone number specified by

SETDIALOUTNUMBER. If a call cannot be placed for any reason, the command is ignored. This is often used as an alarm action command to cause the *DT80* to dial out when an alarm condition arises (see Alarm Action Processes (P76)).

HANGUP Command

The **HANGUP** command causes the *DT80* to instruct its modem to hang up (disconnect) the current dial-out or dial-in connection. If there is currently no connection, **HANGUP** is ignored. This can be used in an alarm action command to cause the *DT80* to hang up a call in progress when an alarm condition arises (see Alarm Action Processes (P76)).

Example — Modem Control Commands

The use of the *DT80*'s modem control commands is demonstrated in the following program:

```
BEGIN"FLUFFY"
SETDIALOUTNUMBER"12345678"
RA10M
 'Read boiler temp
 1TK(=1CV,W)
 IF(1CV>120){DIAL}
END
```

Every 10 minutes, the program checks the boiler temperature and then, if it exceeds 120°C, instructs the modem to initiate a dial-out to phone number **12345678**.

Modem Status

The system variable 25SV gives an indication of the current state of the modem. It can be used with alarms to determine the current state of the modem connection and to behave accordingly.

See **25SV** (P35) in the Table 2: *DT80* System Variables (P35).

Setting Up a Remote Connection

The following is a brief summary of the steps involved in setting up a remote modem connection between the *DT80* and a host computer.

1. Review the *DT80*'s default modem initialisation string and verify that it is suitable for your modem. Determine what, if any, commands need to be added.
2. Connect to the *DT80* using a USB or direct RS232 connection.
3. Set the required profile settings to configure the host port and modem. For example:
   ```
   PROFILE "HOST_PORT" "FLOW_CONTROL"="HARDWARE"
   PROFILE "HOST_MODEM" "EXT_POWER_SWITCH"="3"
   SINGLEPUSH
   ```
   will set HWFC mode (recommended), and configure modem power control using digital output **3D**.
4. Connect power to the modem (in the above example the power would be supplied via a relay driven by **3D**).
5. Connect a suitable comms cable between the serial port on the *DT80*'s modem and the *DT80*'s Host RS-232 port. This cable is normally supplied with the modem, see also *DT80-to-Modem* Cable (P119).
6. At the remote end of the link, connect a suitable comms cable between the serial port on the host computer and the local modem.
7. On the host computer, configure the modem using the Windows control panel. It is recommended that hardware flow control and an error-correcting protocol (eg V42) are used.
8. On the host computer, launch suitable terminal software, eg. DeLogger or DeTransfer. Set up a connection to use the modem.
9. Attempt to connect to the *DT80* from the host computer.

Visits to Site

If the site is visited where the *DT80* and the modem are installed, the *DT80* can be communicated with directly from a PC/Notebook by unplugging the cable from the modem to the *DT80* at the *DT80* end and then plugging in a direct RS232 cable from your PC/Notebook to the *DT80*. Differences in the cable wiring allow the *DT80* to determine the type of connection and to respond appropriately.

Alternatively, a USB cable can be used, in which case the modem cable need not be disturbed.

Ethernet Communications

The *DT80*'s Ethernet port allows you to connect the *DT80* to a TCP/IP (P210) based local or wide area network. The *DT80* can then provide the following **services** to client computers:

- access to the *DT80*'s standard **command interface**
- access to the *DT80*'s built-in **web interface** (P98)
- transfer of files (eg. job files, store files) to or from the *DT80*'s file system, using the **FTP** protocol (P126)

These client computers may be on the desk next to the *DT80* or, via the Internet, on the other side of the world:

TCP/IP Concepts

To use a *DT80* on a TCP/IP network, you must configure the *DT80* with three numbers:

- its own unique **IP address**
- the **subnet mask** applicable to the network to which it is connected
- the **gateway IP address** applicable to the network to which it is connected

IP Address

Every device that is connected to a TCP/IP network must have its own unique identifier, called its **IP address**, and the *DT80* is no exception. No two devices in the same network can have the same IP address.

An IP address is single 32-bit integer, but it is normally written as four numbers (each in the range 0-255), separated by periods, eg. **192.168.2.101**.

Assigning an IP Address

Some devices are able to automatically request an IP address when they connect to a network; such a device will then have a **dynamic** IP address.

Other devices, such as the *DT80*, need to be configured with a **static** IP address. Each time the device connects to the network it will use the same IP address. In the case of the *DT80* its IP address is configured as part of its startup profile.

Subnet Mask

The *DT80*'s IP address actually consists of two parts:

- The **network number**, which identifies the network to which the *DT80* and its neighbours are connected.
- The **node number**, which uniquely identifies this *DT80*. No two devices on the network may have the same node number. Also known as a **host number**.

The **subnet mask** is a property of the network to which the *DT80* is connected and specifies which part of the IP address is the network number and which is the node number.

For example, the subnet mask **255.255.255.0** specifies that the first three parts of the IP address are the network number, and the last is the node number within the network. So in this case the IP address **192.168.2.101** would represent node **101** on the "**192.168.2**" network (sometimes written as "**192.168.2.0/24**", which indicates that 24 of the 32 bits in the IP address are used to specify the network number, leaving 8 bits for the node number).

Gateway

The *DT80* can communicate directly with any of the nodes connected to its local network. In many cases this is all the connectivity that is required.

If, however, the *DT80* needs to be accessed from farther afield then it needs to know how to communicate with computers on different networks. This is done by assigning one of the computers on the *DT80*'s local network to be a **gateway**. The IP address of the gateway is then entered into the *DT80*.

Now, any time the *DT80* wants to talk to a computer on a "foreign" network it will simply passes the data to the designated gateway and let it sort it out. The gateway may then pass the data on to other gateways, until eventually the data finds its way to its destination.

As hinted above, if you only want to connect your *DT80* to one computer, or to a few computers which are all on the same local network, then it is not necessary to specify a gateway.

Connecting to the *DT80* Ethernet Port

Important Do not connect your *DT80* to the network until you've configured the *DT80* with a suitable IP address and subnet mask. Connecting a device with an invalid or conflicting IP address may cause significant disruption to the operation of the network.

Connection Topology

The *DT80*'s Ethernet port is designed to connect to any **10-BaseT Ethernet** compatible network. The port operates at a maximum data rate of 10Mbps.

There are two ways to connect to a network:

- directly connect the *DT80* to a single host computer using a "cross-over" cable. In this case you are effectively creating a new mini-network, with just two devices connected – the *DT80* and the host computer.
- connect the *DT80* to a spare port on an Ethernet hub, bridge or router, using a standard ("straight through) cable. In this case the *DT80* will be joining an existing network.

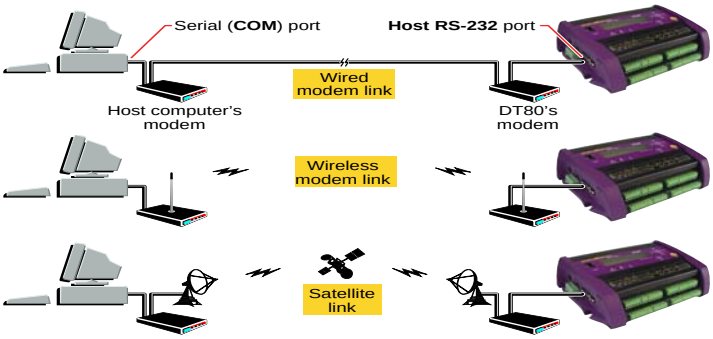


Figure 36: Ethernet connection types — direct connection and network connection

Ethernet Port Indicators

The two LEDs on the *DT80*'s Ethernet port *(P140)* indicate the following:

- Green LED – **Link OK**; should come on and stay on as soon as you connect the Ethernet cable
- Amber LED – **Activity**; blinks every time a data packet is received

If the green LED does not come on then either the cable is faulty, or the socket you are connecting to is not connected to an active Ethernet network, or the hub/bridge/router is not powered up.

Note that the amber LED indicates communications activity anywhere on the local network – this activity is not necessarily directed at the *DT80*.

MAC Address

All Ethernet devices have a globally unique 12-digit identifier programmed into them during manufacture. This is called the **MAC Address**. The *DT80*'s assigned MAC address can be viewed using the **EAA** (Ethernet Adapter Address) command, but it cannot be changed. You should not need to be concerned with this address.

Ethernet Commands

Querying Ethernet Parameters

The current Ethernet parameters can be viewed using the following commands

Command	Description	
IP	Returns the *DT80*'s current IP address	
IPSN	Returns the *DT80*'s current IP subnet mask	
IPGW	Returns the *DT80*'s current IP gateway	
EAA	Returns the *DT80*'s Ethernet network adapter address	Set at factory (read-only)

For example:

```
IP
192.168.42.15
IPSN
255.255.255.224
IPGW
192.168.42.3
EAA
00-90-2D-00-12-6B
```

In this example the *DT80* has IP address **192.168.42.15**, and is connected to network **192.168.42.0/27** (Notice that in this the subnet mask specifies that 27 bits are used to identify the network, leaving only 5 bits to identify the node. Consequently no more than 32 (2^5) nodes could be connected to this network.)

The computer at IP address **192.168.42.3** is the designated gateway for this network. Any data that needs to be sent to a different network will be sent via this computer.

Setting Ethernet Parameters

All three Ethernet parameters (IP address, subnet mask and gateway) are set using profile settings (see *Startup Profile* (P133)), so that they are loaded every time the *DT80* restarts.

For example, the following commands would be used to set the parameters in the above example:

```
PROFILE "ETHERNET" "IP_ADDRESS"="192.168.42.15"
PROFILE "ETHERNET" "SUBNET_MASK"="255.255.255.224"
PROFILE "ETHERNET" "GATEWAY"="192.168.42.3"
SINGLEPUSH
```

Note that the **SINGLEPUSH** command will cause a hard reset, which will cause the new settings to be loaded.

Selecting Ethernet Parameters

The general procedure for determining what the *DT80*'s Ethernet parameters should be set to is to first setup the host PC (or examine its existing settings), then configure the *DT80* to match.

The first decision is whether to directly connect the *DT80* to a single computer, or to connect it to existing network infrastructure.

Single Computer Connection

A single computer connection is appropriate if you only want to access the *DT80* from the one computer, and the computer has an unused Ethernet interface which can be dedicated to the *DT80*. If this is the case, read on.

The exact procedure for setting up and configuring a PC network interface varies with the operating system being used. In general terms, the procedure is as follows:

1. Open the **Network Connections** section of the Windows control panel.
2. Look for an icon representing your Ethernet/LAN interface. If you do not see an icon for the LAN interface you want to use, you will need to **Add** a new connection, which will start the Network Connection wizard. Tell the wizard that you want to set up an Internet connection, but select manual configuration when asked. When the wizard completes you should now have a network connection icon.
3. Open the network connection icon's Properties page and ensure that the **Internet Protocol (TCP/IP)** entry is enabled. Select the TCP/IP entry and open its Properties page.
4. On the TCP/IP Properties dialog, you should now be able to enter an IP address for the computer's Ethernet interface, and a subnet mask for the mini-network you are creating. The only requirement here is that if the computer has a second Ethernet interface (for connecting to an office LAN, for example), then the IP addresses for the two interfaces must have different network numbers.

   For example, a suitable choice might be set the IP address to **192.168.2.1** and the subnet mash to **255.255.255.0** (The IP address range **192.168.***x.y* is reserved for "private" networks such as this, so it should be used unless there is a compelling reason to do otherwise.)
5. You now need to select an IP address for the *DT80*. This must have the same network number as the PC (ie. **192.168.2** in this example) but a different node number (ie. anything other than **1**). So a *DT80* IP address of **192.168.2.2** would be suitable.
6. You now have all the information you need and can proceed to configuring the *DT80*. Connect to the *DT80* using a USB or RS232 connection and enter the appropriate profile settings. In this case:

   ```
   PROFILE "ETHERNET" "IP_ADDRESS"="192.168.2.2"
   PROFILE "ETHERNET" "SUBNET_MASK"="255.255.255.0"
   SINGLEPUSH
   ```

   Note that it is not necessary to set a gateway address.
7. Connect a cross-over cable between the *DT80* and the computer and verify that the green **Link OK** LED lights on the *DT80* and on the computer (if it has one).

You now test the connection, see xx

Joining an Existing Network

When connecting the *DT80* to an existing Ethernet network, you need to be a little more careful. Setting the *DT80* to an inappropriate IP address can severely disrupt the operation of the network.

By far the preferred approach here is to ask your network administrator what you should set the *DT80*'s IP address, subnet mask and gateway to (and which outlet you should connect the Ethernet cable to). You can then simply enter the required profile settings (using a USB or serial connection), connect the *DT80*'s Ethernet cable, and you are ready to test the connection.

If you don't have a "network administrator", then the following general procedure can be used, using a computer that is connected to the network to which you want to connect the *DT80*:

1. Open a command prompt window and type **ipconfig** (or **winipcfg** for Windows 98). This will display a summary of the computer's current network parameters, similar to:

   ```
   Ethernet adapter Local Area Connection:
   ```

```
   Connection-specific DNS Suffix  . : ad.datataker.com.au
   IP Address. . . . . . . . . . . . : 192.168.1.52
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.1.6
```

although the numbers you see may be completely different.

2. Take note of the subnet mask and default gateway settings. Later, these exact settings will need to be entered into the *DT80*.
3. Using the displayed IP address and subnet mask, work out what the network number is. If the subnet mask is **255.255.255.0**, then the network number is simply the first three parts of the IP address, ie. **192.168.1** in this case. Similarly, if the subnet mask is **255.255.0.0** then the network number is the first two parts of the address. If the subnet mask is something other than these then it becomes a little technical and you should ask an IP expert.
4. You now need to select an IP address for the *DT80*. This must have the same network number as the PC (ie. **192.168.1** in this example) but a unique node number which does not conflict with any other device on the network.

   This is where it can get tricky. One method of finding a free address is to "ping" various IP addresses to see whether there is any reply. Be aware that this is not foolproof, but it may be adequate for a small network where you are aware of all the devices that use it. The procedure is as follows:

   a) Switch on all computers and devices that are connected to the network in question and allow them to boot up.

   b) From the command prompt window, use the **ping** utility to test a candidate IP address, eg:

   ```
   ping 192.168.1.10
   ```

   c) If you see a `Reply from 192.168.1.10`... response then that address is not free and cannot be used for the *DT80*.

   d) If you see a `Request timed out`... response then the address can probably be used for the *DT80*.
5. You now have all the information you need and can proceed to configuring the *DT80*. Connect to the *DT80* using a USB or RS232 connection and enter the appropriate profile settings. For this example (assuming that IP address **192.168.1.12** passed the ping test) you would enter:

   ```
   PROFILE "ETHERNET" "IP_ADDRESS"="192.168.1.12"
   PROFILE "ETHERNET" "SUBNET_MASK"="255.255.255.0"
   PROFILE "ETHERNET" "GATEWAY"="192.168.1.6"
   SINGLEPUSH
   ```

6. Connect a standard (not cross-over) cable between the *DT80* and the Ethernet hub, bridge, router or wall socket and verify that the green **Link OK** LED lights on the *DT80* and on the device you connect to (if it has one).

Using the *DT80* Command Interface

Connecting

To access the *DT80*'s command interface over Ethernet you need to use a terminal program on the host computer that can send/receive text to **TCP port 8**. DeTransfer and DeLogger support this.

Using DeTransfer, for example, you first need to set up a **connection**. This is the same as setting up a connection for RS232 or USB, except that instead of specifying a COM port number, you now need to specify an IP address. The following screen shot illustrates this.

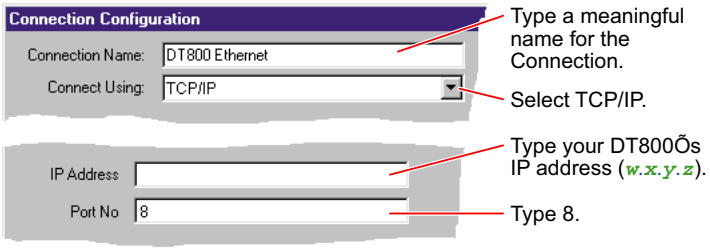


Figure 37: Example software connection for a DT80 on an Ethernet network (DeTransfer software shown)

If you now press Connect, you should be able to send commands to the *DT80* and receive data, just as you would using USB or RS232.

Multiple Connections

Up to ten client computers (or ten DeTransfer sessions on the one computer) can simultaneously connect to the *DT80* using the TCP/IP network. This is in addition to a possible USB and/or RS232 connection.

At any one time, only one of these interfaces/sessions can be the **active** interface. The active interface is the one on which the most recent *DT80* command was sent.

Whenever the *DT80* transmits something over the command interface (eg. returned/unloaded data, prompt strings, messages, etc.), it is send to the current active interface, plus all connected TCP/IP sessions. This provides a way to "broadcast" data to a number of different computers, each of which operates as a passive listener.

Disconnecting

It is important to note that all Ethernet sessions will be disconnected if the *DT80*:

- undergoes a hard reset (**SINGLEPUSH**, manual reset or power failure)
- enters low power sleep mode.

For this reason, the *DT80* normally disables sleep mode while an Ethernet cable is connected (although this can be overridden by setting P15 to 3 or 4.

If an Ethernet session is disconnected in this way, you may need to manually disconnect then reconnect in DeTransfer.

Internet Access

The *DT80* can be accessed remotely over the Internet in the same way that it can from a computer on the local network. This may, however, require some work by the network administrator.

If the *DT80*'s network is directly connected to the Internet via a router, then the *DT80* will already have an externally visible IP address, eg. **203.72.44.166**, in which case nothing needs to be done – anyone in the world can start up DeTransfer, point it at the *DT80*'s IP address, and connect to the logger.

A much more common scenario is where the *DT80* is connected to a private LAN, which is "hidden" from the Internet by a Network Address Translation (NAT) router/firewall. In this case the *DT80*'s IP address (eg **192.168.1.200**) is not visible to the Internet. The network administrator will need to obtain a public IP address for use by the *DT80*, and configure the router to allow TCP port 8 connections to this address to be forwarded to the *DT80*'s private IP address.

Using the *DT80* FTP Server

The *DT80* can also function as an FTP (File Transfer Protocol) server. You can use this mechanism to transfer data and program files to and from the *DT80*. This is done by running an **FTP client** application on the host computer and using it to connect to the *DT80*'s **FTP server** (by specifying the *DT80*'s IP address).

Passwords

The FTP server supports two types of access:

- an anonymous login (username **ANONYMOUS**, password can be anything) provides **read-only** access to the *DT80*'s file system.
- a full login (using the username and password configured in the startup profile) provides **read/write** access.

To set the FTP password, use the following profile commands:

```
PROFILE "FTP" "USER"="DT"
PROFILE "FTP" "PASSWORD"="TOPSECRET"
```

If, for security reasons, you want to disable the FTP server altogether, enter:

```
PROFILE "FTP" "SUPPORTED"="NO"
```

FTP Client Software

A Windows computer includes at least two different FTP clients that can be used to access the *DT80*'s file system. You can run the traditional command-line version by typing

ftp *ip-address*

in a command prompt window (*ip-address* is the IP address of the *DT80*).

Alternatively, most web browsers will allow you to browse the *DT80* by entering the following URL:

ftp://*user***:***password***@***ip-address***/***drive***:/** (for full read-write access), or

ftp://*ip-address***/***drive***:/** (for anonymous read-only access)

where:

- *user* and *password* are the username and password to use.
- *ip-address* is the *DT80*'s IP address
- *drive* is the *DT80* drive to browse (**A** or **B**).

For example, if you want to load a file onto the *DT80*, or delete a file, and the default username and password is set in the profile, then you would browse to:

```
ftp://DATATAKER:DATATAKER@192.168.1.202/B:/
```

The browser should then present a list of available files and folders, through which you can navigate simply by clicking on links. For example, you can navigate to a job's data directory (eg. **B:\JOBS\MYJOB\A**) then double click on a store file (eg. **DATA_A.DBD**). If DeView has been installed on the host PC then it will automatically download and open the file, then present the data in tabular or graphical form.

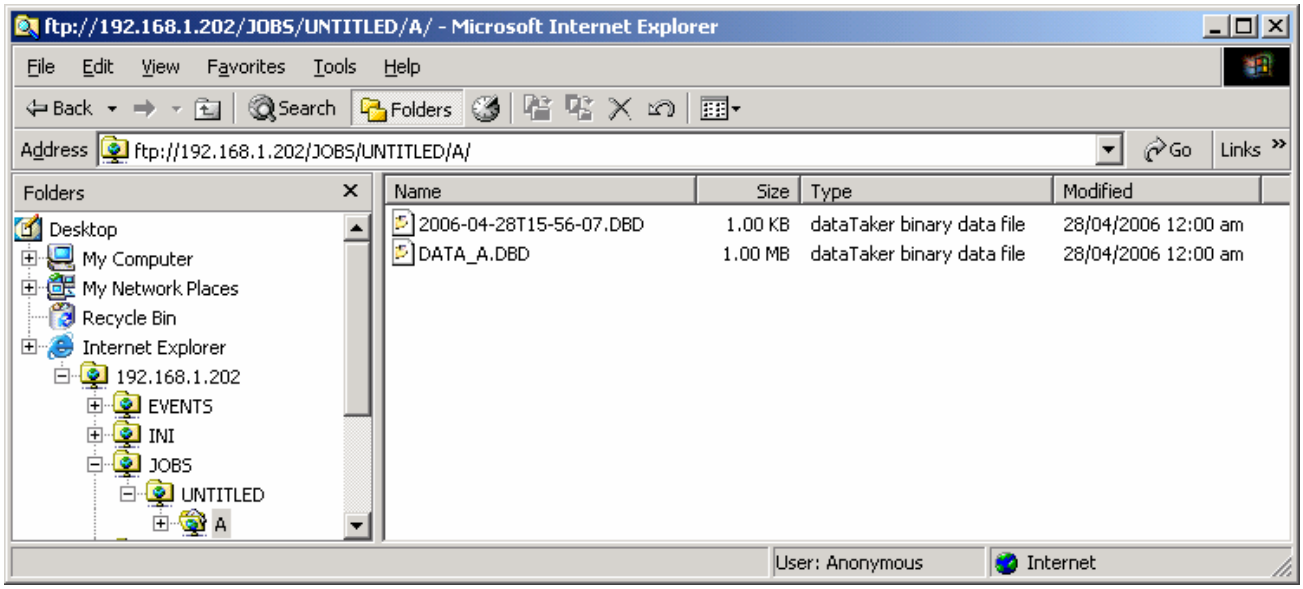


Figure 38: Typical file display when connected to DT80 *FTP server*

Troubleshooting

If you experience problems connecting to the *DT80* FTP server, it can be helpful to examine the raw FTP messages that are being exchanged. To enable display of received and transmitted FTP messages, set **P56=8**.

PPP Communications

Point-to-Point Protocol (PPP) allows TCP/IP-based protocols to be run over the Host RS-232 port of the *DT80*.

A client computer can connect to the *DT80*, via modem or direct cable, in much the same was connecting to a dial-up Internet Service Provider.

To initiate a PPP session with the *DT80*, clients must send the word **CLIENT** to the *DT80* by means of its Host RS-232 port. The *DT80* responds with **CLIENTSERVER** and, from then on, expects PPP packets. Note that Microsoft Windows PPP client software issues **CLIENT** and expects the **CLIENTSERVER** response by default.

To close a PPP session with the *DT80*, clients can simply close the PPP session from their end. Alternately, you can send the command **CLOSEDIRECTPPP** to the *DT80* and it will close the PPP session.

Setting up PPP

Setting up a PPP connection to the *DT80* involves configuring a "dial-up Internet connection" on a Windows based client PC. The *DT80* is essentially performing the function of the Internet Service Provider – when the client computer establishes a connection, the *DT80* will allocate an IP address to the PC and check the username and password supplied by it.

There are four *DT80* profile settings relating to PPP:

- **IP_ADDRESS** – this is the IP address that the *DT80* will use for its PPP interface. This must be different to that used for the Ethernet port. Default setting is `1.2.3.4`
- **REMOTE_IP_ADDRESS** – this is the IP address that the *DT80* will allocate to the client computer to use for its PPP interface. Default setting is `1.2.3.1`
- **USER** – the client computer must supply this username in order to be granted access. Default is `ANONYMOUS`
- **PASSWORD** – the client computer must also supply this password. Default is `PASSWORD`

When first setting up a connection, these settings can all be left on their default settings.

The general procedure for setting up a PPP link (which may vary somewhat depending on your operating system) is as follows:

1. Determine the current baud rate setting for the *DT80*'s host RS232 port (default is 57600, but you can check by connecting via USB and using the **PH** command)
2. Set the host RS232 port to HWFC, ie.
   PROFILE "HOST_PORT" "FLOW"="HARDWARE"
   and execute a **SINGLEPUSH**. Software flow control is not recommended for PPP.

3. Connect the host computer to the *DT80* using a standard RS232 cable (dataTaker part IBM-6)
4. Open the **Network Connections** section of the Windows control panel.
5. You will now need to **Add** a new connection, which will start the Network Connection wizard. Tell the wizard that you want to set up a dial-up Internet connection, but select manual configuration when asked.
6. Now tell the wizard that you connect to the internet via a modem (as opposed to a LAN)
7. If you don't have any modems installed on the computer, Windows will now want to search for one. Tell the wizard that you will select a modem manually. When the list of modem types is presented, choose "Communications cable between two computers" and specify the correct COM port.
8. When prompted for the "account details", enter the username and password as specified in the *DT80* profile (by default `ANONYMOUS` and `PASSWORD`, note these are case sensitive). If prompted for a telephone number you can enter anything, it will not be used.
9. When the wizard completes you should now have a Dial-up network connection icon.
10. Open the network connection icon's Properties page and ensure that the "Communications cable between two computers" modem is selected. Click **Configure** to open the modem's configuration dialog.
11. Set the "maximum speed" option equal to the *DT80*'s host port baud rate (57600 by default). Close the Properties dialogs.

    Note that we did not set the IP address for the connection. It is left on its default setting, which is "automatic". The *DT80* will tell the client computer what IP address to use.
12. Right click on the network icon and select **Connect**, then press **Dial**
13. After a short delay, Windows should report that you are now connected.
14. Disconnect by clicking on the control panel or task bar icon and selecting **Disconnect**.

Once this basic setup is working, you can modify the *DT80*'s username/password settings for improved security, remembering that you will also need to edit the client's network connection settings to match.

You can also edit the network connection properties to use a real dial-up modem rather than a null-modem cable (and of course install a modem at both the *DT80* and the client computer ends). This would then provide remote dial-up access to the *DT80*.

Using PPP

Once you have a PPP connection established, you can use it in exactly the same way as you would use an Ethernet connection. The only difference is that when you define the connection in DeTransfer or DeLogger, you will now enter the IP address of the *DT80*'s PPP interface (`1.2.3.4` by default), rather than the IP address of its Ethernet interface.

Part L — Configuration

Configuring the *DT80*

Parameters

DT80 **parameters** are internal system settings. They are global in their effect, and allow a variety of options to be set. As a general rule, set the parameters that require changing before programming schedules and alarms.

Parameters are numbered from **P0** to **P62**, although not all numbers are used. Each parameter is an integer; the range of allowable values varies from parameter to parameter.

Reading Parameters

To read the current setting of a parameter, simply send the parameter's ID. For example, to read the value of P11:

```
P11
50
```

Setting Parameters

Parameters can be set at any time, and new settings generally take effect immediately. For example, send:

```
P11=60
```

to set parameter P11's value to 60.

Note Parameters are not channels. The statement **P11=60** is a **command**, and is carried out immediately, even if it appears within a schedule definition. You can use the **DO** command to set parameters when a schedule executes – see *Executing Commands in Schedules* (P52) for more information.

The *DT80* recognises the following parameters:

Parameter	Specifies	Units	Default Value	Range of Values	Comment
P0	Max analog input drift before re-calibration	µV	4	0 to 10000	Voltage measurements may "drift" as the ambient temperature changes. If the drift is greater than this amount the *DT80* will automatically re-calibrate itself to restore accuracy.
P3	Minimum sleep period	seconds	4	1 to 30000	The *DT80* will only go to sleep if the sleep duration can be for at least this period of time
P4	Sleep-to-wake settling latency	seconds	3	1 to 30000	Time required by *DT80* to resume normal operation after leaving sleep mode
P5	Maximum sleep period	minutes	60	1 to 30000	Maximum time that *DT80* is allowed to sleep for.
P8	Command processor diagnostic mode	mode	0	0 to 1	If this parameter is set to 1 then each and every command string will be displayed before being executed. This can be useful for verifying that alarm actions are being carried out, as these commands are not normally echoed.
P9	Logging of alarm state	mode	1	0 to 3	0 = do not log alarms 1 = log false to true transitions only 2 = log true to false transitions only 3 = log both transitions
P11	Mains frequency	Hz	50	1 to 10000	Sets analog measurement duration to 1/P11 seconds . Set P11 to the local mains frequency for best noise rejection.
P14	Comms ports password protection timeout	seconds	600	1 to 30000	When a password is defined, the *DT80* automatically signs off after this period of inactivity (see *Password Protection* (P114)
P15	Low-power operation	Mode	0	0 to 4	0 = Allow sleep if battery powered and Ethernet/USB not connected 1 = Allow sleep if Ethernet/USB not connected 2 = Do not allow sleep 3 = Allow sleep 4 = Allow sleep if battery powered See *Controlling Sleep* (P146)
P17	Delay to low-power mode	seconds	30	1 to 255	Sets how long the *DT80* waits before entering low-power mode after the last communication or key press
P19	Status screen display	bitmask	255	0 to 255	bit 0 set = display sign-on screen bit 1 set = display date/time screen bit 2 set = display battery status screen

Parameter	Specifies	Units	Default Value	Range of Values	Comment
					Default is to display all status screens. See *Enable/Disable status screens* *(P94)*
P22	Data delimiter character	ASCII	32 (space)	1 to 255	In free format mode (**/h**), this character is inserted between the data value and the units string (if **/U**), or between the data value and the next data value (if **/u**)
P24	Scan delimiter character	ASCII	13 (CR)	1 to 255	In free format mode (**/h**), this character is inserted at the end of each schedule's data. Note that CR characters are always followed by LF.
P26	Flow control timeout	seconds	30	0 to 255	If the *DT80* has been prevented from sending on the host RS232 port for this amount of time (due to XOFF received or CTS not active) then it will assume that the host computer is no longer connected and will discard subsequent output text. Set to 0 to disable this timeout
P27	3HSC input mode	mode	0	0 to 3	Clock source for 3HSC counter 0 = **3C** terminal 1 = internal 32768Hz signal, count while **3C** terminal is high 2 = **3C** terminal, count while **4C** terminal is high 3 = internal 1024Hz signal
P31	Date format	mode	1	0 to 3	0 = days since 1-Jan-1989 1 = European (DD/MM/YYYY) 2 = North American (MM/DD/YYYY) 3 = ISO (YYYY/MM/DD)
P32	Number of significant digits	digits	8	1 to 9	Sets the number of significant digits shown in returned data or unloaded logged data
P33	Field width	characters	0 (variable)	0 to 80	If non-zero, all data values returned in free format mode (**/h**) will be padded with leading spaces so that the total field width is P33 characters. Caution - may truncate data values (eg. "12345" may display as "123") if P33 is too low.
P36	Temperature units	mode	0	0 to 3	Units for all temperature measurements: 0 = ºC Celsius 1 = ºF Fahrenheit 2 = K Kelvin 3 = ºR Rankin
P38	Decimal point character	ASCII	46 (.)	1 to 255	In free format mode (**/h**), this character is used as the decimal point character.
P39	Time format	mode	0	0 to 3	0 = HH:MM:SS.TTT 1 = decimal seconds since midnight 2 = decimal minutes since midnight 3 = decimal hours since midnight
P40	Time separator character	ASCII	58 (:)	1 to 255	This character is used to separate HH, MM and SS fields in time values.
P41	Time sub-second digits	digits	3	0 to 6	Sets number of decimal places in time values
P45	DDE/OLE tag control	mode	0	0 to 2	0 = do not add DDE/OLE tags 1 = output OLE tag ($!) before channel name 2 = output DDE tag (&!) before channel name Whitespace is replaced with underscores
P50	Time instant format	mode	0	0 to 5	Specifies the format to use when returning an absolute date/time value: 0 = P39P22P31 (time, delimiter, date) 1 = P31P22P39 (date, delimiter, time) 2 = decimal seconds since 1-Jan-1989 3 = decimal minutes since 1-Jan-1989 4 = decimal hours since 1-Jan-1989 5 = decimal days since 1-Jan-1989
P51	Time interval format	mode	0	0 to 5	Specifies the format to use when returning a relative time value: 0 = P39P22d.d (time, delimiter, days) 1 = d.dP22P39 (days, delimiter, time) 2 = decimal seconds 3 = decimal minutes 4 = decimal hours 5 = decimal days
P53	Default serial sensor timeout	seconds	10	0 to 30000	Max time that the *DT80* will wait for a serial sensor input action to complete. May be overridden by channel factor If P53=0 then characters satisfying the input action must already have been received at the time that the input action is processed.

Parameter	Specifies	Units	Default Value	Range of Values	Comment
P55	Enable schedule wakeup	bitmask	16383	0 to 16383	bit 0 = not used bit 1 set = wake from sleep if schedule X is due bit 2 set = wake from sleep if schedule A is due bit 3 set = wake from sleep if schedule B is due … bit 12 set = wake from sleep if schedule K is due bit 13 set = wake from sleep if schedule S is due By default *DT80* will wake if any schedule is due.
P56	Diagnostic output	mode	0	0 to 2	0 = no diagnostic output 1 = SERIAL channel diagnostic output 2 = SDI12 channel diagnostic output
P62	Retain multiplexer settings after measurement	mode	0	0 to 1	0 = all terminals are disconnected from the *DT80* measurement and excitation circuits at the end of each scan 1 = connections are left set according to the last measurement in the schedule. This can be useful for verifying the *DT80*'s excitation output, or for rapid measurements of a single channel.

Table 4: DT80 Parameters

Setting Default Parameter Values

All parameter settings are cleared back to their power-on default values when a soft or hard reset (**RESET** or **SINGLEPUSH**) is performed.

However you can change the default value for any parameter using the startup profile. For example:

```
PROFILE "PARAMETERS" "P11"="60"
```

will change the default value for P11 to 60.

Switches

DT80 **switches** provide a further set of boolean parameters. Each switch is identified by a letter, and can either be **on** (uppercase) or **off** (lowercase).

Reading Switches

To read the current settings of all switches, use the **STATUS9** command *(P138)*, eg:

```
STATUS9
/B/C/d/E/f/h/i/K/l/M/N/R/S/t/U/w/x/Z
```

Switches that are ON are displayed in uppercase.

Setting Switches

Switches can be set at any time, and new settings generally take effect immediately. For example, send:

```
/T /e/m
```

to set switch T **on**, and set switches E and M **off**.

Note Switches are not channels. The statement **/T** is a **command**, and is carried out immediately, even if it appears within a schedule definition. You can use the **DO** command to set switches when a schedule executes – see *Executing Commands in Schedules* *(P52)* for more information.

The *DT80* recognises the following switches:

Switch	Function	Default	Comment
/B	-	/B	Not used.
/C	Include Channel type	/C	Returns channel type (eg. TK) before each data value (**/h** mode only)
/D	Add date to returned data	/d	Returns current date before each scan's data. (**/h** mode only)
/E	Echo	/E	Enables echo of commands to host computer (if not unloading data and not in fixed-format mode).
/F	Fix (lock) schedules	/f	Prevents a *DT80*'s scan schedules (trigger or channel list) being modified. Note that a reset still erases the schedules.
/H	Fixed-format (Host) mode	/h	Returns data in fixed-format mode *(P23)*. **Note** Setting **/H** will also set **/e/r**
/I	Schedule ID	/i	Returns schedule identifier before returning the schedule's data (**/h** mode only)
/K	Enable automatic re-calibration	/K	Before each scan the *DT80* checks for drift due to changes in ambient temperature and re-calibrates if required.
/L	*dataTaker* serial number prefix	/l	Returns "dataTaker DT80 *serial-num*" before each scan's data (**/h** mode only)
/M	Messages	/M	Enables error and warning messages to be returned to host
/N	Channel numbers	/N	Returns channel number before each data value (**/h** mode only)
/R	Return data	/R	Returns real-time data to the host computer.
/S	Synchronize to midnight	/S	Synchronizes all schedules' time intervals to midnight — for example, **RA1H** scans on the hour. See *Time Triggers — Synchronizing to Midnight* *(P51)*.
/T	Add time to returned data	/t	Returns current time before each scan's data. (**/h** mode only)
/U	Units text	/U	Appends measurement units to returned data (**/h** mode only) and makes error messages verbose
/W	Intermediate (working) channels	/w	Allows working channels (see **W** channel option *(P40)*) to be returned (for diagnostic purposes) but not logged.
/X	Progressive maxima and minima	/x	Displays the progressive maximum and minimum values for statistical channels on the built-in display only.
/Z	Stops alarm messages	/Z	Enables alarms to issue action text to host computer or printer.
//	Default switches	–	Sets all switches to their default state

Table 5: DT80 Switches

Setting Default Switch Values

All switch settings are cleared back to their power-on default values when a soft or hard reset (**RESET** or **SINGLEPUSH**) is performed, or the **//** command is sent.

However you can change the default value for any parameter using the startup profile. For example:

```
PROFILE "SWITCHES" "T"="ON"
```

will change the default setting for switch T to **on** (/T).

Startup Profile

The **startup profile** is a group of settings which are read and applied every time the *DT80* starts up after a hard reset (**SINGLEPUSH** or power-on). These settings relate to the environment in which the *DT80* operates, and include such things as communications parameters, network addresses, switch/parameter settings to suit the local region, and so on.

Structure

The startup profile is divided into a number of **sections**, each of which deals with a particular area, eg. host port, modem, switch settings, etc. Each section is identified by name, eg "**HOST_PORT**".

Each section then contains a number of **keys**. Each key has a name (eg "**BPS**") and a value (eg "**57600**").

The value of any of the defined keys can be viewed of changed using the **PROFILE** command.

The PROFILE Command

The **PROFILE** command syntax is as follows:

Command	Description
PROFILE	return current settings for all profile keys
PROFILE "*section*"	return current settings for all profile keys in *section*
PROFILE "*section*" "*key*"	return current value of specified profile key
PROFILE "*section*" "*key*"=	delete specified profile key (revert to default value)
PROFILE "*section*" "*key*"="*keystring*"	set specified profile key to *keystring*

For example:

```
PROFILE "HOST_PORT"
[HOST_PORT]
BPS = 57600
DATA_BITS = 8
STOP_BITS = 1
PARITY = NONE
FLOW = SOFTWARE
PROFILE "HOST_PORT" "BPS"="115200"
```

Note that this will not immediately change the host port baud rate, because the startup profile is not applied until the next reset. Once you have entered all your required profile setting changes, you would normally use **SINGLEPUSH** to force the *DT80* to load the new settings.

The *DT80* supports the following profile keys:

Section Name	Key Name	Legal Values	Factory Default	Comment
PARAMETERS	**P***n*	*integer* limits vary with parameter	varies with parameter	Power-on default value for specified parameter See *Table 4: DT80 Parameters* (P131).
SWITCHES	**A**, **B**, **C**,…**Z**	**OFF**, **ON**	varies with switch	Power-on default value for specified switch See Table 5: *DT80* Switches (P132)
HOST_PORT	**BPS**	**300**, **600**, **1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600**, **115200**	57600	Default serial parameters for Host RS-232 port See *Configuring the Host RS-232 Port* (P116)
	DATA_BITS	**7**, **8**	8	
	STOP_BITS	**1**, **2**	1	
	PARITY	**NONE**, **EVEN**, **ODD**	NONE	
	FLOW	**HARDWARE**, **SOFTWARE**, **BOTH**, **NONE**	SOFTWARE	
HOST_MODEM	**DIAL**	*string*	ATD	Issued as prefix to number specified in the **SETDIALOUTNUMBER** command
	INIT	*string*	ATE0Q1&D2S0=4&C1&S0	Automatically issued by the *DT80* to initialise a connected modem To disable automatic initialization, set to empty string.
	EXT_POWER_SWITCH	**0**, **-1**, **1**…**8**	0	0 = do not control modem power -1 = use **RELAY** channel to switch modem power 1..8 = use the specified digital output channel to switch modem power

Section Name	Key Name	Legal Values	Factory Default	Comment
	MAX_CD_IDLE	*integer* (seconds)	43200 (12 hours)	The number of seconds to wait while the **CD** signal is inactive before re-initialising the modem. Set to 0 to disable this function.
	SEND_BANNER_CONNECT	**YES**, **NO**	YES	When set to YES a string such as dataTaker 80 Version 5.08 is sent whenever the **CD** signal changes from inactive to active.
	COMMAND_PROCESSING_TIME	*integer* (seconds)	1	The number of seconds the *DT80* waits after sending a command to the modem (to give the modem time to respond).
ETHERNET	**IP_ADDRESS**	*n.n.n.n*	0.0.0.0	IP address to assign to the *DT80*'s Ethernet port
	SUBNET_MASK	*n.n.n.n*	255.255.255.0	Subnet mask for the network segment connected to the *DT80*'s Ethernet port.
	GATEWAY	*n.n.n.n*	0.0.0.0	IP address of the computer that acts as a gateway to other networks. Set to 0.0.0.0 if there is no gateway
PPP	**IP_ADDRESS**	*n.n.n.n*	1.2.3.4	IP address to assign to the *DT80*'s PPP over RS232 interface
	REMOTE_IP_ADDRESS	*n.n.n.n*	1.2.3.1	IP address to assign to the computer at the other end of the PPP link
	USER	*string*	ANONYMOUS	User name and password that a remote PPP client must supply in order to connect to the *DT80* via PPP.
	PASSWORD	*string*	PASSWORD	
FTP_SERVER	**SUPPORTED**	**YES**, **NO**	YES	If set to YES the *DT80*'s FTP server will be started
	USER	*string*	DATATAKER	Username and password that an FTP client must supply in order to be granted read/write access. (Anonymous FTP users have read-only access)
	PASSWORD	*string*	DATATAKER	
HTTP_SERVER	**DOC_ROOT**	*string*	C:\WWW	Base directory for web pages
	PORT	*integer*	80	TCP port number used by web server, set to 0 to disable web server.
STORE	**DEFAULT_DATA_STORE_SIZE**	*integer*	1048576	default space to allocate for a schedule's logged data (bytes)
	DEFAULT_ALARM_STORE_SIZE	*integer*	104800	default space to allocate for a schedule's logged alarms (bytes)

Table 6: DT80 PROFILE Details

Examples

To set the *DT80*'s IP address for the Ethernet port:

```
PROFILE "ETHERNET" "IP_ADDRESS"="192.168.1.225"
```

To set the default date format to North American style:

```
PROFILE "PARAMETERS" "P31"="2"
```

Note Characters used within strings are case sensitive. Therefore all settings should be in case shown (usually uppercase).

Note If a section or key name is misspelled, it will still be entered into the profile. However, when the profile is read (on the next hard reset) the invalid key will be ignored. No error message will be generated, and the factory default setting for the key will be used. Be careful.

USER.INI

All changed profile settings are stored in a text file, **B:\INI\USER.INI**. You can therefore display just the non-default profile settings by sending:

```
TYPE "B:\INI\USER.INI"
```

(Remember that if you are using DeTransfer then all backslashes must be entered as ****.)

Backup Copy of USER.INI

Each time the **USER.INI** file is updated (ie. a profile setting is modified using the **PROFILE** command), a second copy of the file is written to a hidden area of the *DT80*'s internal flash memory. If the **B:\INI\USER.INI** file is accidentally deleted (eg. using **FORMAT"B:"**), it will be automatically restored from the flash copy.

Reverting to Default Settings

The **DELUSERINI** command will delete both the **B:\INI\USER.INI** file and the backup copy in flash memory. It will therefore restore all profile settings back to their factory defaults.

Setting the *DT80*'s Clock/Calendar

The *DT80*'s real-time clock/calendar is based on a 24-hour clock that has a resolution of approximately 0.1ms. This is used to timestamp all logged data.

Time and date are maintained when the logger is switched off or reset. If the logger is switched off and the internal Memory-Backup battery (see *Internal Memory-Backup* Battery (P144)) is removed or discharged, then the date and time will be reset to 1989/01/01 00:00:00

D and T Channel Types

The *DT80*'s time and date can be set using the **D** and **T** internal channel types (*Time & Date* (P32)), eg:

```
D=1/4/2006
Date 01/04/2006

T=13:05
Time 13:05:00.000
```

The date and time must be specified in the current P32 and P39 formats.

DT Command

Alternatively, the **DT** command can be used to set both date and time. In this case the date/time is always specified in ISO format, eg:

```
DT=2006/04/01,13:05:00
```

Time Zone

Depending on the application, you may choose to set the *DT80*'s time to either:

- local time. In this case you may need to alter the time periodically to adjust for daylight saving time, if applicable.
- local standard time. The logger is still set to the local time zone, but no correction is made for daylight saving time.
- UTC (GMT) time. This may be appropriate if there are a number of *DT80*s (connected to the host computer via a wide area TCP/IP network, for example) which are in different time zones.

The *DT80* and the dataTaker host software currently do not provide any specific support for data collected in disparate time zones – it is therefore up to the user to manage this.

Resetting the *DT80*

The *DT80* can be **reset** in the following ways:

- **soft reset** – clears current job and resets parameters and switches
- **hard reset** – restarts *DT80* firmware
- **safe mode reset** (a.k.a. **triple push reset**) – restarts *DT80* firmware and runs using factory default settings.

Soft Reset

The **RESET** command performs a **soft reset**, which has the following effects:

- The message `Initializing...` is returned.
- The current job is cleared, ie. all schedule and channel definitions are cleared.
- All span/polynomial definitions are cleared.
- All CVs are reset to 0.0.
- All parameters and switches are reset to their power-on default values, as specified in the startup profile.
- All digital outputs are reset to their default state.
- All counter channels are reset to 0.
- A self-calibration is performed.

A soft reset is analogous to closing and restarting an application on a PC.

Hard Reset

A **hard reset** causes all *DT80* hardware to be physically reset, and the *DT80* firmware will be restarted. A hard reset may be caused by:

- the **SINGLEPUSH** command

- pressing the manual reset button (by inserting a straightened paper clip into the small hole between the Ethernet and USB connectors)
- loss of power (eg. if you disconnect external power and the battery link is not in place, or the main battery is completely flat)
- the *DT80* detecting a critical error such as a serious hardware fault, or an internal inconsistency in the firmware. In these situations a hard reset is forced in order to try to return the *DT80* to a stable operational state.

A hard reset is analogous to rebooting a PC.

A hard reset has the following effects:

- All TCP/IP (Ethernet or PPP over RS232) connections are terminated.
- USB connection is terminated if power was lost.
- All communications and other settings are read from the startup profile and applied.
- All LEDs flash rapidly four times.
- If the reset was due to a power failure, manual reset button or critical error, then a message is displayed on the LCD (eg. `DT80 restarted` / `Power loss`) and the **Attn** LED starts flashing.
- A sign-on message eg. `dataTaker 80 Version 5.08` is returned. (This can be disabled if required by switching off the messages flag in the startup profile, ie. **`PROFILE "SWITCHES" "M"="OFF"`**)
- A **soft reset** is performed, as described in the previous section.
- The ONRESET job (if any) is loaded, and becomes the current job.

Note A hard reset may take a few seconds to perform. You should refrain from sending further commands to the *DT80* during this time.

Safe Mode

If there is an error in the *DT80*'s startup profile or ONRESET then you may find that you are unable to communicate with the logger. For example – the startup profile specifies the wrong RS232 parameters (and you're not sure what they are), or the ONRESET job specifies an immediate **`1SERIAL`** channel with a very long timeout (and the serial channel is not connected to anything).

Safe Mode provides a mechanism to regain control and fix the problem. To select safe mode you need to perform a "triple-push" reset – press the manual reset button three times within 10 seconds.

This will have the same effect as a **hard reset**, except that:

- a message indicating safe mode is returned, and displayed on the LCD
- all startup profile settings are ignored – factory default settings are used (for example, the host port parameters will be set to **`57600,8,N,1,SWFC`**)
- the ONRESET job (if any) is not loaded.

Note that the profile settings and ONRESET job are not cleared – they are simply ignored for this session only. If you now do a normal hard reset they will once again be loaded.

To permanently delete the profile settings or ONRESET job, the **`DELUSERINI`** or **`DELONRESET`** commands can be used., or the **`FACTORYDEFAULTS`** command will delete both.

TEST Commands

The **TEST** command causes the *DT80* to perform a self-calibration, then run a series of self tests. Test results that are out of range are flagged with a FAIL message.

A typical test report is shown below. Note that a description of the various entries is beyond the scope of this manual.

```
TEST
TEST report generated at 2005/09/19,11:49:56
dataTaker 80 Version 5.02.0040 2005/09/15 15:51:36

Serial Number:    082005

VEXT                  13.7   V
VBAT (6V)             6.88   V   PASS
IBAT                   +19  mA   PASS
VSYS                  7.26   V   PASS
VLITH (3.6V)          3.65   V   PASS
VDD (3.3V)            3.25   V   PASS
VANA (3.8V)           3.98   V   PASS
VRELAY (4.5V)         4.65   V   PASS

VREF (2.5V)         2504.04  mV   PASS
Ics I               0.21305  mA   PASS
Ics II               2.5688  mA   PASS
Vos diff               -1.9  uV   PASS
Vos 3W                123.8  uV   PASS
Vos shunt               0.9  uV   PASS
Vos +                  -0.3  uV   PASS
Vos -                   1.0  uV   PASS
Vos *                 -29.4  uV   PASS
Vos #                  65.0  uV   PASS
Term. factor       1.00486        PASS
Shunt (100R)       100.112 Ohm    PASS
CMRR                 106.7  dB   PASS

DT80 health                       PASS
```

You can also request just one line of the report using **TEST***n*, where *n* is the line number (0-23). For example, the following will return the firmware version number:

```
TEST0
dataTaker 80 Version 5.02.0040 2005/09/15 15:51:36
```

Note that if the main or Lithium battery are absent then there will be FAIL lines in the output from the normal TEST command but this will not affect the overall pass/fail result.

Power on Self Test

The TEST command is also run automatically, following a hard reset. If this test fails then

```
Self test failed
```

is displayed on the LCD in conjunction with two flashing LEDs, and a message is output to comms port and event log. Press any key to clear the LCD message. You should then send a **TEST** command to see which particular test is failing.

If the power-on test passes there will be no indication (no report is generated).

Event Logs

To aid in troubleshooting, the *DT80* automatically logs significant events (power failures, resets, program failures, etc.) into an **event log**, which is a text file **B:\EVENTS\EVENT.LOG**.

The event log may help pinpoint the cause of any unexpected readings or failures, and will be used by *dataTaker* engineers if the *DT80* is returned for service.

In the event of an abnormal reset due to a firmware error, the *DT80* may store additional information in a companion file, the **error log** (**B:\EVENTS\ERROR.LOG**).

Unloading the Event and Error Logs

The contents of the event and error logs can be viewed using the **UEVTLOG** and **UERRLOG** commands.

If an abnormal reset occurs, it is recommended that you:

- use **UEVTLOG** and **UERRLOG** to unload both log files, and save the returned text to a file on your host PC (eg. by cutting and pasting from a DeTransfer window), and
- contact dataTaker support. You will probably be asked to send a copy of the saved log files for analysis.

Clearing the Event and Error Logs

The event and error logs can be cleared using the **CEVTLOG** and **CERRLOG** commands.

STATUS Commands

STATUS

The **STATUS** command returns a report showing the status of the *DT80*'s schedules, channels, alarms, memory and logging to the host computer. A typical report is shown below:

```
STATUS
dataTaker 80 Version 5.08.0002 Flash 2006/03/01 16:21:13
A B C,F Scan Schedules Active,Halted
4,0 Alarms/IFs Active,Halted
0 Polynomials/Spans Defined
A B C F,none Scan Schedules LOGON,LOGOFF
61650,1026 Internal kB free,used
0,0 External kB free,used
/B/C/d/E/f/h/I/K/l/M/N/r/S/t/U/w/x/Z
```

If the **/u** switch is set (don't display units) then the descriptive text on the end of each line is not returned.

STATUS*n*

As with the **TEST** command, lines in the status report can be returned individually, using **STATUS***n*.

Command	Description
STATUS1	returns model name and firmware version (same as **TEST0**)
STATUS2	lists active schedules, and halted schedules
STATUS3	returns the number of alarms in active schedules, and the number in halted schedules
STATUS4	returns the number of polynomial/spans defined, and displays the definition of each
STATUS5	lists schedules with logging enabled, and schedules with logging disabled
STATUS6	returns total free space (kbytes) on internal file system, and total used space
STATUS7	returns total free space (kbytes) on inserted USB memory device, and total used space
STATUS9	returns current settings for all switches
STATUS10	returns internal details about the current job
STATUS14	an extended version of **STATUS10**

The **STATUS14** command is somewhat special in that it can also be applied to a non-current job, ie:

STATUS14 "*jobname***"**

Part M — Hardware and Power

Inputs and Outputs

DT80 Front Panel

Figure 39 DT80 Front Panel

The top face of the *DT80* is the user interface, which comprises:

- 2 Line LCD Display *(P93)*
- Directional Keypad *(P96)*
- USB Stick Interface *(P89)*
- Sampling Indicator *(P96)*
- Internal Disk Indicator *(P96)*
- Attention Indicator *(P97)*
- Edit of Confirm Key *(P96)*
- Func or Reject Key *(P96)*

DT80 Wiring Panel

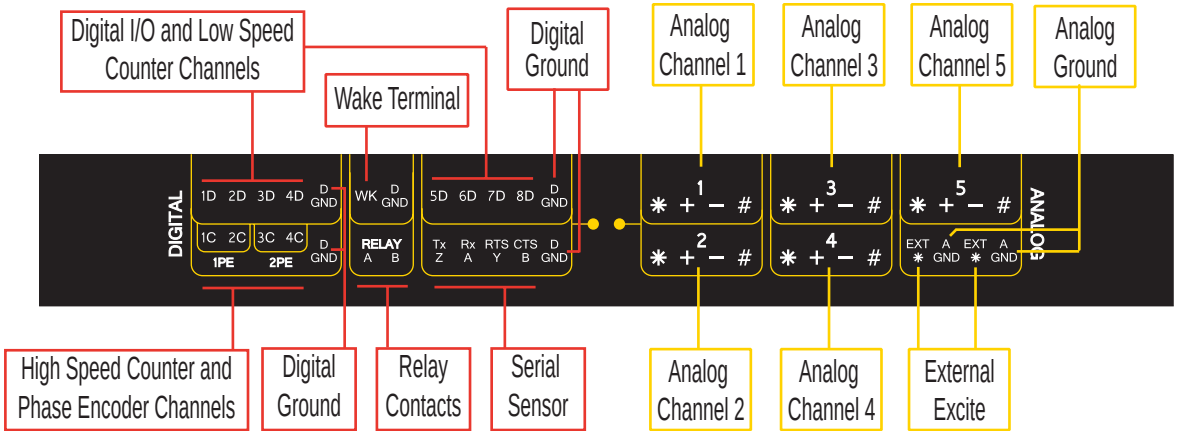


Figure 40 DT80 Wiring Panel

The front face of the *DT80* is the sensor interface, which comprises:

- Digital Input/Output Channels *(P155)*
- Wake Terminal *(P145)*
- Digital Ground *(P154)*
- Counter Inputs *(P164)*, shared with Phase Encoder Inputs *(P165)*
- Relay Output *(P158)*
- Serial Sensor Port *(P166)*
- External Excitation Input *(P20)*
- Analog Inputs *(P147)*

DT80 Side Panel

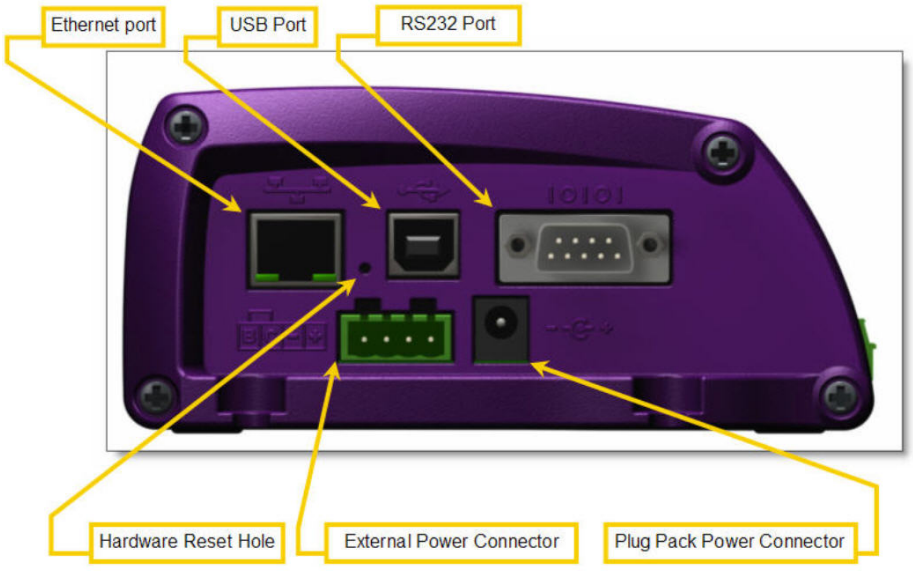


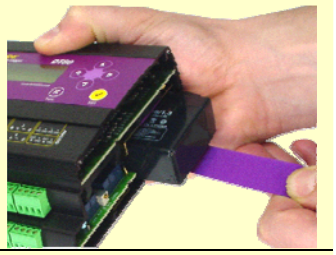
Figure 41 DT80 Side Panel

The *DT80*'s side panel provides communications and power interfaces:

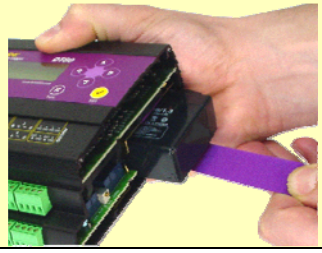
- Ethernet Port *(P122)*
- USB Port *(P114)*
- RS232 Port *(P115)*
- Hardware Reset Hole *(P135)*
- External Power Connector *(P144)*
- Plug Pack Power Connector *(P144)*

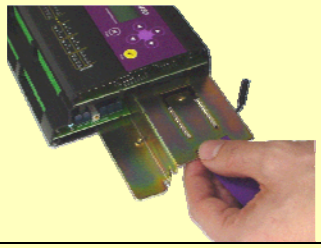
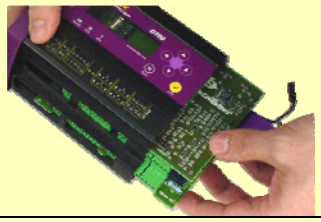
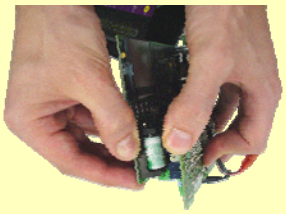
INSIDE THE *DT80*

Accessing the main battery

1. Remove the power connector	
2. Remove the screws from the other end of the logger	
3. Remove this end of the logger	
4. Pull the purple 'battery tail'	
5. Disconnect the battery terminals	

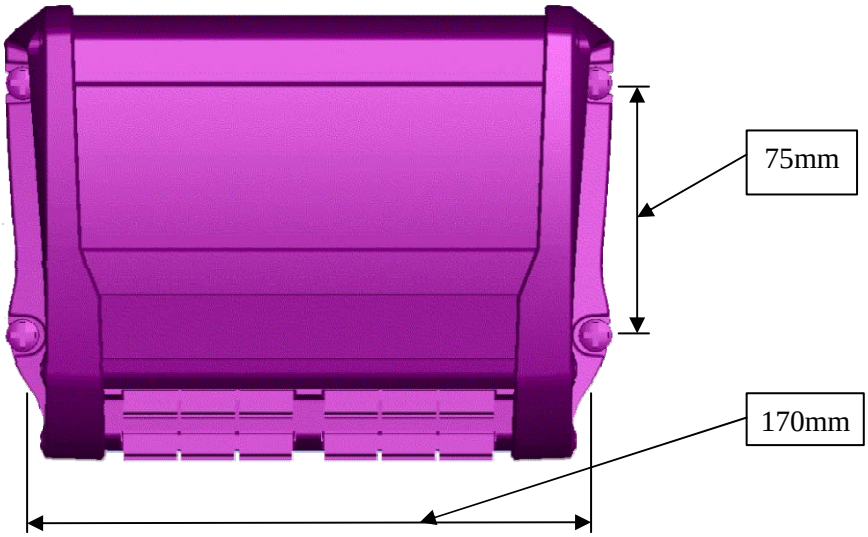
Accessing the lithium memory backup battery

1. Remove the power connector	
2. Remove all the terminal blocks	
3.	
4. Remove the screws from the other end of the logger	
5. Remove this end of the logger	
6. Pull the purple 'battery tail'	

7. Disconnect the battery terminals	
8. Remove battery bracket locking screw (underneath logger) and then remove the battery cage.	
9. Remove the circuit board bundle	
10. Open the boards slightly so the lithium battery can be removed.	

Mounting the *DT80*

Dimensions, Clearances



Powering the *DT80*

External Power

The *DT80* is normally powered by an external 10-30V DC supply. This may be connected in one of two ways:

- the round plug pack power socket (inner pin is positive)
- the rightmost two terminals (**-** and **+**) of the adjacent 4-way removable screw terminal power connector. This provides a more robust connection.

Internally these two connectors are wired in parallel, so you can for example power the unit via the plugpack and then draw power from the screw terminals for powering external relays or sensors.

Internal Power (Main Battery)

The *DT80* is fitted with an internal **6V 1.2Ah** sealed lead-acid gel-cell battery. It's known as the *DT80*'s "main" battery to distinguish it from the *DT80*'s other internal battery, the "memory-backup" battery.

The main battery is completely maintenance-free and rechargeable, being automatically charged by the *dataTaker* data logger's inbuilt battery charger whenever an external power supply is connected to the *DT80*. If properly cared for (which essentially means keeping it charged), the battery should give several years' service.

If the main battery ever needs to be replaced, *Inside the DT80* (P141) explains how to do so.

Connect the Battery Link

The *DT80* is shipped with the main battery disconnected.

To connect the battery, all you need to do is plug the supplied 4-way terminal block into the power connector on the side of the *DT80*. The supplied terminal block includes a link which connects the **B** and **C** terminals on the power connector. This will connect up the internal battery to the *DT80* circuitry.

Note It is recommended that the battery link be left permanently attached to the *DT80* during operation. This guarantees uninterrupted data acquisition and logging because the internal main battery is always available to continue powering the *dataTaker* data logger if the primary/external supply is accidentally disconnected or fails.

Main Battery Life

The life of the *DT80*'s internal main battery depends on ###

- the scan interval
- the number of channels being scanned
- the number of alarms
- excitation power drawn by sensors
- the complexity of any calculations
- communications activity.

Storage

If the *DT80* is not to be used for a period of time, consideration needs to be given to the health of its internal battery.

Important Avoid storing the internal battery in a discharged state. If a gel-cell battery remains flat for any length of time, its capacity and service life will be significantly reduced.

Before placing a *DT80* into storage, you should therefore ensure that the main battery is fully charged (at least eight hours charge time). The battery link should then be disconnected.

Internal Memory-Backup Battery

In addition to the internal main battery, the *DT80* contains a small lithium "memory-backup" battery.

The memory-backup battery maintains the *DT80*'s clock/calendar and certain memory settings. (Note that the *DT80*'s internal file system, which stores programs and logged data, uses non-volatile flash memory. This does not depend on the memory-backup battery.)

Replacing the Battery

Under normal operation the memory-backup battery should last approximately five years, or approximately one year if there is no other power to the *DT80* (ie. both external power and the main battery are disconnected).

See *Inside the DT80* (P141) for details on how to remove and replace the internal memory-backup battery. The memory-backup battery is a 1/2AA size 3.6V lithium type (for example, SAFT LS 14250). It's important that 3.6V and not 3.0V types be used (both types are the same physical size).

Storage

If the *DT80* is to be placed in long term storage, it is recommended that the memory-backup battery be removed, to keep it from discharging. When disconnected, the battery has a 10-year shelf life.

Monitoring *DT80* Power

The *DT80* provides a number of internal channel types for monitoring the various power systems. These can be queried at any time or used in alarms, like any other channel type.

The following channel types are available:

Channel Type	Units	Description
VEXT	V	External power supply voltage
VBAT	V	Main battery terminal voltage
IBAT	mA	Instantaneous main battery current – positive if charging, negative if discharging
VLITH	V	Memory-backup battery voltage

Note that

- **VEXT** will read about 1V under the actual input voltage, due to a series diode
- **IBAT** will generally read substantially higher than the actual average battery discharge current. This is due to the fact that the *DT80*'s microprocessor switches to a lower power mode when idle, but at the instant of measurement it will always be non-idle, and hence in the normal (higher power) mode.

Low-Power Operation

Sleep Mode

During normal operation the *DT80* typically draws approximately 150mA from the main battery (400mA max).

However, the *DT80* also has a low power "sleep" mode that reduces battery current to just 350µA. While asleep no measurements or processing can be done, but the state of the current job is preserved. The *DT80* will automatically wake from sleep when a measurement is due, or some other event occurs.

For applications where power consumption is critical, it is therefore vital to ensure that the *DT80* does not wake more often than necessary.

Wake Events

Once asleep, the *DT80* will stay that way until one of the following events occur:

- a scheduled scan becomes due
- a keypad button is pressed
- a USB memory device is inserted or removed
- the **WK** (wake) input terminal is pulled to logic low
- a character is received on the serial sensor port, or there is a transition on the CTS line
- external power is connected
- a USB communications cable is connected
- a character is received on the host RS232 port

Any of these will cause the *DT80* to wake. If the reason for waking was a scheduled scan, the *DT80* will execute the schedule, then immediately go back to sleep (if there are no other schedules due within the next few seconds).

For all other wake sources, the *DT80* will stay awake for at least the period specified by parameter **P17** (default 30s). This timer will be reset if any further wake events occur, or if data is received on any comms port. Once the timer expires the *DT80* will go back to sleep.

Points to Note

- If the *DT80* is woken by receipt of RS232 data on the host or serial sensor port, the first character (and possibly some of the following ones) will be lost. You should therefore always send a dummy character (eg. CR) to wake the *DT80*, then wait about 0.5s before sending the first actual command. (DeTransfer will do this automatically if you set the Enable Wakeup option)

- Digital inputs are not scanned while asleep, so event-triggered schedules (eg **RA1-E**) cannot be used to wake the logger.
- High speed counters (*n***HSC**) continue to count in sleep mode, but software counters (*n***C**) do not.

Controlling Sleep

By default, the *DT80* will only go into sleep mode if:

- it is battery powered (ie. external power is not connected), and
- the Ethernet port is not connected, and
- the USB port is not connected.

Note that any Ethernet and USB connections are terminated whenever the *DT80* goes to sleep – which is why the *DT80* will by default disallow sleep while either of these ports are connected.

The above conditions can be overridden using the **P15** parameter, as follows.

Setting	Description
P15=0	Allow sleep if battery powered and Ethernet/USB not connected (default)
P15=1	Allow sleep if Ethernet/USB not connected
P15=2	Do not allow sleep
P15=3	Allow sleep
P15=4	Allow sleep if battery powered

For example, if you set **P15=3** then the *DT80* will always be allowed to go to sleep, even if the logger is externally powered or Ethernet/USB is connected.

Maximising Battery Life

Given that sleep mode uses roughly one thousandth the power of normal operation, the key factor in maximising battery life is maximising the time that the *DT80* spends sleeping.

Note also that the *DT80* takes approximately 4 seconds to resume normal operation following a wakeup event.

The following guidelines will help maximise battery life:

- Scan as slowly as possible – don't scan every minute if you can get away with scanning every 5 minutes
- Align schedule intervals to minimise the number of wakeups, even if this means that some schedules sample more frequently. For example:
  RA40S 1V RB20S 2V
  is better than
  RA40S 1V RB30S 2V
  because the two schedules are more likely to be processed together.
- reduce the **P17** setting (say **P17=5**) so that if a wakeup event does occur, the logger will go back to sleep quickly.

Forced Sleep Mode

The *DT80* provides some protection against gradual power failure (eg. the internal battery becoming discharged). If it detects that the supply voltage is becoming critically low, the *DT80* will automatically close all store files and force the unit into sleep mode. The *DT80* will remain asleep until the power supply recovers to an adequate level. During this time schedules will not execute.

An entry will be added to the event log *(P137)* any time that the *DT80* enters forced sleep mode.

Operating Environment

The *DT80* is an electronic instrument. Electronics and water in any form do not mix. Condensation can be a serious problem in the tropics, and in cooler areas where wide temperature variations are possible. Use a sealed case and include sachets of silica gel to avoid problems.

If the *DT80* gets wet, immediately disconnect and remove all power sources (including the main internal battery), and dry the *DT80* in a warm place. If the unit comes into contact with salt water, rinse it thoroughly in fresh water, then in distilled water, then dry it — salt must NOT be allowed to remain on the circuit boards.

The *DT80* operates over a wide temperature range (–45°C to +70°C), but its accuracy can be reduced at extremes. While the electrical zero is stable with temperature, the scale factor can drift slightly. Try to minimize the *DT80*'s exposure to temperature extremes. The lead Acid battery in the logger best operates between -15°C to 20°C. When operating outside this range consideration must be given to an alternative power source for the logger.

Part N — Sensors and Channels

This section discusses:

- interfacing various common types of sensor to the *DT80*'s analog channels
- digital and counter channels (including SDI-12)
- the serial channel
- wiring configurations for analog channel types
- wiring configurations for digital channel types

Analog Channels

4–20mA Current Loops

Wiring Diagrams: see *Current Inputs* (P179)

Many different sensor types provide a 4-20mA current output, where the current is proportional to the quantity being measured.

Each *DT80* analog input channel has an internal 100R shunt resistor connected between the **#** terminal and **AGND**. External shunt resistors can be used to expand the number of 4-20mA sensors that can be used per *DT80* analog channel.

The channel type for 4–20mA current loop measurement is

L(*shuntR***)**

where *shuntR* (the channel factor) is the value of the shunt resistor in the loop (default 100R)

Current-loop measurement is essentially the same as voltage measurement — the *DT80* measures the voltage across the internal or external shunt resistor and, knowing the shunt resistance, calculates the loop current. This is then scaled and returned as a percentage 0% for a measured current of 4mA and 100% for 20mA. A span is often applied to this value so that the final reported value is in the proper engineering units for the quantity being measured.

For example, if a pressure sensor with 4-20mA output operated over a range of 100-500kPa then

```
BEGIN S1=100,500"kPa" RA2S 4#L(S1) END
```

will return a pressure reading in kPa every 2s. The loop would in this case be connected across the **4#** and **AGND** terminals, making use of the internal shunt resistor.

Frequency

Wiring Diagrams: see *Voltage Inputs* (P178)

The frequency of an analog input signal can be measured using the **F** channel type, which returns a value in Hz.

Useful channel options for **F** channels are:

Channel Option	Description
(channel factor)	sample period (gate time) in ms (default is **30ms**)
2V	offset input signal by -2.5V. This effectively changes the threshold point from 0V to approx. +2.5V, which is useful for TTL level inputs

The range of frequencies that can be measured depends on the configured sample period (channel factor). For the default setting of 30ms, this range is approximately 25Hz – 20kHz. If the input frequency is too low to be measured, the underrange error value (`-99999.9`) will be returned.

To measure lower frequencies, the sample period should be increased. For example

```
3F(1000)
```

will measure down to 1Hz (upper limit is still 20kHz), while

```
3F(10000)
```

will allow frequencies down to 0.1Hz to be resolved.

The drawback to selecting a long sample period is that the measurement will take a long time to complete. This may delay the execution of other schedules.

Note that the default threshold point is 0V, so the input signal must have zero crossings in order to be measured. If this is not the case (eg. for a logic signal), the **2V** channel option can be used to change the threshold point to +2.5V.

Period Measurement

The period of a signal can be measured by taking the reciprocal of a frequency measurement, eg:

```
RA5S 3+F(2V,1000,F1,"Period~s",FF4)
```

will return the period, in seconds, of an TTL-level logic signal connected between **3+** and **3#**. Given the 1000ms sample period, the maximum period that can be returned will be approximately 1.0s. The **F1** option applies intrinsic function #1 (1/x).

Thermocouples

Wiring Diagrams: see *Voltage Inputs* (P178)

Thermocouple Theory

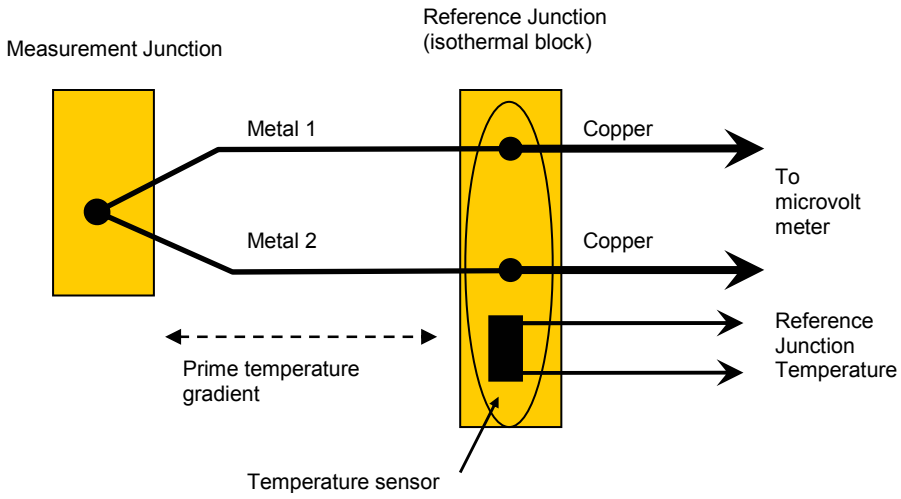


Figure 42: Thermocouple principle of operation

A thermocouple consists of two wires of dissimilar metals that are

- electrically connected at one end (the measurement junction) and
- thermally connected at the other end (the reference junction).
- A small voltage is produced when the two junctions are at different temperatures. (The voltage is produced by the temperature gradient along the wires, not by the junctions.)

It's important that the purity of the thermocouple wire be maintained where significant temperature gradients occur. Because high purity wire can be expensive, it's common practice to use thermocouple extension wire to cover long distances where temperatures are within the normal environmental range. Such wire can be used for measurement junctions, but only over a restricted temperature range of typically –20°C to 120°C.

Making the Measurement Junction

The measurement junction can be made by welding, brazing, soldering or crimping the two wires together. Take care to ensure that the wire material is not contaminated where the temperature gradient is to occur.

The junction can be insulated, or left bare for a more rapid response. If left bare, ensure that the junction does not make intermittent contact with metal objects. This can introduce electrical noise.

Sometimes thermocouple measurement junctions are electrically connected (by welding, brazing, soldering or by contact) to the object being measured. This is only possible if the object is grounded to the *DT80*'s analog ground terminal **AGND**, or if the voltage on the object relative to **AGND** is within the *DT80*'s common mode limits.

Reference Junction Compensation

Conventionally, the reference junction is held at 0°C, and thermocouple responses are determined with a 0°C reference. This is inconvenient in most situations and so, in practice, the reference junction is allowed to follow to ambient temperature. Then this non-zero reference junction temperature must be compensated for by measuring the reference temperature with another temperature sensor.

The *DT80* makes this correction in software. The software approach allows support for any thermocouple type without hardware dependence.

Isothermal Block

Generally the reference junctions and the associated temperature sensor are held at the same temperature by a physical arrangement that ensures good thermal conductivity between the junctions. This structure is called an "isothermal block". It

is advisable to insulate the isothermal block from rapid ambient temperature changes.

Thermocouple Types

The *DT80* supports all commonly-recognized thermocouple types:

Type	Positive	Negative	Range °C		
B	Pt, 30%Rh	Pt, 6%Rh	+300	to	1700
C	W, 5%Re	W, 26% Re	0	to	2320
D	W, 3%Re	W, 25%Re	0	to	2320
E	Ni, 10%Cr	Cu, 45%Ni	–200	to	900
G	W	W, 26% Re	0	to	2320
J	Fe	Cu, 45% Ni	–200	to	750
K	Ni, 10%Cr	Ni,2%Mn, 2%Al	–200	to	1250
N	Ni, 14%Cr, 1%Si	Ni, 4%Si, 0.1%Mg	–200	to	1350
R	Pt, 13%Rh	Pt	0	to	1450
S	Pt, 10%Rh	Pt	0	to	1450
T	Cu	Cu, 45%Ni	–200	to	350

Each type has characteristics (sensitivity, stability, temperature range, robustness and cost) that make it appropriate for particular applications.

The *DT80*'s thermocouple linearisation data are based on the ITS90 International Temperature Scale.

Using Thermocouples with the DT80

Thermocouples are wired to the *DT80* as for any other voltage input. The channel type is a **T***t* where *t* is the thermocouple type. So to measure a K-type thermocouple you would use the **TK** channel type.

Using the thermocouple channel type reads the channel as a voltage and automatically applies cold junction compensation and linearization.

Reference Junction and Isothermal Block Support

By default, the *DT80* uses an internal LM35 temperature sensor to measure the reference junction temperature. You can check the reading of this sensor using the **REFT** channel type.

For higher accuracy measurements, the *DT80* also supports the use of an external isothermal block. In this case the isothermal block's temperature sensor is measured by a separate *DT80* channel. This channel uses the **TR** channel option to identify it as the temperature reference – its reading will then be used as the reference for all subsequent thermocouple measurements in that schedule.

For example:

```
RA10S 5AD590(TR) 1..4TT
```

In this example four thermocouples are measured. Their reference junctions are enclosed in an isothermal block, along with an AD590 temperature sensor, which is connected to analog channel 5.

Accuracy — Thermocouple Techniques

The accuracy of temperature measurement with thermocouples depends on

- the reference junction isothermal characteristics
- the reference temperature sensor accuracy
- induced electrical noise
- the quality of the thermocouple wire
- drift in the wire characteristics, especially at high temperatures
- the basic measurement accuracy of the *DT80*
- the linearization accuracy of the *DT80*.

Reference Junction Error

The most significant source of error is the reference junction. The *DT80* must not be exposed to non-uniform heating because a single reference temperature sensor is used to measure the temperature of the terminals of all channels. If a temperature gradient occurs along the terminals, errors of the magnitude of the temperature difference occur.

The *DT80*'s reference temperature sensor is positioned behind analog channels 2 and 4. Therefore, when precise temperature measurements are required, attach thermocouples here for the least temperature differential from the *dataTaker*'s reference temperature.

Linearization Error

The *DT80*'s linearization errors are much lower (< 0.1°C over the full range) than other error sources.

Thermistors

Wiring Diagrams: see *Resistance Inputs* (P180)

Thermistors are devices that change their electrical resistance with temperature. They measure temperatures from –80°C up to 250°C, and are sensitive but highly nonlinear. The *DT80* has channel types for many 2-wire YSI (Yellow Springs Instruments) thermistors and, for other thermistor types, the *DT80* supports thermistor scaling — see *Thermistor Scaling* (P59).

Channel Type	R (ohms) at 25°C	YSI Thermistor	Max. Temp °C	Min. Temp °C (without Rp)
YS01	100	44001A, 44101A	100	–65
YS02	300	44002A, 44102A	100	–45
YS03	1000	44003A, 44101A	100	–20
		44035	100	
YS04	2252	44004, 44104	150	1
		44033	75	
		45004, 46004	200	
		46033, 46043		
		44901	90	
		44902	70	
YS05	3000	44005, 44105	150	7
		44030	75	
		45005, 46005	200	
		46030, 46040		
		44903	90	
		44904	70	
YS07	5000	44007, 44107	150	18
		44034	75	
		45007, 46007	250	
		46034, 46044		
		44905	90	
		44906	70	
YS17	6000	44017	150	22
		45017	250	
		46017	200	
		46037, 46047		
YS16	10k	44016	150	34
		44036	75	
		46036	200	
YS06	10k	44006, 44106	150	35
		44031	75	
		45006	250	
		46006	200	
		46031, 46041		
		44907	90	
		44908	70	

RTDs

Wiring Diagrams: see *Resistance Inputs* (P180)

Resistance Temperature Detectors are sensors generally made from a pure (or lightly doped) metal whose electrical resistance increases with temperature. Provided that the element is not mechanically stressed and is not contaminated by impurities, the devices are stable, reliable and accurate.

The *DT80* supports four RTD types:

Channel Type	Metal	Alpha	Standard
PT385	Platinum (PT385)	α = 0.003850	DIN43760
PT392	Platinum (PT392)	α = 0.003916	JIS C1604
NI	Nickel (Ni)	α = 0.005001	
CU	Copper (Cu)	α = 0.00390	

The alpha is defined by

$$\alpha = \frac{R_{100} - R_0}{100 R_0} \quad \Omega/\Omega/^\circ C$$

where R_o is the resistance at 0°C and R_{100} is the resistance at 100°C.

The 0°C resistance is assumed to be 100Ω for platinum, and 1000Ω for nickel types. Other values can be specified as a channel option.

The RTD channel types are connected as for a resistance, so you can choose a 2, 3 or 4 wire measurement (3W is the default).

For example,

```
PT385(4W,50)
```

reads a 4-wire 50Ω (at 0°C) device.

IC Temperature Sensors

IC (Integrated Circuit) temperature sensors are devices that are constructed on small silicon chips. These are linear, sensitive and available in both voltage and current output configurations. Sometimes called "monolithic" sensors. Their disadvantages are

- limited temperature range; generally –40°C to +150°C (like thermistors)
- self-heating from power dissipation caused by the excitation current needed to read the sensor.

The *DT80* supports the following commonly-available IC temperature sensors:

Sensor Family	Channel Type	Output
Semiconductor current source types (Analog Devices) **Wiring:** *AD590-Series Inputs* (P183)	**AD590**	1µA/K
	AD592	1µA/K
	TMP17	1µA/K
Semiconductor voltage output types (National Semiconductor Corp.) **Wiring:** *LM135-Series Inputs* (P185)	**LM135**	10mV/°C
	LM235	10mV/°C
	LM335	10mV/K
Semiconductor voltage output types (National Semiconductor Corp., Analog Devices) **Wiring:** *LM35-Series Inputs* (P184)	**LM34**	10mV/°F
	LM35	10mV/°C
	LM45	10mV/°C
	LM50	10mV/°C + 500mV
	LM60	6.25mV/°C + 424mV
	TMP35	10mV/°C
	TMP36	10mV/°C + 500mV
	TMP37	20mV/°C

Calibration

IC temperature sensors have different calibration grades. The lowest grades typically have an error of up to ±2°C at 25°C. More expensive sensors have an error of ±0.25°C. This error is a combination of an offset (or zero) error and a slope error.

For **LM135 series** sensors, the channel factor is a scaling factor, which can be used to correct a slope error based on a single point calibration. For example, if an LM135 reads 22.6°C when the actual temperature is 24.0°C (an error of -1.4°C) then the required scaling factor would be 1 – (–1.4 / 24.0) = 1.0583, which would be applied as:

```
2LM135(1.0583)
```

For **AD590 series** sensors, the channel factor is the value of the shunt resistor used to measure the current output. This can be used as a calibration factor. For example, if the sensor reads 290.7K when the actual temperature is 289.5K (an error of +1.2K) then the required scaling factor would be 1 – (1.2 / 289.5) = 0.9959, which would then be multiplied by the nominal shunt resistance (100). So the correction would be applied as:

```
1AD590(99.59)
```

Note that if the *DT80*'s internal shunt is used (eg. **1#AD590**), then you need to specify the shunt resistors nominal resistance as the channel factor when doing the calibration measurement, ie. **1#AD590(100)**. If this is not done then the *DT80* will use the actual shunt resistance (which it determines during its self calibration process), which will upset the above calculation because you won't know what to multiply the scaling factor by.

For **LM35** series sensors, the channel factor is an offset correction, in °C. So if the sensor reads 25.4°C when the actual temperature is 25.0°C (an error of +0.4°C) then the channel factor would simply be specified as:

```
4+LM35(0.4)
```

Bridges

Wiring Diagrams: see *Bridge Inputs – Voltage Excitation* (P181)

Because of its sensitivity, the Wheatstone bridge circuit is commonly-used for the measurement of small changes in electrical resistance. Applications include load cells, pressure sensors and strain gauges.

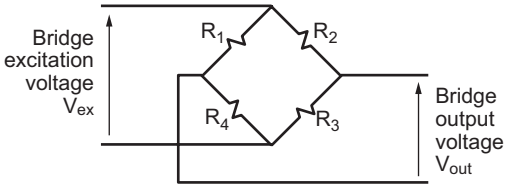


Figure 43: Wheatstone bridge

When one of the four resistors in a bridge is active (that is, sensitive to the quantity being measured) the circuit is called a **quarter bridge**, and the remaining three resistors are called **bridge completion resistors**. Similarly, **half** and **full** bridges imply two and four active gauges.

Bridge Excitation (Lead Compensation)

The bridge is a ratiometric circuit where the output sensitivity is proportional to the excitation voltage. Unfortunately, the excitation voltage is reduced by resistive cable and connector voltage drops. There are two ways the *DT80* can resolve this problem:

Voltage Excitation BGV

The *DT80* can measure the excitation voltage at the bridge and compensate numerically for the lead voltage loss. This requires a 6-wire connection with the **BGV** channel type (see *B1 – 6-Wire* BGV Inputs (P182)). This is termed **voltage excitation**.

Constant-Current Excitation BGI

The alternative lead compensation method is to apply a constant-current (2.5mA or 200uA) to the bridge — assuming the bridge resistance is known and constant — and then calculate the excitation voltage Vex.

For full and half bridge constant current excitation use the *n***BGI(***Ra***)** channel type, where *Ra* is the bridge arm resistance in ohms. If the arm resistances are not equal, a correction must be applied.

For the full bridge, all four resistors are external to the *DT80*. One or more of these resistors may be active, and the remainder are completion resistors. Four connection wires are required so that the **4W** channel option is required. For example, *n***BGI(4W,120)** defines a 4-wire constant-current bridge with an arm resistance of 120 ohms.

For the half bridge, bridge completion resistors are external to the *DT80*.

Scaling

The *DT80* scales all bridge channel types to a ratiometric form with units of parts per million (ppm):

$$\text{Reading } B_{out} = \left(\frac{V_{out}}{V_{ex}}\right) 10^6 \quad \text{ppm}$$

where:

- **V_{out}** is the measured bridge output voltage
- **V_{ex}** is the excitation voltage

For a **BGV** channel, V_{ex} is measured; for **BGI**, V_{ex} is calculated from the known current and arm resistance values.

To convert to other engineering units, apply a polynomial, span or use calculations (see *Manipulating Data* (P58)).

Strain Gauges

Strain gauges change resistance when stretched or compressed, and are commonly wired in a bridge. The strain-to-resistance relationship is

$$\text{Strain} = \frac{\Delta L}{L} = \frac{1}{G} \cdot \frac{\Delta R}{R}$$

where:

- **L** is the original length
- **ΔL** is the length change
- **R** is the original gauge resistance
- **ΔR** is the gauge resistance change
- **G** is the gauge factor, a measure of the sensitivity of the gauge (typical foil gauges have a gauge factor of 2.0, which means that if they are stretched by 1% their resistance changes by 2%)

To convert the *DT80*'s ppm bridge readings to strain, use the formula:

$$\text{Bridge reading in microStrain} = \left(\frac{4}{G \times N}\right) B_{out}$$

where

- **B_{out}** is the *DT80*'s bridge channel (BGV or BGI) result (ppm)
- **G** is the gauge factor
- **N** is the number of active gauges in the bridge

The conversion can be done in the *DT80* by applying a polynomial as a channel option (see *Polynomials* (P59)):

```
Y1=0,k"uStrain"  ' Polynomial definition
3BGV(Y1)         ' Bridge channel
```

where *k* = 4 / GN

Humidity Sensors

Relative humidity is commonly measured by the "wet bulb depression" method. Two temperature sensors are required, one to measure air temperature and the other the cooling effect of a wetted surface. Usually a temperature sensor is encased in a wick extending into a reservoir of distilled water. The temperature difference between the two sensors is the wet bulb depression.

The choice of temperature sensors is critical if reasonable accuracy is required at high relative humidity where the wet bulb depression is small. If platinum RTDs are used they should have good accuracy or matching (0.2°C).

Good accuracy can also be achieved by use of a temperature difference sensor such as a thermocouple or thermopile. Measure the dry bulb with a standard grade temperature sensor and subtract the difference sensor reading to obtain the wet bulb temperature.

The sensors are normally placed within a radiation screen to prevent radiant heat affecting the readings. This is particularly important for outdoor applications.

Example — Humidity Measurement

The following program reads two RTDs and calculates the relative humidity with an accuracy of a few percent for temperature above 5°C and over most of the relative humidity range (the algorithm assumes that the sensors are ventilated but not aspirated):

```
BEGIN"STICKY"
Y1=6.1,0.44,0.014,2.71E-4,2.73E-6,2.75E-8 'SVP polynomial
RA5S
 1PT385("Dry bulb",4W,=1CV)
 2PT385("Wet bulb",4W,=2CV)
 3CV(Y1,W)=1CV
 4CV(Y1,W)=2CV
 5CV("RH%",FF1)=(4CV-0.8*(1CV-2CV))/3CV
END
```

Analog Logic State Inputs

The *n***AS** channel type configures analog channel *n* to detect an input voltage relative to a threshold:

- When the input is above the threshold, **1** is returned.
- When the input is below the threshold, **0** is returned.

The default threshold is 2500mV, but can be set to any value in mV.

For example:

```
1AS(1750)
```

configures analog channel **1** as an analog state input with a threshold of 1.75V.

DT80 Analog Sub-System

A block diagram of the *DT80*'s analog sub-system is shown below.

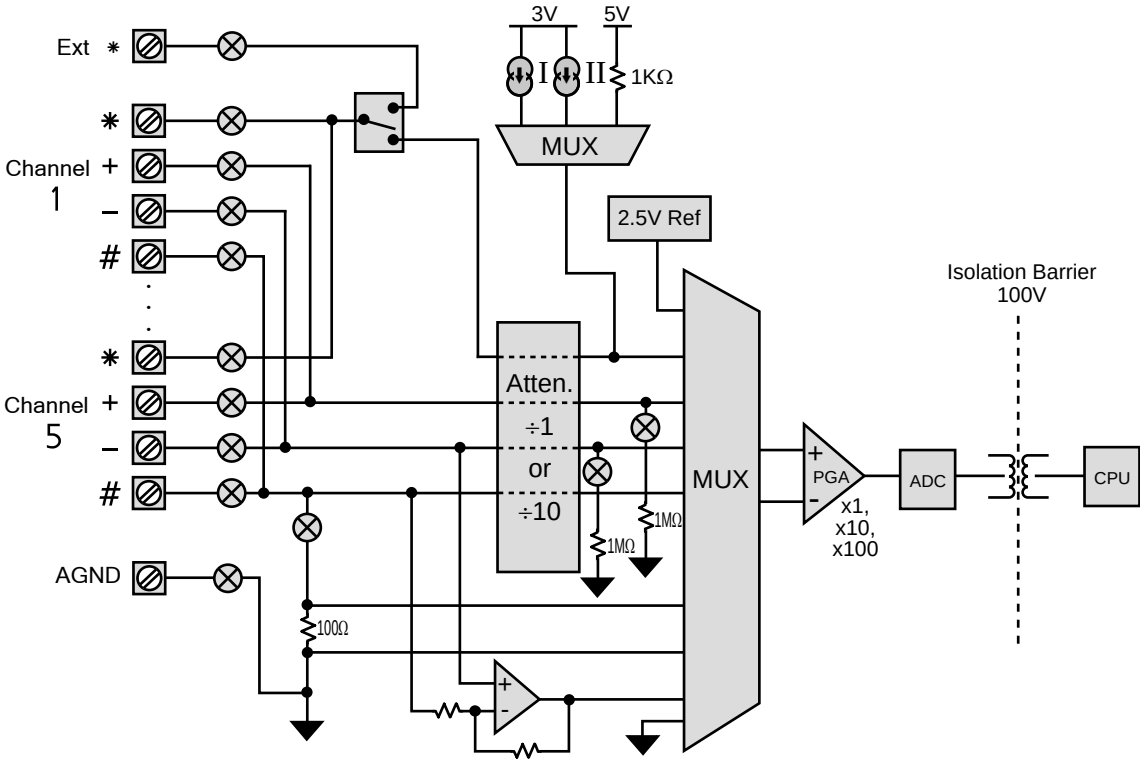


Figure 44: DT80 Analog Sub-System

Note that the analog section is electrically isolated from the rest of the *DT80*. This means that sensor-to-equipment ground loops (see *Grounds, Ground Loops and Isolation* (P24)) are unlikely to arise.

DT80 Ground Terminals

The *DT80* has two separate grounds — **digital ground** and **analog ground**. To preserve the *DT80*'s isolation these grounds should not normally be connected together.

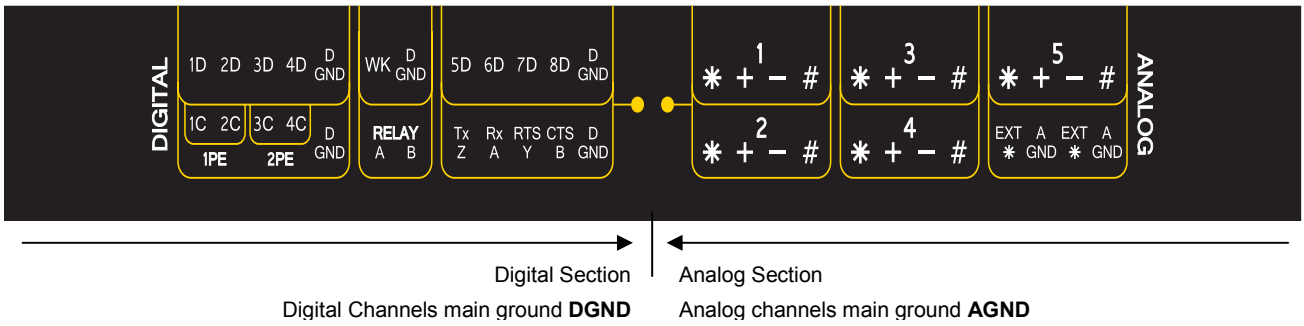


Figure 45: The DT80 has two ground systems

Digital Channels

The *DT80* provides:

- 4 bidirectional digital I/O channels (**1D**-**4D**) with open drain output driver and pull-up resistor (3 channels **1D-3D** for DT81);
- 4 bidirectional digital I/O channels (**5D**-**8D**) with tri-stateable output driver and weak pull-down resistor (SDI-12 compatible) (1 channel **4D** for DT81);
- 1 voltage free latching relay contact output (**RELAY**)
- 1 LED output (**Attn**)
- 4 hardware counter inputs (**1C**-**4C**) which can be used as independent counter channels or as two quadrature (phase encoder) inputs (one only phase encoder input on DT81). Channels 1C and 2C are low threshold capable.

Bidirectional Digital I/O Channels

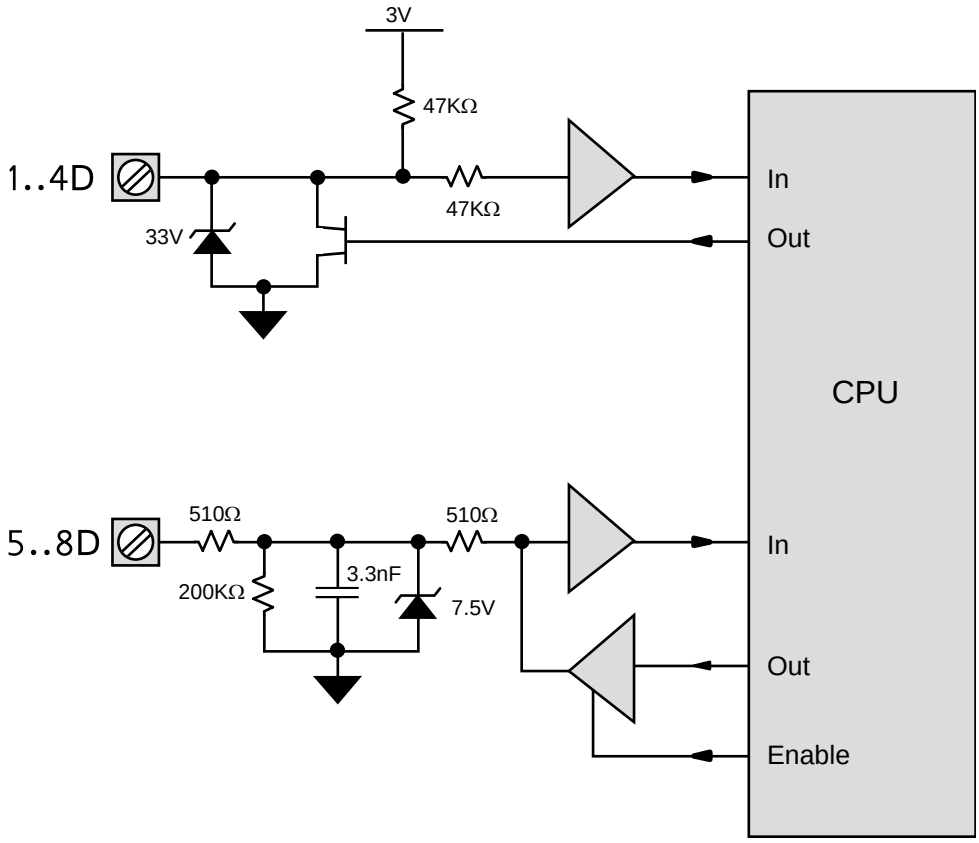


Figure 46 Digital Circuit (DT80 channel numbers shown)

Figure 46 shows a simplified circuit diagram for the DT80's eight digital I/O channels. As can be seen, the channels can be divided into two groups, **1D**-**4D** and **5D**-**8D** (**1D-3D** and **4D** for DT81). While these two groups have different hardware characteristics (discussed below), all eight channels are accessed and used in much the same way.

Each of the digital channels can be used as *either*:

- a digital input (for monitoring the state of a relay or logic signal), or
- a digital output (for driving a relay or other control device)

Warning Beware of conflicts when using the *DT80*'s bi-directional digital channels (**1D** to **8D**). For example, if a device such as a PLC is actively driving one of these channels and you program the *DT80* to also drive the same channel as an output (for example, **1DSO=0**), then a conflict exists. This has the potential to damage the digital channel or the driving source. We recommend placing a series resistor between the digital channel and the signal source to limit the current that can be driven into the channel. When choosing the resistor's value and power rating, be sure to consider the source's output voltage, drive current and operating frequency.

Using Digital Inputs

Channel Types

Digital inputs can be monitored using the following channel types. In each case *n* represents the channel number (1-8).

Type	Function
*n***DS**	**Digital State**: returns the state of digital input *n***D**; 0=low, 1=high
*n***DN**	**Digital Nybble**: returns the state of four consecutive digital inputs starting at *n***D** as a 4-bit number (0-15). For example, if channel **3DN** returns the value 13 (binary 1101) then this indicates that input **3D** is high, **4D** is low, **5D** is high and **6D** is high.
*n***DB**	**Digital Byte**: returns the state of all eight digital inputs as an 8-bit number (0-255). For this channel type, *n* must always be 1.
*n***C**	**Counter**: returns the number of positive-going edges seen on digital input *n***D**. Counter value is a signed 32-bit integer. These counters are low speed polled counters.

Channel Options

The following channel options are applicable to digital input channel types:

Type	Option	Description
DS	none	
DN, **DB**	(channel factor)	**Bitmask**: This specifies which input channels to read. For example 2DN(7) (bitmask = 0111 binary) will return the state of inputs 2D, 3D and 4D in bits 0 (lsb), 1 and 2 respectively. For channel 5D the mask bit is zero so it is not read and bit 3 of the returned value will always be zero (5D can then be used as an output if desired). The default values for **DN** and **DB** are 15 and 255 respectively (ie. read all bits)
C	(channel factor)	**Wrap Value**: Counter will reset to 0 (or "wrap around") when this value is reached. For example, if 8 pulses are received on input 4D then channel 4C(3) will count in the sequence 1, 2, 0, 1, 2, 0, 1, 2 so after 8 pulses the value 2 will be returned. Default value is 0 (do not reset)
C	**R**	**Reset**: counter is cleared to 0 after returning its current value.

Connecting to Digital Inputs

Warning The *DT80*'s digital inputs are NOT reverse-polarity-protected. Therefore ensure signal polarity is correct — positive to numbered terminals, negative to **DGND** terminals — before connecting signals to the *DT80*'s digital inputs.

Warning Do not apply more than 30Vdc to inputs **1D**-**4D** (**1D-3D** for DT81), and do not apply more than 20Vdc to inputs **5D**-**8D** (**4D** for DT81).

The two groups of digital input channels have different electrical characteristics. In particular:

- Inputs **1D-4D** (**1D-3D** for DT81) include a 47k pull-up resistor. The default state (if nothing is connected) is therefore HIGH. This in turn means that channels **1..4DS** will return 1 if the inputs are not connected.
- Inputs **5D-8D** (**4D** for DT81) include a 200k pull-down resistor. Their default state is therefore LOW (0). So if all 8 inputs are disconnected then **1DB** will return 15 (00001111).

Relay Inputs

Voltage-free relay contact closures can easily be detected on channels **1D-4D** (**1D-3D** for DT81) by wiring the relay contacts between the input pin and **DGND**. *n***DS**=0 indicates contacts closed, *n***DS**=1 indicates contacts open.

Channels **5D-8D** (**4D** for DT81) are less suitable for relay contact inputs, but they can still be used, for example if the contacts are wired between the input pin and an external 3-20V dc supply.

Logic Inputs

Actively driven logic signals can be directly connected to all input channels, subject to the input voltage level specifications detailed below:

Terminal	Limit	
1D-4D (**1D-3D** for DT81)	Maximum continuous terminal voltage	30Vdc
	Minimum continuous terminal voltage	-0.6Vdc
	Note Voltages outside this range can permanently damage the channel.	
	Minimum input high voltage (Polled input)	3.0V
	Maximum input low voltage (Polled input)	0.75V
5D-8D (**4D** for DT81)	Maximum continuous terminal voltage	20Vdc
	Minimum continuous terminal voltage	-0.6Vdc
	Note Voltages outside this range can permanently damage the channel.	

	Minimum input high voltage (Polled input)	3.0V
	Maximum input low voltage (Polled input)	0.75V

See *Wiring Configurations — Digital Channels* (P185) for sample digital input wiring configuration diagrams.

Other Considerations

Scan Rate

The digital input channels are scanned at 17ms intervals (60Hz). This means that:

- the minimum input pulse width is **17ms** – shorter pulses may not be recognised.
- the maximum input count frequency, assuming a 50% duty cycle, is **30Hz**.

Use the high-speed counter channels (P159) for higher count frequencies.

Sleep Mode

Digital inputs are *not* scanned while the *DT80* is asleep. Use the high-speed counter channels (P159) if you need the logger to continue to count pulses even while asleep.

Schedule Triggers

Digital input transitions can be used to trigger or enable a report schedule. (see *Trigger on External Event* (P46))for more details.

Counter channels can also be configured to trigger a schedule when the wrap value is reached.

Wake Terminal

A high to low digital input transition can be used to wake the *DT80* by connecting the digital input in parallel with the **WK** (wake) terminal. The *DT80* can then be programmed so that each time an external pulse occurs the *DT80* will wake, run an event triggered schedule (see *Trigger on External Event* (P46)), then go back to sleep.

Presetting Counters

The count value for a digital input channel can be preset using an expression, eg.

```
RA1M 8C=1000  RB2S 8C
```

If a 1Hz signal is now applied to input **8D** you would expect the values returned every 2s for channel **8C** to follow a sequence similar to:

1000, 1002, 1004 ... 1056, 1058, 1000, 1002 ...

Setting Counter Wrap Value

Note that a counter's wrap value (channel factor) is applied when the channel is defined (ie. when the job is entered), not when it is evaluated. Also, setting the wrap value has the side effect of resetting the count value to zero. This implies that:

- a particular counter's wrap value need only be specified *once* in the job. It does not need to be specified every time the counter is evaluated.
- If querying a counter using the immediate schedule (eg. by periodically typing "**1C**"), do not specify a wrap value each time. Each time you evaluate an immediate channel you are also defining it, so the counter value will always be returned as zero if you specify a wrap value each time.

Using Digital Outputs

Channel Types

Digital outputs can be used to control external devices using the following channel types. In each case *n* represents the channel number (1-8); *x* can be either a number, CV or expression.

Type	Function
*n*DSO=*x*	**Digital State Output**: sets the state of digital output *n***D**; 0=low, 1=high. For example 2DSO=0 sets output **2D** low.
*n*DNO=*x*	**Digital Nybble Output**: simultaneously sets the state of four consecutive digital outputs starting at *n***D**. For example, 5DN=5 (binary 0101) sets **5D** high, **6D** low, **7D** high and **8D** low.
*n*DBO=*x*	**Digital Byte Output**: simultaneously sets the state of all eight digital outputs as an 8-bit number (0-255). For this channel type, *n* must always be 1.
1RELAY=*x*	**Relay Output**: sets the state of the latching **RELAY** output: 0=open, 1=closed
1WARN=*x*	**LED output**: sets the state of the **Attn** LED: 0=off, 1=on

Channel Options

The following channel options are applicable to digital output channel types:

Type	Option	Description
DSO, RELAY, WARN	(channel factor)	**Delay (ms)**: The *DT80* waits for the specified number of milliseconds after setting the output state. Default is 0, ie. no delay. If the R option is specified then the default and minimum delay setting is 10ms.
DNO, DBO	(channel factor)	**Bitmask**: This specifies which output channels to set. For example 1DNO(14)=1CV*2 (bitmask = 1110 binary) will output bits 0 (lsb), 1 and 2 of 1CV on outputs **2D**, **3D** and **4D** respectively. For digital channel **1D** the mask bit is 0 so its state will not be affected by this command. The default values for DNO and DBO are 15 and 255 respectively (set all bits)
DSO, DNO, DBO, RELAY, WARN	R	**Reset**: After setting the output bit(s) to the specified state(s) and waiting for the delay time the output(s) will be set to the opposite state. In other words a pulse will be generated.

Digital Output Operation

All digital output channels are initialised to their default states on initial power-up, hard reset (SINGLEPUSH) or soft reset (RESET). Entering a new job does not initialise the digital outputs.

The default states are summarised below:

Channel	Default state	Comments
1..4DSO (DT80) 1..3DSO (DT81)	1	output pulled up (high), controlled load OFF
5..8DSO (DT80) 4DSO (DT81)	0	output driver disabled, pulled down (low)
1RELAY	0	contacts open
1WARN	0	LED off

A digital output command, eg. 1DSO(20,R)=1 is processed as follows:

1. First, the output (or outputs for DNO/DBO) is set to the specified state; if no state is specified then nothing is done.
2. Then the *DT80* waits for the specified delay, if any. If a state was specified and the R option was also specified then the default delay is 10ms, otherwise 0ms.
3. Then, if R is specified, the output(s) is/are inverted.
4. Finally, the output value as at Step 2 is returned.

The current state of any digital output is thus returned when a digital output command is evaluated. For example, typing 2DSO will return the state to which the output was last set. This will not necessarily reflect the actual state of the **2D** terminal (use 2DS to read the actual state). And if 2DSO(R) is entered then the state of **2D** will be inverted and the original state will be returned.

Connecting to Digital Outputs

As noted above, the two groups of digital channels have different electrical characteristics. In particular:

- Outputs **1D-4D** (**1D-3D** for DT81) use an open-drain FET output driver. This can sink up to 100mA @ 30Vdc so it can drive a low voltage actuator or relay or LED directly. See (wiring diags). A 47k pull-up resistor (to +3.3V) is also included, allowing logic devices to be driven.
- Outputs **5D-8D** (**4D** for DT81) are *not* suitable for directly driving loads such as relays or LEDs. Logic devices can however be driven. Note that each of these output drivers is tri-stateable.

When the open-drain outputs are used to directly drive loads, the load will be ON when the output is in the LOW state. Thus if a load was wired up to output **1D** you would use **1DSO=0** to turn the load ON and **1DSO=1** to turn it OFF.

The default state of the output drivers on second group of channels is *disabled* (tri-stated). This allows these channels to be used as inputs.

Important Although the digital state outputs incorporate transient protection for inductive loads, we recommend that you place a reversed diode across such loads. The output drivers are not current-limited, so avoid shorting a supply line directly to a digital state output.

Output Driver

When a digital output command for channels **5D-8D** (**4D** for DT81) is evaluated, the output state is set to the required value, then the output driver is enabled. The output will then stay enabled until an input command (eg. **5DS**) is evaluated for that channel.

For example,

```
6DSO=0 DELAY=10 6DSO(R,5)=1 DELAY=10 6DS(W)
```

will drive logic 0 on output **6D** for 10ms, then logic 1 for 5ms, then logic 0 for 10ms, then the output driver will be disabled.

Relay Output

The **RELAY** terminals are voltage-free, normally-open, latching relay contacts. These are rated at 1A @ 30Vdc. Use **1RELAY=1** to close the contacts, **1RELAY=0** to open.

See *Wiring Configurations — Digital Channels* (P185) for sample digital output wiring configuration diagrams.

Other Considerations

Sleep Mode

The states of all digital outputs are maintained while the *DT80* is asleep. Note also that the **RELAY** output uses a latching relay, so no extra current is required to hold it in the closed state.

Alarm Digital Actions

One or two digital outputs can be configured to follow the state of an alarm. That is, when the alarm is inactive the output(s) are in their default state (1 for **1..4DSO**, 0 for **5..8DSO**, **1RELAY** and **1WARN**) and when the alarm is active the output(s) will be in their non-default state. *Alarm Digital Action Channels* (P73) for more information.

Delay Accuracy

The actual pulse width generated by the Delay option for **DSO** will not necessarily be exactly as specified. For delays of 20ms or less it will be close (within 1ms). For longer delays the resolution is +/- 16ms however it is guaranteed that the duration will be at least the specified time.

Note also that, like the **DELAY=** channel type (*Table 1: DT80 Channel Types* (P32)), high values for the DSO Delay option are not recommended as they can prevent the timely evaluation of other schedules.

Attention LED Usage

The **Attn** LED may also be flashed by the *DT80* to indicate an internal fault or warning condition (*Attn Indicator* (P97)). This will override the state set using the **1WARN** output channel.

SDI-12 Channels

About SDI-12

SDI-12 is a serial communications protocol for interfacing multiple microprocessor based sensors to a data logger. SDI-12 uses a shared three-wire "bus" – 12V power, data (0-5V signaling levels) and ground – and operates at a data rate of 1200 baud.

Each sensor connected to an SDI-12 bus is configured with a unique **address**, which is usually just a single digit 0-9. The data logger specifies this address when it requests data from the sensor. This transmission will be received by all sensors, but only the one with the matching address will respond. If two sensors have the same address then they will both try to transmit at once, resulting in garbled communications.

The SDI-12 standard has undergone a number of revisions; at the time of writing the current version is 1.3. (1.0 was released in 1988, 1.2 in 1996 and 1.3 in 2000.) Not all sensors support the latest version. The *DT80* can determine which version of the standard a given sensor supports, and act appropriately.

Each SDI-12 message sent by the data logger is a short (up to 5 characters) plain ASCII string, terminated by a **!** character. The response from the addressed sensor is also in ASCII format (up to 80 characters).

For more details see http://www.sdi-12.org.

Connecting to SDI-12 Devices

The *DT80*'s tri-stateable digital I/O channels (**5D-8D** on DT80, **4D** on DT81) can be used to control up to four SDI-12 buses. Up to ten SDI-12 sensors can be connected to each bus.

As shown in *Figure 68* *(P187)*, an SDI-12 bus is connected to the *DT80* as follows:

- The SDI-12 **DATA** line connects to one of the digital I/O terminals **5D – 8D** (**4D** for DT81)
- The SDI-12 **GROUND** line connects to the **D GND** terminal
- The SDI-12 **POWER** line is typically connected to the *DT80*'s external power input terminal (**+**), or alternatively a separate 9.6-16V DC supply can be used.

When connecting a sensor to the *DT80* for the first time, it's best to connect only that sensor, ie. you should temporarily disconnect any other sensors on the same SDI-12 bus. This ensures there will be no address conflicts

Testing and Configuring an SDI-12 Device

SDI-12 Address

The first task is to determine the address of the sensor. All SDI-12 sensors are able to be set to one of at least ten different addresses. Depending on the sensor, this may be done by:

- changing a hardware setting, eg. DIP switches
- sending an SDI-12 "change address" command (*a***A***b***!**, where *a* is the current address and *b* is the new address)
- connecting the sensor to a PC serial port and using configuration software supplied by the sensor manufacturer. You may also need to use this configuration software to configure other aspects of the device – for example the device's SDI-12 interface may be disabled by default, so you would need to enable it using the configuration software.

Consult the sensor's documentation to determine how to set its address. Note that all SDI-12 sensors are factory set to address 0. If you are only connecting one sensor to the SDI-12 bus then you can leave it set to this value.

Using SDI12SEND

The *DT80*'s **SDI12SEND** command allows you to manually send SDI-12 commands to the sensor for testing and configuration purposes. The format of this command is as follows:

SDI12SEND *channel* **"***string***"**

where:

- *channel* is the digital I/O channel (**5** – **8**) (**4** for DT81)
- *string* is a valid SDI-12 command string to send to the device. All commands start with the sensor address (**0** – **9**) and end with a **!** character.

If there is a reply from the device then it will be displayed, assuming the **/M** (enable messages) and **/h** (free format) switches are set.

For example, the *a***I!** command (*a* = address) should result in the sensor returning an identification string, eg.

```
SDI12SEND 5 "0I!"
5SDI12: 0I!012SENTEK  XEPI  1165FA14F000800
```

In this example a sensor with address **0** is connected to digital channel **5D**. The output of the **SDI12SEND** command shows the complete transaction: the first few characters (up to the `!`) are the command string that was sent, the rest are the response from the sensor. In this case, the response indicates:

- `0` – the sensor's address
- `12` – the version of SDI-12 supported by the sensor (1.2)
- `SENTEK` – the sensor manufacturer
- `XEPI` – the model name
- `116` – the sensor firmware version
- `5FA14F000800` – other sensor details, eg. serial number

If a valid response is not received, an error message will be displayed, eg:

```
SDI12SEND 5 "3I!"
5SDI12: 3I! *no response
```

In this case the command was sent to the address 3, which is the wrong address. This error may also indicate a wiring problem, or perhaps the SDI-12 interface on the device has not been enabled.

Errors such as "`*framing error`", possibly in conjunction with a garbled looking message, generally indicate an address conflict (more than one device with the same address is connected) although it may also indicate an electrical noise issue.

Reading Data from SDI-12 Devices

Measurement Modes

SDI-12 sensors can operate in one of two different modes:

- **Measure on demand** is the traditional SDI-12 method, which all sensors support. The sensor is idle (typically in a low power mode) until it is woken by the data logger sending it a measurement request. The sensor then takes the measurement and then, possibly several seconds later, returns the data.
- **Continuous measurement** is an alternative method, supported by some SDI-12 sensors. The sensor takes measurements at regular intervals, then when the data logger requests data it immediately replies with the last reading it took.

In Measure On Demand mode the *DT80* must send a request then wait until the measurement is ready, which may be immediate or it may take several seconds – depending on the sensor. During this time no other schedules will run, and no commands will be executed (similar to a **DELAY** channel).

Measure On Demand mode most suitable in applications where the sensor is being polled infrequently. This mode minimises system power usage because the sensor only takes a measurement when it is requested to.

Continuous measurement mode is suited to applications where the logger needs to poll the sensor relatively often, or it is running other schedules which should not be delayed. Note that the sensor may need to be sent some special commands (either via SDI-12 or via a separate RS232 configuration interface) to enable continuous mode, set the measurement rate and so on.

Check the sensor's documentation to see whether continuous measurement mode is supported. If it is, your first decision is whether you want to use it.

Registers

Most SDI-12 devices can measure a number of different quantities. For example, a device might have 4 temperature sensors plus 8 moisture sensors, ie. it can measure 12 distinct quantities.

In *DT80* parlance, each individual data item (quantity that can be measured) is termed a **register**. An SDI-12 device may then divide its particular set of registers into a number of groups, or **register sets**. For example the abovementioned device might define register set #2 as the 4 temperature sensors, and register set #3 as the 8 moisture sensors.

The measurement process then proceeds as follows:

1. The *DT80* sends a request message (*a***C***r***!**) to the SDI-12 device, specifying the register set (*r*) of interest – only one register set can be requested at a time. (If *r* is 0 then it is omitted, ie. *a***C!** is sent.)
2. The sensor will now measure and update all of the registers in the specified set.
3. After the required time interval, the *DT80* sends a second message (*a***D0!**) to request the actual data values.
4. The sensor replies immediately, sending some or all of the register values.
5. If not all register values were sent, the *DT80* may send further *a***D***n***!** message(s) to request the remainder.

In Continuous Measurement mode the process is considerably simpler:

1. The *DT80* sends a message (*a***R***r***!**) to request the most recent data values for register set *r*.
2. The sensor replies immediately, sending all of the register values.

The SDI12 Channel Type

The *DT80*'s **SDI12 channel type** allows you to read a data value from an SDI-12 device in much the same way as you would read a voltage using the **V** channel type – without worrying about the technicalities of the SDI-12 protocol. There are four of these channels available, **5..8SDI12**, corresponding to the four SDI-12 compatible digital I/O channels (**5D – 8D**).

When an **SDI12** channel is used, you need to specify additional information via channel options. The following channel options apply:

Option	Function
AD*a*	**Address**: specifies the SDI-12 address of the sensor to read (**0**-**9**). If not specified, **0** is assumed
R*nnn*	**Register**: specifies the particular register to return. If the sensor defines more than one register set then the hundreds digit specifies the register set. If the hundreds digit is not specified then the sensor's default register set (register set #0) is assumed. The tens and units digits specify the register number within the register set: **1** for the first data item in the set, **2** for the second, and so on. If this option is not specified, **1** is assumed (which would be suitable for a sensor that only returned one value)
CM	**Continuous Measurement**: If this option is present then the *DT80* will use Continuous Measurement Mode. The sensor is assumed to have been configured to take continuous measurements.

R*nnn* Settings

Some sample settings for the **R***nnn* channel option are shown below:

Option	Data Value (Register) to Read
(none)	First data value in default register set (register set #0)
R001 (or **R1**)	First data value in default register set (register set #0)
R019 (or **R19**)	19th data value in default register set (register set #0)
R100	not valid

R101	First data value in register set #1
R344	44th data value in register set #3

For example, suppose a particular SDI-12 device measures 9 different quantities in one go (ie. it has 9 registers in its default register set). These 9 values would then be returned to the data logger using an exchange of messages similar to the following (underlined text is sent by the data logger, the remainder is returned by the sensor):

```
0D0!0+005.7541+068.0368+017.6721+054.3521+052.0475+016.2069+017.1182+016.8696
0D1!0+019.1727
```

These 9 register values can then be accessed using **5SDI12(R1)** through **5SDI12(R9)**. So if you were interested in the third value in the list you would use:

```
5SDI12(R3)
5SDI12 17.7
```

As can be seen, the third value returned by the sensor (+017.6721) is the return value of the channel.

Example

Measure on Demand

In this example the documentation for a hypothetical SDI-12 weather station states: "Send the *a***C1!** (or *a***M1!**) command to measure (1) internal temperature (degC), (2) external temperature, (3) humidity (%RH) and (4) pressure (hPa). Send the *a***C2!** (or *a***M2!**) command to measure (1) wind speed (km/h), (2) max gust and (3) direction (degrees)." The device is connected to the *DT80* using digital I/O **7D**, and has been configured with an SDI-12 address of **3**.

In this case the device has two register sets (#1 and #2), one with four registers (measured quantities), one with three. The following *DT80* job will read and log external temperature, pressure and wind speed every two minutes:

```
BEGIN"CLOUDY"
RA2M 7SDI12(AD3,R102,"Ext temp~degC")
     7SDI12(AD3,R104,"Pressure~hPa")
     7SDI12(AD3,R201,"Wind speed~km/h")
LOGON
END
```

Note If your sensor supports both the *a***M***n***!** and the newer *a***C***n***!** SDI-12 commands (most modern sensors will) then be sure to refer to the section on the *a***C***n***!** command in the sensor documentation when determining which register numbers to use. These two SDI-12 commands do much the same thing but the ordering of the returned data values may be different. The *DT80* always uses the *a***C***n***!** command in preference to *a***M***n***!**.

Continuous Measurements

The weather station documentation goes on to say "To enable continuous measurement mode (sampling every *t* seconds), use the *a***XC=***t***!** command; to disable use *a***XCD!**. [SDI-12 "**X**" commands are often used to implement device specific functions such as this.] Use *a***R1!** and *a***R2!** to return the most recent values of int temp/ext temp/RH/pressure and wind speed/gust/direction respectively."

The following job does the same thing as the previous example, but this time continuous measurement mode is used:

```
BEGIN"CLOUDY_CM"
SDI12SEND 7 "3XC=10!"  ' enable continuous mode
RA2M 7SDI12(AD3,R102,CM,"Ext temp~degC")
     7SDI12(AD3,R104,CM,"Pressure~hPa")
     7SDI12(AD3,R201,CM,"Wind speed~km/h")
LOGON
END
```

Other Considerations

Execution Time

In Measure on Demand mode, **SDI12** channels may take a significant amount of time to execute – often 10 seconds or more, depending on the sensor. During this time no other schedules or commands are executed.

Note however that the *DT80* will only request a measurement of a given register set once per schedule. So the following schedule:

```
RA1M 5SDI12(R1) 5SDI12(R3) 5SDI12(R201) 5SDI12(R4)
```

would execute as follows:

1. *DT80* requests a measurement of register set #0 (**0C!**), then waits until it is ready.
2. *DT80* reads values for registers 1, 3 and 4, which are all part of register set #0. It will probably be given values for other registers (eg. register 2), which it will discard because they are not referenced in the job
3. *DT80* can now evaluate (ie. return/log values for) the first two channels.
4. *DT80* requests a measurement of register set #2 (**0C2!**), then waits until it is ready.
5. *DT80* reads value for register 1 (in register set #2) and discards any other values that it receives.

6. *DT80* can now evaluate the last two channels.

Notice that the *DT80* waits for the sensor on two occasions, once for each register set.

Versions

The *DT80* automatically determines the version of the SDI-12 specification that a given sensor supports, and tailors the types of messages it sends accordingly. For example:

- Error check codes (CRCs) are used on data messages, but only if the sensor supports SDI-12 Version 1.3 or later.
- Continuous Measurement mode is only available if the sensor supports SDI-12 Version 1.2 or later.

Troubleshooting

There are two main areas where difficulties may arise when setting up an SDI-12 system

- the *DT80* cannot communicate properly with the sensor
- the sensor does not support the request you are making of it

These will be discussed in the sections below.

Diagnostic Messages

When troubleshooting an SDI-12 connection, it can often be helpful to see the actual SDI-12 messages. The *DT80* provides a special parameter setting for this purpose:

```
P56=2
```

If this parameter setting were used with the weather station example described above (Measure on Demand mode), you might see something like:

```
7SDI12: [8] 3C1!300704
7SDI12: [25] 3D0!3+22.91+42.40+21.0+1013.9
Ext temp 42.4 degC
Pressure 1013.9 hPa
7SDI12: [8] 3C2!300603
7SDI12: [18] 3D0!3+4.29+31.43+012
Wind speed 4.3 km/h
```

which shows the measurement request message (`3C1!` or `3C2!`) and response, followed by the data retrieval message (`3D0!`) and response, for each register set.

Set **`P56=0`** to turn off these messages.

Communications Problems

If the sensor does not reply at all to a request, the *DT80* will output an error message, eg:

```
8SDI12(R3)
dataTaker 80 E80 - Serial device not responding (8SDI12:AD0:R3)
8SDI12 -9000000000.0
```

Note also that the value returned by the channel is the special "NotYetSet" error value (-9.0e9, see *Data Errors* *(P201)*)

The main things to check here are:

- cabling (Is the sensor powered?)
- correct **`SDI12`** channel number (In the above example the SDI-12 data wire should be connected to digital input **8D**.)
- correct SDI-12 address (In the above example the device should have been configured for address 0.)

This error message may also indicate an **address conflict** – a response was received from the sensor but it was garbled because two or more sensors tried to both transmit at the same time, which will occur if they are both configured to use the same address.

Try connecting only one sensor at a time and verifying the address of each sensor. For most sensors you can use the following command:

```
SDI12SEND 8 "?!"
8SDI12: ?!1
```

In this case the sensor has responded, stating that it has been set to address `1`.

Communications may also be affected by electrical noise or poor cable connections. If the sensor supports it, the *DT80* will request that it include an error checking code (CRC) with each data record, which the *DT80* will then check. Any corruption of these messages will then result in an error message such as:

```
dataTaker 80 E81 - Serial device invalid response (8SDI12:AD0:R1)
8SDI12 -9000000000.0
```

Unsupported Functions

The other error message that you may see is:

```
5SDI12(R207)
dataTaker 80 E82 - Serial device data not available (5SDI12:AD0:R207)
5SDI12 -9000000000.0
```

In this case the sensor has indicated that the requested register does not exist. The sensor either does not support register

set #2 (ie. the *a***C2!** command), or that register set returns fewer than 7 values.

This error may also occur if you have requested continuous mode operation (using the **CM** channel option) but the sensor does not support continuous mode, or continuous mode has not been enabled on the sensor.

Double check the sensor documentation. It may help to turn on the diagnostic messages, eg:

```
P56=2 5SDI12(R207)
5SDI12: [8] 0C2!000000
dataTaker 80 E82 - Serial device data not available (5SDI12:AD0:R207)
5SDI12 -9000000000.0
```

In this case the sensor has returned `00000` in response to the *DT80*'s request, indicating that no data values are available in register set #2.

High Speed Counter Channels

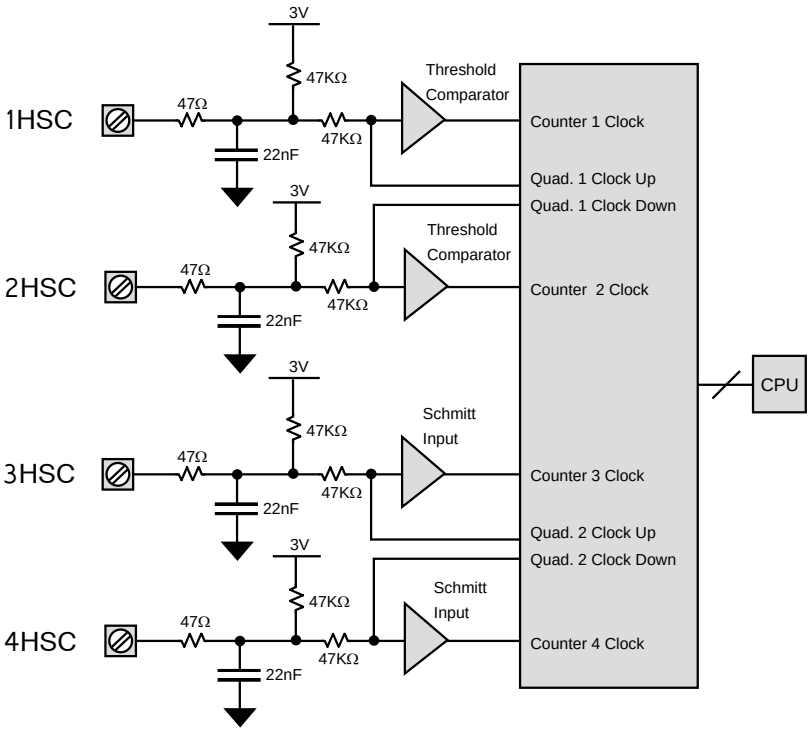


Figure 47 High Speed Counter Channels

Figure 47 shows a simplified circuit diagram for the *DT80*'s four hardware counter inputs. As can be seen, the channels can be divided into two pairs of inputs, **1C-2C** and **3C-4C**. Each pair can be used as either:

- two independent counter inputs, for pulse counting, or
- a single phase encoder (quadrature) input, for use with position sensors that provide phase encoded outputs ("A" and "B")

Note that on the DT81, only the **3C-4C** inputs can be used with a phase encoder.

Using Counter Inputs

Channel Types

Counter inputs can be monitored using the following channel types. In each case *n* represents the channel number (1-4 for **HSC**, 1-2 for **PE**). On the DT81 there is only one phase encoder channel, **1PE**.

Option	Function
*n***HSC**	**High Speed Counter**: returns the number of positive transitions seen on counter input *n***C**. The counter value is a signed 32-bit integer.
*n***PE**	**Phase Encoder**: returns the current relative position of the phase encoder device connected to input pair *n***PE**. The value returned is in counts and is a signed 32-bit integer, which may be positive or negative depending on the direction of travel.

Channel Options

The following channel options are applicable to high speed counter input channel types:

Type	Option	Description
HSC, **PE**	(channel factor)	**Wrap Value**: Counter will reset to 0 when this value is reached. For example, if 8 pulses are received on input **3C** then channel **4HSC(3)** will count in the sequence 1, 2, 0, 1, 2, 0, 1, 2 so after 8 pulses the value 2 will be returned. Default value is 0 (do not reset)
HSC, **PE**	**R**	**Reset**: counter is cleared to 0 after returning its current value.
1..2HSC, **1PE**	**LT**	**Low Threshold**: select low-level input thresholds (low threshold 2mV and high threshold 7mV). Not applicable to inputs **3C**-**4C**.

Connecting to Counter Inputs

Warning The *DT80*'s counter inputs are NOT reverse-polarity-protected. Therefore ensure signal polarity is correct — positive to numbered terminals, negative to **DGND** terminals — before connecting signals to the *DT80*'s counter inputs.

Warning Do not apply more than 30V to inputs **1C**-**4C**.

Counter input channels **1C-2C** and **3C-4C** have different electrical characteristics. In particular:

- Inputs **1C-2C** include selectable TTL or low-level input thresholds. Low thresholds (selected by using the **LT** channel option) allow direct connection to sensors whose output is only a few mV, eg. inductive-pickup flow sensors.
- Inputs **3C-4C** use a standard TTL level Schmitt trigger input.

Voltage-free relay or switch contact closures can be counted on channels **1C-4C** by wiring the relay contacts between the input terminal and **DGND**.

All inputs include low-pass filtering to assist in "debouncing" mechanical switch or relay inputs. For voltage-free contact inputs this limits the maximum count rate to approximately 500Hz. For actively driven inputs, however, the maximum count rate is approximately 100kHz.

Phase Encoders

A phase encoder is a device for measuring relative angular or linear position. As it moves, it outputs two streams of pulses ("A" and "B") whose phase relationship (A leading or B leading) indicates the direction of travel.

The *DT80*'s **PE** channel type decodes these pulses and returns a signed position value in counts.

Note that the "mode" of a counter channel pair (ie. whether it operates as two counters or a single phase encoder channel) is set when the channel is *defined* (ie. when the job is entered), not when it is *evaluated.* This implies that a particular counter input pair *cannot* be read as a phase encoder value at one point in a job, and as a pair of counters at another. In other words, if your job defines a channel **1PE** then it should *not* also define channels **1HSC** or **2HSC**, and vice versa.

Other Considerations

The high-speed counter inputs continue to function while the *DT80* is asleep.

However, it is important to note that each hardware counter is 16 bits wide. (Count values are maintained and returned as 32-bit values, but the physical hardware counters attached to inputs **1C**-**4C** are 16-bit.) If more than 65536 pulses occur while the *DT80* is sleeping then the hardware counter will overflow, and this will cause an inaccurate count value to be returned when the *DT80* wakes.

It is therefore necessary to ensure that the *DT80* is programmed to wake often enough to ensure that the hardware counters can be read before they overflow.

For example, if the average counter input frequency is 100Hz then the *DT80* must be programmed to wake at least every 65536/100 seconds (about every 10 minutes). This can be done by including a 10-minute schedule (eg **RA10M**) in the job.

Most of the other comments made above regarding digital input counter channels apply equally to the high speed counter channels. For example, HSC channels can be preset to a particular starting value (eg **2HSC=1CV*10**), HSC channels can trigger a schedule when their wrap value is exceeded, and so on.

Examples

Pulse Train Output

The schedule command

```
RA2S 6DS0(500,R)=1
```

produces a pulse train from channel **6D** which is HIGH for 0.5s and LOW for 1.5s.

Sensor Power Control

In the schedule command

```
RA20M D T 4DSO(1000)=0 1..4V 4DSO=1
```

digital state output 4 controls a relay that switches the power supply to a group of sensors. Every 20 minutes the sensors are powered up, the system waits one second while the sensors settle, the sensors are scanned, and the sensor power supply is turned off again.

Manual Control

The polled schedule (see *Trigger on Schedule-Specific Poll Command* (P47)) can also be used to switch digital state output channels. For example, the command

```
RBX 3DSO(5500,R)=0
```

turns a load connected to channel **3D** ON for 5.5 seconds when an **XB** poll command is received.

Frequency Measurement

The **R** channel option can be used to measure the frequency of an input signal, eg.

```
RA1S 1HSC(R,RS)
```

will return the frequency in Hz of an input signal on channel **1C**, while

```
RA10S 1HSC(R,RS)
```

will do the same thing but resolve down to 0.1Hz.

This technique can also be used for the digital input channels (**1D-8D**), eg.

```
RA1S 7C(R,RS)
```

will return the frequency in Hz of an input signal on channel **7D**, in the range 1-30Hz.

Serial Channel

(Not applicable to DT81)

The *DT80*'s Serial Channel (see *Figure 48* (P167)) can be used to connect to serial input and/or output devices such as a serial sensor, GPS terminal, printer, barcode reader, display panel, PLC, or even to another *dataTaker*.

The Serial Channel

- can transmit programmable 'prompt' or 'poll' messages to serial devices and interpret their replies
- can respond to asynchronous incoming serial messages. Incoming data can wake the logger from sleep mode.
- can be configured for either the RS-232, RS-422 or RS-485 comms standard (RS-232 supports a single point-to-point connection; the other standards support multiple devices in a multi-drop configuration)
- has a differential transmitter and receiver that provide for the different serial standards
- has RTS/CTS handshake lines (RS232 only)
- supports baud rates of 50 to 115200 baud

Note Serial **SDI-12** based sensors do not use the serial channel. SDI-12 sensors should be connected to one of the digital I/O pins **5D** - **8D**. See *SDI-12* Channels (P159).

Connecting to the Serial Channel

The *DT80* serial channel terminals have different functions depending upon the configured serial standard (RS232, RS422 or RS485).

Terminal	RS232	RS422	RS485	Wake
Tx **Z**	Transmit Data	Transmit Data– (A)	Data– (A)	
Rx **A**	Receive Data	Receive Data+ (B)		☑
RTS **Y**	Handshake output	Transmit Data+ (B)	Data+ (B)	
CTS **B**	Handshake input	Receive Data– (A)		☑
D **GND**	Signal Ground	Ground	Ground	

Figure 48: The DT80's Serial Channel terminals (DTE)

Note that:

- The RTS and CTS handshake/control signals are available for RS232 only
- The DGND terminal is the signal return (common) for RS232. RS422/485 use differential signalling – the ground is only used for connection to the cable shield.
- Activity on either of the indicated terminals (ie. **Rx/A** and **CTS/B**) will wake the logger from sleep mode, although the data in the particular message that woke the logger will be lost. Note also that if the Wake feature is required and RS485 is being used then it will be necessary to link the Data terminals (**Tx/Z** and **RTS/Y**) to the wake-enabled terminals (**Rx/A** and **CTS/B**).

See *Serial Channels* (P187) for typical wiring diagrams.

Setting Serial Channel Parameters

The Serial Channel communications parameters are set by the command

PS=*type*,*baud*,*parity*,*databits*,*stopbits*

where:

Parameter	Settings	Default
type	specifies the signal standard: **RS232**, **RS422** or **RS485**	**RS232**
baud	is the baud rate at which you want the Serial Channel to operate. Use **50**, **75**, **110**, **150**, **300**, **600**, **1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600** or **115200**.	**1200**
parity	can be **N** (none), **O** (odd) or **E** (even)	**N**
databits	can be **7** or **8**	**8**
stopbits	can be **1** or **2**	**1**

These parameters may be specified in any order and all are optional.

For example, the command

PS=RS485,9600

sets the Serial Channel to RS485 mode, 9600 baud, no parity, 8 data bits, 1 stop bit.

These settings will be reset to their defaults by a hard reset (eg. **SINGLEPUSH**).

Serial Channel Commands

SERIAL Channel Type

Data flow into and out of the Serial Channel is controlled by the Serial Channel commands. These commands provide for

- formatting and management of output strings and prompts to be sent to the connected serial device.
- interpretation and parsing of input strings received from the connected device into *dataTaker* variables
- general management of the Serial Channel

The general form of a Serial Channel command is:

*n***SERIAL("***control_string***",***options***)**

where:

- *n* is the serial channel number. For a DT80, this is always **1**.
- *control_string* is a string of commands that specify the required output and input actions of the Serial Channel. See ##
- *options* are any other channel options that may be required

Note that **SERIAL** is actually a channel type, in the same way that **V** (voltage) is a channel type. It can appear in schedules and it has channel options, like any other channel type. The *control_string* is a special channel option which applies only to the **SERIAL** channel type.

Channel Options

Most of the standard channel options (*Table 3: DT80 Channel Options* (P41)) may be used with the serial channel, eg. **W** (working channel), **=***n***CV** (assign to CV), and so on.

For the **SERIAL** channel type, the **channel factor** is the maximum time to wait for serial data to be received. Default is 10s. This value is a floating point number, so a value of 0.1 will set the timeout to 100ms.

If the standard **"***UserName~UserUnits***"** channel option is specified, it must come after the control string in the list of serial channel options.

Channel Return Value

Depending on the control string, the **return value** of a **SERIAL** channel may be either:

- a data value, interpreted from the data returned by the sensor, or
- a status code, indicating whether the commands in the control string were performed successfully.

See *Return Value* (P172) for more details.

Serial Channel Operation

The Control String

The **"***control_string***"** is always enclosed by quotation marks. It can be broken into two parts:

- **Output actions** — commands, prompts or text strings that are to be sent from the *DT80* to the device connected to the serial channel. The various output actions available are detailed in the section. All output actions are enclosed by **{}**.
- **Input actions** — commands to manage the *DT80*'s Serial Channel and to interpret the information coming back from the serial device into the Serial Channel. The various input actions available are detailed in the section *Control String – Input Actions* (P171). Input actions are not enclosed by **{}**.

The general form of the **"***control_string***"** is

- any combination of output actions enclosed by **{}**, and/or
- any combination of input actions.

There may be any number of blocks of output actions and input actions, as shown in the following example Serial Channel commands:

```
1SERIAL("{output actions}",options)
1SERIAL("input actions",options)
1SERIAL("{output actions}input actions",options)
1SERIAL("{output}input{output}input",options)
```

The **"***control_string***"** is always executed in order left to right, giving you complete control over the sequence of actions.

Where a bi-directional dialog occurs between the *DT80* and serial device, the output actions and input actions can be included in the same Serial Channel command as shown above, or in separate Serial Channel commands as follows:

```
BEGIN
 RA1M
  1SERIAL("{output actions}",options)
  1SERIAL("input actions",options)
END
```

This latter approach simplifies the appearance of the program steps for supervising the Serial Channel, particularly if there are a number of data points to be prompted and interpreted or parsed in each access. Note however that each instance of **SERIAL** uses up one channel table entry (see *channel table* (P203))

Serial Data Transmission and Reception

If a job contains one or more **SERIAL** channel definitions then the serial channel is activated. Data may then be received from a connected serial device at any time whilst the job is loaded. As data is received, it is stored in an area of memory called the serial channel **receive buffer**.

When a **SERIAL** channel is evaluated (ie. when the schedule of which it is part executes), the *DT80* processes the control string from left to right. Output actions involve data being sent from the *DT80*, so they are performed there and then, as they are encountered in the control string.

When the *DT80* finds an input action in the control string it will read any previously received data from the receive buffer and attempt to match it against the format specified in the input action. If no data is present in the receive buffer at the time that the input action is processed then the *DT80* will wait up to 10 seconds (this timeout is configurable) for more data to arrive.

Then:

- If the incoming data matches that required by the input action then the *DT80* will move on to the next input action in the string. If the end of the control string is reached then the **SERIAL** channel will return and set its status code to **0** (success).
- If the timeout expires while the *DT80* is waiting for more data then evaluation of the **SERIAL** channel will be terminated and its status code set to **20** (receive timeout).
- If the timeout expires while the *DT80* is waiting for a particular CTS state (ie. **\c0** or **\c1** input action *(P169)*) then evaluation of the **SERIAL** channel will be terminated and its status code set to **5** (CTS timeout).
- If data is received which violates the input action specification then evaluation of the **SERIAL** channel will be terminated and its status code set to **29** (Scan Error).

Once the **SERIAL** channel has completed and set either a success or failure status code, the *DT80* will then move on to evaluating the next channel (if any) in the schedule.

Depending on how the input actions are specified, the return value of the channel may be either the status code or a scanned data value. See *Return Value* *(P172)*.

The following sections describe in detail the various output and input actions that can be specified in a control string.

Control String – Output Actions

The table below lists the ways in which prompts and text strings can be sent from the *DT80* to the device connected to the Serial Channel. These commands must be enclosed by **{}** in the control string.

What to output	Output Action syntax	Description
Text	*text* eg. **abc\009def\013**, **GETVAL^M^J**	A sequence of characters to be **sent**. Non-printable characters may be specified using *****nnn (where *nnn* is the ASCII code, 1-255). **^***char* notation may also be used for control characters (ASCII 1-31), see *ASCII-Decimal* Tables *(P192)*.
Break character	**\b[***n***]** or **\b[***n***CV]**	Transmit a "break" (set the Tx line to logic-0 state) for *n* or *n*CV character periods
Control signal	**\r1**	Set **RTS** (to a value >+3.5V) – RS232 only
	\r0	Clear **RTS** (to a value <–3.5V) – RS232 only
(Wait)	**\w[***n***]** or **\w[***n***CV]**	Delay for *n* or *n*CV milliseconds. Actual delay time will be approximately 2ms or 2 character times, whichever is longer.
reserved characters	**\%** or **\{** or **\}**	Output a **%, {** or **}** character.
CV value	**%***{flag}{width}.{precision}***type[***n***CV]** eg. **%d[2CV]** or **%9.3f[7CV]** or **%06d[1CV]**	Output the value of *n*CV in the specified numeric format (see below). Note that *{ }* signifies "optional"
string value	**%***{flag}{width}.{precision}***s[***n***$]** eg. **%s[1$]** or **%-9.9s[2$]**	Output the value of string variable *n*$

Numeric Formats

This table describes the possible values for *type* – that is, the different ways in which a CV value can be converted into a string of characters.

Type	Description	Example, assumes 1CV = 74.36
f	floating point	**1SERIAL("{%f[1CV]}")** → 74.36
e	floating point, exponential format	**1SERIAL("{%e[1CV]}")** → 7.436e01
E	floating point, exponential format	**1SERIAL("{%E[1CV]}")** → 7.436E01
g	**f** or **e** format depending on value	**1SERIAL("{%g[1CV]}")** → 74.36
G	**f** or **E** format depending on value	**1SERIAL("{%G[1CV]}")** → 74.36
d	integer	**1SERIAL("{%d[1CV]}")** → 74
x	hexadecimal integer	**1SERIAL("{%x[1CV]}")** → 4a
X	hexadecimal integer	**1SERIAL("{%X[1CV]}")** → 4A
o	octal integer	**1SERIAL("{%o[1CV]}")** → 112
c	single character	**1SERIAL("{%c[1CV]}")** → J

Note that

- The **%c** conversion outputs the value of *n*CV as a single 8-bit character. Only the lower 8 bits of the integer portion of *n*CV are output. So in the above example the character value 74 (ASCII "J") will be sent.

- The **%g** and **%G** conversions select exponential notation if the exponent is less than –4, or greater than or equal to the specified

Width, Precision and Flag

The various conversion types described above can be further qualified using the optional *width*, *precision* and *flag* specifiers. These allow you to control exactly how the transmitted data will be formatted.

Field Width

The *width* value specifies the **minimum output field width –** that is, the minimum number of characters that will be output. If the converted value requires fewer characters than the specified field width, then space or zero characters are used to pad the field to the specified width. If the converted value results in *more* characters than the specified field width, then all characters will still be output. The *width* parameter is not applicable for the **%c** conversion type.

The *precision* value means different things depending on the conversion type:

Type	*precision* term specifies:	Default
d, **x**, **X**, **o** (integer)	minimum number of digits to print (leading zeroes will be added if necessary)	no minimum
e, **E**, **f** (floating point)	number of digits to the right of decimal point	6 digits
g, **G** (mixed)	number of **significant digits** shown	6 digits
c (single character)	not applicable	
s (string)	**maximum** number of characters from the string to print	no maximum

Variable Width & Precision

The *width* and *precision* values are normally specified as numeric constants (eg. **%9.2f**), but they can also be specified as an asterisk (*****), in which case the value of a CV is used instead.

Output Action syntax and example	Description
%{*flag*}***.**{*precision*}***type*****[***n*CV,*w*CV**]** eg. **%*d[1CV,4CV]** or **%-*.2f[1CV,3CV]**	output the value of *n*CV in the specified numeric format, with the *width* parameter set to the value of *w*CV
%{*flag*}***.*[***n*CV,*w*CV,*p*CV**]** eg. **%*.*g[1CV,4CV,5CV]**	as above, but also set the *precision* parameter to the value of *p*CV

Flag Character

Finally, the *flag* character allows some further options:

Flag	Applicable conversion types	Description
-	**d**, **x**, **X**, **o**, **e**, **E**, **f**, **g**, **G**, **s**	left justify (if spaces need to be added to make up the minimum field width, add them *after* the number rather than before)
+	**d**, **x**, **X**, **o**, **e**, **E**, **f**, **g**, **G**	if the value is positive, prefix it with **+** character
(space)	**d**, **x**, **X**, **o**, **e**, **E**, **f**, **g**, **G**	if the value is positive, prefix it with space character
0 (zero)	**e**, **E**, **f**, **g**, **G**	pad the field with leading zero characters (rather than spaces) if required to make up the minimum field width
#	**x**, **X**, **o**	prefix value with **0x**, **0X** or **0**, respectively
#	**e**, **E**, **f**	always include a decimal point
#	**g**, **G**	do not truncate any trailing zeroes after the decimal point

Examples

Examples assume 1CV = 12345.67, 1$ = "pumpkin"	
1SERIAL("{%f[1CV]}")	→ "12345.67"
1SERIAL("{%10f[1CV]}")	→ " 12345.67"
1SERIAL("{%10.1f[1CV]}")	→ " 12345.7"
1SERIAL("{%-10.1f[1CV]}")	→ "12345.7 "
1SERIAL("{%010.1f[1CV]}")	→ "00012345.7"
1SERIAL("{%10.10d[1CV]}")	→ "0000012345"
1SERIAL("{%10.4g[1CV]}")	→ " 1.235e04"
1SERIAL("{%#10.0f[1CV]}")	→ " 12346."
1SERIAL("{%s[1$]}")	→ "pumpkin"
1SERIAL("{%10s[1$]}")	→ " pumpkin"
1SERIAL("{%10.4s[1$]}")	→ " pump"

Control String – Input Actions

The table below lists the commands available to interpret the information coming back into the Serial Channel from the serial device. Input actions are not enclosed by **{}** in the control string.

Expected data	Input Action syntax	Description
Characters	*text*	For each character in the input action string, the *DT80* will read and discard all incoming characters from the serial device until that particular character is seen. It then discards the matching character and starts looking for the next character in the input action text. For example, if the input action string is **abc** and the input data from the serial device is 3c3aabaAAc123 then all characters up to and including the second "c" will match, ie. they will be read and discarded. Non-printable characters may be specified using *****nnn (where *nnn* is the ASCII code, 1-255). **^***char* notation may also be used for control characters (ASCII 1-31), see *ASCII-Decimal* Tables *(P192)*. To include a literal **%**, **{** or **}** character, use **\%** or **\{** or **\}** respectively.
Control signal state	**\c1[***n***]** or **\c1[***n***CV]**	wait up to *n* or *n*CV milliseconds for **CTS** input to be set (high) – RS232 only
	\c0[*n***]** or **\c0[***n***CV]**	wait up to *n* or *n*CV milliseconds for **CTS** input to be cleared (low) – RS232 only
(Wait)	**\w[***n***]** or **\w[***n***CV]**	Delay for *n* or *n*CV milliseconds. Actual delay time will be approximately 2ms or 2 character times, whichever is longer.
(Erase receive buffer)	**\e**	Clear all previously received characters from the receive buffer
Fixed text string	**\m[***text***]** or **\m[***n***$]**	Read and discard incoming characters until the exact string *text* (or the text in *n*$) is seen, then discard the matching string
Numeric data	**%***{width}***type***{***[***n***CV]***}* eg. **%d[2CV]**, **%9f**	Interpret the received data according to the specified numeric format and store the result into *n*CV. If the **[***n***CV]** is not specified, the result will be returned as the return value of the channel. Note that *{ }* signifies "optional"
String data	**%***{width}***type[***n***$]** eg. **%6s[5$]**	Interpret the received data according to the specified string format and store the result into *n*$
Data to skip	**%*** *{width}***type** eg. **%*6s**, **%*f**	Interpret the received data according to the specified numeric/string format but do not store the result. In other words, skip over this data value.
One of a set of strings	**%***{width}***type['***str1***','***str2***',**...**,***n***CV***{=***m***}***]** eg. **%9s['goose','moose',23CV=2]**	If the incoming string matches *str1* then set *n*CV=0 If the incoming string matches *str2* then set *n*CV=1 If the incoming string matches *str3* then set *n*CV=2 (etc.) If a default value (**=***m*) is specified and the incoming string matches none of the strings then set *n*CV=*m*

Numeric and String Formats

These tables describe the possible values for *type* – that is, the different ways in which the incoming string of characters can be interpreted.

Type	Description	Example, assumes input data string is 123.456
f	floating point	**1SERIAL("%f[1CV]")** → 1CV = 123.456 (nothing left in receive buffer)
d	decimal integer	**1SERIAL("%d[1CV]")** → 1CV = 123 (.456 left in receive buffer)
x	hexadecimal integer	**1SERIAL("%x[1CV]")** → 1CV = 291 (.456 left in receive buffer)
o	octal integer	**1SERIAL("%o[1CV]")** → 1CV = 73 (.456 left in receive buffer)
i	decimal/hex/octal integer	**1SERIAL("%i[1CV]")** → 1CV = 123 (.456 left in receive buffer)
c	character	**1SERIAL("%c[1CV]")** → 1CV = 49 (23.456 left in receive buffer)
b	binary (no conversion)	**1SERIAL("%b[1CV]")** → 1CV = 49 (23.456 left in receive buffer)

Type	Description	Example, assumes input data string is aaba cxyab↵
s	string (↵ terminated)	**1SERIAL("%s[1$]")** → 1$ = "aaba cxyab" (nothing left in receive buffer)
S	string (whitespace terminated)	**1SERIAL("%S[1$]")** → 1$ = "aaba" (cxyab↵ left in receive buffer)
[*chars***]**	string containing only specified chars	**1SERIAL("%[abc][1$]")** → 1$ = "aaba c" (xyab↵ left in receive buffer)
[~*chars***]**	string not containing specified chars	**1SERIAL("%[~bc][1$]")** → 1$ = "aa" (ba cxyab↵ left in receive buffer)

- Conversions which may be terminated by whitespace (**%f**, **%d**, **%x**, **%o**, **%i** and **%S**) will skip over any leading whitespace, eg. **%d** will match input strings of "123", " 123" and " \013\013\010 123"
- The **%i** conversion assumes that the value is hexadecimal if it starts with 0x or 0X, octal if it starts with 0 (zero), otherwise decimal.
- The **%f** conversion will accept numbers in standard (eg. -12.39904) or exponential (eg. -1.239904e01) format.
- The **%c** and **%b** conversions treat the characters as 8-bit binary values. So the character "1" (ASCII 49) will result in the value 49 being stored in the CV.

Return Value

The **return value** of a **SERIAL** channel may be either a status code or a data value:

- If all numeric input conversions in the control string include a **[***n***CV]** specification then the **SERIAL** channel will return the status code – 0 (success), 20 (receive timeout), 5 (CTS timeout) or 29 (scan error); see *Serial Data Transmission and Reception* (P168)
- If one or more numeric input conversions in the control string do not include a **[***n***CV]** specification then the **SERIAL** channel will return the result of the rightmost conversion. If any part of the channel's evaluation fail (ie. the channel's status code is non zero) then the returned value will be the special "Not Yet Set" error value (-9.0e9).

The following example will attempt to read a floating point value from the serial sensor and return the value read

```
RA2+E 1SERIAL("%f")
1SERIAL 27.9

1SERIAL 31.2

dataTaker 80 E89 - Serial sensor receive time out
1SERIAL -9000000000.0
```

Compare this with the following example, which instead assigns the value to a CV:

```
RA2+E 1SERIAL("%f[1CV]") 1CV
1SERIAL 0 State
1CV 27.9

1SERIAL 0 State
1CV 31.2

dataTaker 80 E89 - Serial sensor receive time out
1SERIAL 20 State
1CV 31.2
```

For many applications the form where the **SERIAL** channel returns the actual value scanned provides a simpler solution.

Width

The optional *width* value specifies the maximum number of characters to read for conversion. For example, with the above example's input data: **1SERIAL("%2d[1CV]")** will result in 1CV = 12 (3.456 left in receive buffer). The default for most of the conversions (except **%c** and **%b**) is to keep reading characters until an invalid character is read. (That's why the integer conversions in the above example stop when the "." character is seen.)

The default *width* value for **%c** and **%b** is 1; with this setting the two conversions behave identically. However, if *width* is specified then:

- for **%c**, only the last character is read; preceding characters are skipped
- for **%b**, the specified number of characters are treated as a multi-byte binary word, in "big-endian" (most significant byte first) format. Note that due to the limited precision of CVs, the maximum practical width value is 3 (24 bits).

For example:

Example assumes input data is `123.456`
`1SERIAL("%1c[1CV]")` → 1CV = 49 (`23.456` left in receive buffer)
`1SERIAL("%2c[1CV]")` → 1CV = 50 (`3.456` left in receive buffer)
`1SERIAL("%3c[1CV]")` → 1CV = 51 (`.456` left in receive buffer)
`1SERIAL("%1b[1CV]")` → 1CV = 49 (`23.456` left in receive buffer)
`1SERIAL("%2b[1CV]")` → 1CV = 12594 (49*256 + 50) (`3.456` left in receive buffer)
`1SERIAL("%3b[1CV]")` → 1CV = 3223859 (49*65536 + 50*256 + 51) (`.456` left in receive buffer)

Important If *width* is not specified then the incoming data must be terminated by a non-matching character, otherwise the serial channel will continue to wait for more characters to be read, eventually returning a timeout.

For example, if the control string is **`1SERIAL("%d")`**:

Input data	Result
"`abc`"	Scan Error (return -9.0e9, `abc` left in receive buffer)
"`123`"	Receive Timeout (return -9.0e9, nothing left in receive buffer)
"`123 `"	return 123 (" " left in receive buffer)
"`123abc`"	return 123 (`abc` left in receive buffer)

Control String – Example

The control string in the Serial Channel command

`1SERIAL("\e{WN\013}%d[1CV],%f[2CV]{C\013}\w[2000]")`

specifies the following output and input actions for supervising electronic weighing scales connected to the serial channel of a *DT80*:

Input/Output action	Description
`\e`	An input action. **`\e`** erases any extraneous characters that may have been sent by the scales at some earlier time.
`{WN\013}`	An output action. Sends the "Weigh Now" command (**`WN`**) to the scales. The **`WN`** command is terminated by a carriage return (**`\013`**). (See your serial device's manual for details of its command set.)
`%d[1CV],%f[2CV]`	Three input actions. These scales return two comma-separated values: a batch number as an integer, and the weight as a floating-point value, followed by a carriage return. • **`%d[1CV]`** will interpret the first returned value as an integer batch number, and assign this to 1CV. • Skip the comma in the returned data string (**`,`**). • **`%f[2CV]`** will interpret the second returned value as a floating-point weight in kilograms, and assign this to 2CV.
`{C\013}`	An output action. These scales also have a Clear command (**`C`**), which instructs the scales to clear ready for the next weighing operation.. This output action sends the Clear command to the scales. The Clear command is terminated by a carriage return (**`\013`**).
`\w[2000]`	An input action. These scales do not respond to commands for 2s after a Clear operation. The **`\w[2000]`** action ensures that at least this time elapses following a Clear.

Important The **DeTransfer** program, which is often used to supervise the *DT80*, has a number of special commands that begin with a **`\`** (backslash) character. These are interpreted by **DeTransfer** and not sent to the *DT80*. In order to send a **`\`** character from **DeTransfer**, you need to enter a double backslash (**`\\`**). For example, the above example would be entered into **DeTransfer** as follows:

`1SERIAL("\\e{WN\\013}%d[1CV],%f[2CV]{C\\013}\\w[2000]")`

This rule applies to **DeTransfer** only; it does not apply to the "Text" window in **DeLogger**, for example.

Schedules

Executing Serial Channel Commands in Schedules

Like any other channel type, Serial Channel commands can be placed into scan schedules. For example

```
BEGIN
PS=RS485,9600
RA1M 1SERIAL("\e{01READ^M}%6f")
RB2-E 1SERIAL("\e{02READ^M}%12s[1$]",W) 1$
LOGON
END
```

This example will, once a minute, request then read a floating point value from device #1 on the multi-drop RS485 link connected to the serial channel. Also, every time digital input **2D** goes low, the serial channel will request then read a string value from device #2.

Notice that in the first schedule the scanned floating point value is the return value of the **SERIAL** channel, which will then be logged and returned. In the second schedule, the scanned string is assigned to string variable 1$. The **SERIAL** channel will then return a status code – which in this example we are not concerned about so the **W** channel option is used to make the channel a working channel (not logged or returned).

Serial commands can also be used in the "immediate" schedule, ie. executed immediately after they are entered. For example, sending

```
1SERIAL("{hello^M^J}")
```

will immediately transmit the indicated string on the serial channel.

Triggering Schedules

Sometimes the serial device connected to the Serial Channel returns data unsolicited, and so the program must be capable of responding to the device at any time. As discussed in *Trigger on External Event* (P46), any schedule (**R***a*) can be defined to trigger on the receipt of the specified string on the Serial Channel as follows:

```
Ra1SERIAL"text"
```

The text string may also be blank:

```
Ra1SERIAL""
```

in which case any character received into the Serial Channel produces a trigger.

Whenever the Serial Channel produces a trigger by either of these methods, the receive buffer will contain the string that caused the trigger, ready to be processed by a **1SERIAL** command.

In the following example a serial device transmits whitespace separated temperature readings at irregular intervals. The following job will read and log readings when they are received:

```
BEGIN
RA1SERIAL""  1SERIAL("%f","SS Temp~degC")
LOGON
END
```

Note that the **1SERIAL"***text***"** schedule trigger does not consume (ie. remove from the receive buffer) any received characters that did not match *text*. This means that there may be other characters in the receive buffer preceding the *text* string. An input action should therefore normally be included to discard any characters that do not match *text*. For example:

```
RA1SERIAL"abc:" 1SERIAL("\m[abc:]%f")
```

which will read and discard characters until the exact string `abc:` is seen.

Re-triggering

The *DT80* checks any serial schedule triggers:

- on receipt of data on the serial sensor port, and
- following execution of any schedule containing a **1SERIAL** channel.

This means that if multiple messages are received in quick succession then all will be processed in turn. For example, suppose the following schedule is entered:

```
RA1SERIAL"x:" 1SERIAL("\m[x:]%d")
```

and then the following serial data string is received:

```
x:1298 x:1265 x:0772
```

Receipt of this data will trigger the A schedule, and the **1SERIAL** channel will then parse the first value, leaving " `x:1265 x:0772` " in the receive buffer. This string still matches the schedule trigger (ie. it contains "`x:`"), so the A schedule will be immediately re-triggered.

Serial Interface Power Control

If the current job contains no **1SERIAL** commands then the serial channel interface is automatically switched off to conserve power. If the current job does contain **1SERIAL** commands then serial channel will be continuously powered.

The **1SSPORT** channel type allows you to turn power to the interface on and off under program control e.g.

```
RA1H 1SSPORT=1 1SERIAL("\w[1000]{X}%d") 1SSPORT=0
```

will, once an hour, switch on the serial channel, poll and read an integer from a serial device, then switch off the serial channel.

Serial Channel Debugging Tools

P56 Debugging

Setting **P56=1** will cause the *DT80* to output a number of **diagnostic messages**, which are useful when setting up and testing a serial channel application – or trying to figure out why it doesn't appear to be working as expected.

The following information will be returned:

- each string of output actions, and each individual input action, as they are processed
- (indented 1 space) actual transmitted data and other transmit operations eg breaks, delays as they are performed
- (indented 2 spaces) the state of the receive buffer each time something is added (ie received), each time something is removed (ie an input action matches) and the initial state – these are denoted `RxBuf+`, `RxBuf-` and `RxBuf=` respectively – also any schedules that are triggered by received characters.

Using the weighing machine example discussed earlier:

```
P56=1
RA1-E 1SERIAL("\e{WN\013}%d[1CV],%f[2CV]{C\013}\w[2000]")
1SERIAL:   RxBuf=[]
1SERIAL: InputAction: "\e"
1SERIAL: OutputActions: "WN\013"
1SERIAL:  Tx [WN\013]
1SERIAL: InputAction: "%d[1CV]"
1SERIAL:   RxBuf+[0242,1.988\013\010]
1SERIAL:   RxBuf-[,1.988\013\010]
1SERIAL: InputAction: ","
1SERIAL:   RxBuf-[1.988\013\010]
1SERIAL: InputAction: "%f[2CV]"
1SERIAL:   RxBuf-[\013\010]
1SERIAL: OutputActions: "C\013"
1SERIAL:  Tx [C\013]
1SERIAL: InputAction: "\w[2000]"
1SERIAL:  Wait (2000ms)
1SERIAL 0 State

1SERIAL:   RxBuf=[\013\010]
1SERIAL: InputAction: "\e"
1SERIAL:   RxBuf-[]
1SERIAL: OutputActions: "WN\013"
1SERIAL:  Tx [WN\013]
(etc.)
```

In this case you can see that the weighing machine returned the batch number, weight and terminating CR/LF in one burst (`RxBuf+[0242,1.988\013\010]`). The various input actions then dissected this string, removing first the batch number, then the comma, then the weight. At the end of the process the CR/LF was still in the buffer, and it was still there when the next measurement cycle began (`RxBuf=[\013\010]`). It was then cleared by the **\e** input action.

Serial Loopback

A useful technique for testing your parsing commands is to implement a serial loopback in the RS-232 mode. Simply connect the **Tx/Z** and **Rx/A** terminals together, and then send strings out of the Serial Channel by output actions. Because of the loopback, these strings appear in the receive buffer, which can then be parsed by your input actions. The strings you should send should contain data formatted in the same way that the real sensor would. In this way you are simulating the sensor for the purposes of verifying that your program can correctly interpret what it needs.

For example, if a loopback connection is used, the commands

```
1SERIAL("\e{ABCD,1234\013}%4s[1$],%4d[1CV]") 1$ 1CV
```

should store `ABCD` into 1$ and 1234 into 1CV.

Serial Channel Examples

Reading Variable Width ASCII

In this example a sensor with an RS232 interface will, in response to a **M** followed by a CR, transmit an integer status code (which we ignore), followed by four whitespace-separated floating point pressure values. This job reads and logs these values every 5 minutes:

```
BEGIN"LUCY"
PS=RS232,9600
RA5M
 1SERIAL("{\eM^M}%*d%f[1CV]%f[2CV]%f[3CV]%f[4CV]",W)
 1..4CV("~kPa")
```

```
LOGON
END
```

Reading Fixed Width ASCII

In this case a simple serial sensor continuously transmits a stream of records which consist of an `A` character followed by four 4-digit fixed point (2 decimal place) temperature values (`2209` represents 22.09, for example).

This job samples the stream every 30 seconds and logs the values it reads.

```
BEGIN"SPOT"
PS=RS232,1200,7,E,1
RA30S
 1SERIAL("\eA%4d[1CV]%4d[2CV]%4d[3CV]%4d[4CV]",W)
 1..4CV(.01,"~degC",FF2)
LOGON
END
```

Notice that the receive buffer is cleared at the start of the control string. The **1SERIAL** channel will therefore wait until the next update from the sensor. An alternative strategy would be clear the buffer at the end, in which case the **1SERIAL** channel would immediately get what it needs from the buffer. However the data it reads will be the first record in the buffer and would therefore be up to 30s old.

Reading Binary Data

In this example an even more simple sensor outputs 6 bytes of data in response to a digital signal going low. These are to be interpreted as two binary values. The first is a 16-bit integer sequence number, in big endian format (most significant byte first). The second value is a 32-bit voltage measurement, scaled such that 0x00000000 represents -17.0V and 0xFFFFFFFF represents +17.0V. For historical reasons, this value happens to be returned in little-endian (least significant byte first) format.

This job triggers a reading (by pulsing the **1D** output low) every 5 seconds and reads and logs the received values.

```
BEGIN"RAMBUTAN"
PS=RS232,115200
S1=-17,-17,0,4294967296"V"
RA5S
 1DSO(100,R)=0
 1CV(W)=-1
 1SERIAL("%2b[1CV]%b[5CV]%b[6CV]%b[7CV]%b[8CV]",W)
 1CV("Seq")
 2CV(S1)=8CV*16777216+7CV*65536+6CV*256+5CV
LOGON
END
```

Note the following points about this job:

- In this case the sequence number can be read as a single binary number, using **%2b**, but the measured value must be read byte by byte and reassembled into a single value.
- A span (**S1**) is used to scale the reading into the correct range.
- 1CV is set to an error value (-1) before each attempt to read the serial channel. If the attempt fails (eg. no data is forthcoming from the device) then 1CV will be unchanged, so the value -1 will be logged for the sequence number. This makes it easy to identify the reading as invalid.

Note CVs can only precisely store integers with absolute value less than 16,777,216 (24 bits) – above that they will be rounded. In the above example this is not a problem because the value is scaled and rounded anyway.

If, however, you need to recover all 32 bits exactly (for example if they represented 32 separate logic states) then you should read them using two 16 bit conversions and log each half separately, eg:

```
BEGIN"WHISTLE"
RA1+E
 1..2CV(W)=-1
 1SERIAL("%2b[1CV]%2b[2CV]",W)
 1CV("MS 16 bits") 2CV("LS 16 bits")
LOGON
END
```

(This example assumes the data word is in big endian format.)

Output to Serial Printer/Display

The serial channel can also be used to output selected channels to a serial printer or display. This job will measure two voltages once a minute and print the values to a serial printer:

```
BEGIN"SOUP"
RA1M
 1V(W,=1CV) 2V(W,=2CV)
```

```
 1SERIAL("{%9.3f[1CV] mV  %9.3f[2CV] mV^M^J}",W)
END
```

Output to Another dataTaker

You can also connect a the serial channel to the RS232 host port on a second *DT80* (or other dataTaker model).

The following job will send commands to a second logger to read two immediate channels, then interpret its fixed format response, which will be similar to:

```
D,000043,"",2006/02/13,18:16:54,0.191528,0;*,0,22.2172,-12.2002;0063;F2F3
```

```
BEGIN"LAMBDA"
PS=RS232,57600
RA30S
 1..2CV(W)=-999
 1SERIAL("\e{/H/R 1*TK 1+TK^M}\m[,0;*,0,]%f[1CV],%f[2CV]",W)
 1..2CV("~degC")
LOGON
END
```

In this case we first send **/H/R** to ensure that the other logger is in fixed format mode and has data returns enabled, then the three immediate channel definitions. When parsing the response, we look for the exact string "`,0;*,0,`" followed immediately by two comma separated floating point values.

Schedule Triggering (1)

In this example a barcode reader transmits a packet consisting of an STX character (ASCII 01) followed by a 7 digit ASCII integer. Once a valid barcode packet is received, the job will measure three voltages and log these, along with the barcode.

```
BEGIN"ZAMBESI"
PS=RS232,9600
RA1SERIAL"\001"
 1SERIAL("\001%7d") 'read and log barcode
 1..3V              'log voltages
LOGON
END
```

Schedule Triggering (2)

In this example a GPS unit produces an NMEA 183 data stream, eg:

```
$GPGLL,4250.5589,S,14718.5084,E,092204.999,A*2D
```

This job will read and log the latitude degrees (positive for north), latitude minutes, longitude degrees (positive for east) and longitude minutes. Each read is triggered by the `$GPGLL` header at the start of each transmission.

```
BEGIN
PS=RS232,38400
RA1SERIAL"$GPGLL"
 1SERIAL(",%2d[1CV]%f[2CV],%c[3CV],%3d[4CV]%f[5CV],%c[6CV]",2,W)
 IF(3CV><82.5,83.5){1CV=-1CV}  ' S = ASCII 83
 IF(6CV><86.5,87.5){4CV=-4CV}  ' W = ASCII 87
 1CV("Lat deg",FF0) 2CV("Lat mins",FF4)
 4CV("Long deg",FF0) 5CV("Long mins",FF4)
END
```

Note Remember that if **DeTransfer** is used to send commands then two backslash characters must be sent each time a backslash is required. (see *Control String – Example* *(P173)*)

Wiring Configurations — Analog Channels

This section contains configuration diagrams for wiring signals and sensors to the *DT80*'s analog channels (channels 1 to 5 on the right of its front panel). Note that this list of possible wiring configurations is not exhaustive – for some of the input types other variations are possible.

Analog Channels — Introduction (P18) covers important concepts you need to be familiar with to successfully use the wiring configurations presented here; concepts such as the *DT80*'s terminal designations, independent inputs and shared-terminal inputs, and sensor excitation. *Analog Channels* (P147) provides additional information about specific sensor types.

Important For all wiring configurations, the input voltage relative to **AGND** must not exceed the *DT80*'s common mode voltage limits (±3V if input attenuators are off, ±30V if input attenuators are on). Unless otherwise specified, the default for all channel types is input attenuators off. (see *Gain Ranges and Attenuators* (P18)).

Voltage Inputs

These diagrams are applicable to the following channel types:

Channel Types	Description
V	voltage
HV	voltage (attenuators on)
TB, **TC**, **TD**, **TE**, **TG**, **TJ**, **TK**, **TN**, **TR**, **TS**, **TT**	thermocouple
F	frequency
AS	analog state

V1 – Shared-Terminal Voltage Inputs

In this configuration up to three separate voltage inputs can be connected to one analog input channel. The **#** terminal acts as a shared common. See *Shared-Terminal Analog Inputs* (P19)

The optional shield may be helpful when the signal has a high output impedance or when noise pickup for other cables is a problem. Ensure that the shield is only connected to ground at one point – usually the **#** terminal on the same channel as the sensor.

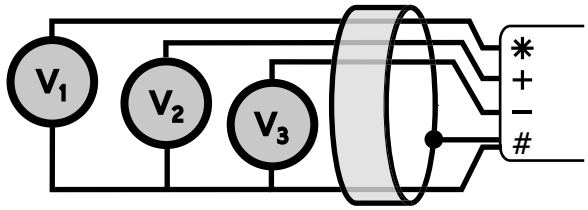


*Figure 49: **V1** Wiring for shared-terminal voltage input*

To measure	Use the command
V1	**1*V**
V2	**1+V**
V3	**1-V**

V2 – Independent Voltage Inputs

In this configuration each voltage measurement is totally independent of any other channel. The trade-off is that only one voltage can be measured per analog input channel. See Independent Analog Inputs (P19)

The optional shield may be helpful when the signal has a high output impedance or when noise pickup for other cables is a problem. Ensure that the shield is only connected to ground at one point – usually the **#** terminal on the same channel as the sensor.

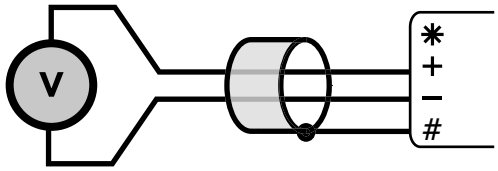


*Figure 50: **V2** Wiring for independent voltage input.*

To measure	Use the command
V	**1V**

Current Inputs

These diagrams are applicable to the following channel types:

Channel Types	Description
I	current
L	4-20mA current loop (attenuators on)

C1 – Independent Current Input with External Shunt

In this configuration an external shunt resistor is used. The value of the shunt resistor should be specified as the channel factor, otherwise 100.0 Ω will be assumed.

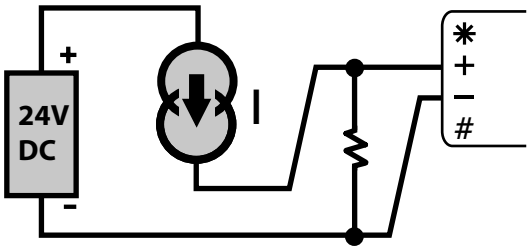


Figure 51: C1 wiring for independent current input using external shunt

To measure	Use the command
I	**1I(*R*)**

C2 – Independent Current Input using the internal shunt

In this configuration the *DT80*'s internal 100 Ω shunt resistor is used. The internal shunt resistor is connected between the channels **#** terminal and **AGND**.

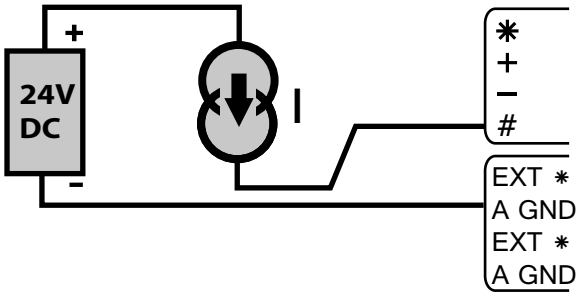


*Figure 52: **C2** Wiring for Independent current input using internal shunt*

To measure	Use the command
I	**1#I**

C3 – Shared-Terminal Current Inputs with External Shunts

In this configuration the voltages across the shunts are measured in the shared terminal configuration.

To avoid cross-channel coupling, connect the bottom of the shunts with the minimum of shared resistance to the sense point.

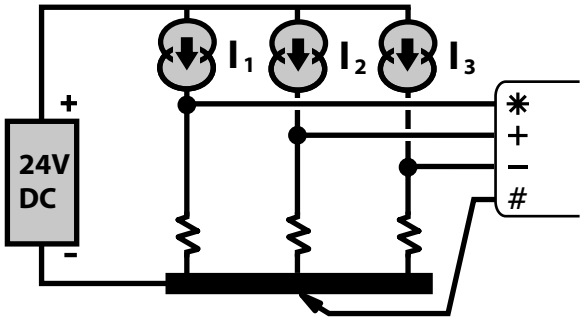


*Figure 53: **C3** Wiring for shared-terminal current input using external shunt*

To measure	Use the command
I1	**1*I(***R1***)**
I2	**1+I(***R2***)**
I3	**1-I(***R3***)**

C4 – Independent current using internal shunt and external excitation

In this configuration the *DT80* switches a single excitation supply through to each channel as it is measured (this means that between measurements the current source will be unpowered).

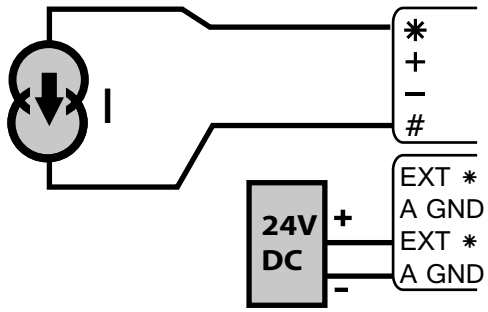


*Figure 54: **C4** Wiring for independent current using internal shunt and external excitation*

To measure	Use the command
I	**1#I(E)**

Resistance Inputs

These diagrams are applicable to the following channel types:

Channel Types	Description
R	resistance
PT385, **PT392**, **NI**, **CU**	RTDs
YS01 to **YS07**, **YS16**, **YS17**	thermistors

Default wiring configuration is **3W**.

Default excitation current is **I** (200µA) for **R**, **NI**, **YS***xx*.

Default excitation current is **II** (2.5mA) for **PT385**, **PT392**, **CU**.

R1 – 4-Wire Resistance Inputs

In this configuration the * and # terminals send an **excitation current** through the unknown resistance while the remaining terminals sense the voltage across it.

4-wire resistance methods are the most accurate because

- the resistances' lead wires are not part of the measurement circuit
- negligible current flows through the sense wires.

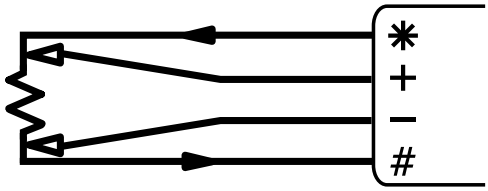


*Figure 55: **R1** Wiring for 4-wire resistance input*

To measure	Use the command
R	**1R(4W)**

R2 – 3-Wire Resistance Inputs

In this configuration the *DT80* effectively measures the voltage drop in the return lead and uses it to compensate for the voltage drop in both leads. This assumes that the excite and return lead resistances are equal.

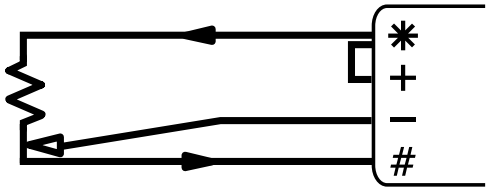


*Figure 56: **R2** Wiring for 3-wire resistance input*

To measure	Use the command
R	**1R**

R3 – 2-Wire Resistance Inputs

This configuration is only recommended if the lead resistance is negligible compared to the resistance being measured.

You can, however, compensate for the lead resistance by inserting a resistor equal to the total lead resistance (excite lead resistance + return lead resistance) between the **–** and **#** terminals, in place of the link shown.

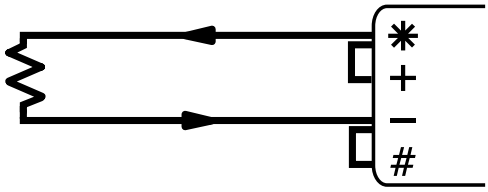


*Figure 57: **R3** Wiring for 2-wire resistance input*

To measure	Use the command
R	**1R**

Bridge Inputs – Voltage Excitation

These diagrams are applicable to the following channel types:

Channel Types	Description
BGV	voltage excited bridge

The output of a bridge channel is the ratio of output voltage to excitation voltage (expressed in ppm).

The **3W** and **4W** channel options are ignored for **BGV** channels.

B1 – 6-Wire BGV Inputs

In this configuration the bridge excitation is supplied by a precision external supply (max 3V). This actual excitation voltage is then measured by the *DT80* prior to measuring the actual bridge output voltage.

This therefore requires <u>two</u> channel definitions.

- The first (**1*V**) measures the voltage between the * and **#** terminals, and includes the **BR** (bridge reference) option which indicates that its value is to be used as the excitation voltage for subsequent bridge channels in the same schedule. It is also usually made a working channel (**W**) because it is merely an intermediate measurement.
- The second (**1BGV**) measures the bridge output voltage between the **+** and **–** terminals. The **N** (no excitation) channel option is specified because in this configuration the excitation is provided externally.

External supplies up to 6V can be supported using a similar technique to that used in configuration B2, where the excitation voltage is read using the **+** terminal rather than the * terminal. The command to measure the reference channel would then be **1+V(BR,2,W)**.

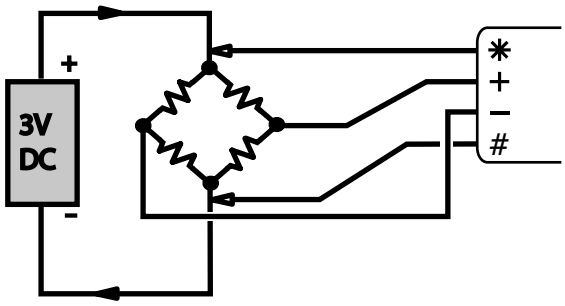


*Figure 58: **B1** Wiring for 6-wire bridge using external voltage excitation*

To measure	Use the command
bridge output	**1*V(BR,W) 1BGV(N)**

B2 – 4-Wire BGV Inputs

This configuration is similar to the 6-wire configuration except that the bridge excitation is supplied by the *DT80*'s internal voltage source (approx 4.5V). This is too large to directly measure using the * and **#** terminals, so instead we use the bridge as a 2:1 voltage divider. That is, we measure the voltage between the **+** and **–** terminals (**1+V**), then scale the result by 2 to give the actual excitation voltage. (Note also that the **V** channel option must be specified to tell the *DT80* to switch on voltage excitation while the voltage measurement is being taken.)

This configuration is not recommended if the lead lengths are long, because the return wire is carrying the excitation current and we cannot compensate for the voltage drop over the length of the return wire. This will lead to an inaccurate excitation voltage measurement, and hence an inaccurate bridge measurement.

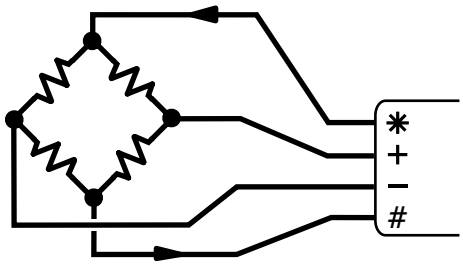


*Figure 59: **B2** Wiring for 4 wire bridge input using internal excitation*

To measure	Use the command
bridge output	**1+V(BR,2,V,W) 1BGV**

Bridge Inputs – Current Excitation

These diagrams are applicable to the following channel types:

Channel Types	Description
BGI	current excited bridge

The output of a bridge channel is the ratio of output voltage to excitation voltage (expressed in ppm). For **BGI** channels the excitation voltage is calculated from the known excitation current and the known bridge resistance.

Default wiring configuration is **3W**.

Default excitation current is **II** (2.5mA)

B3 – 4-Wire BGI Inputs

A current excited bridge is the recommended configuration for 4 wire bridge measurement, especially for bridges that are distant from the *DT80*.

In this configuration the *DT80*'s precision current source provides the excitation. To calculate the excitation voltage, the *DT80* needs to know the arm resistance, R_a, which is specified as the channel factor. The default is 350 Ω.

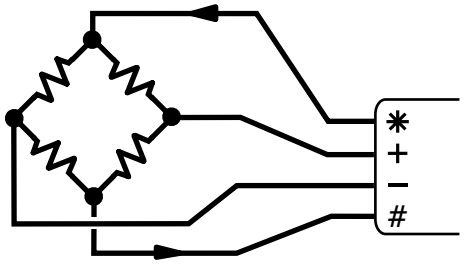


*Figure 60: **B2** Wiring for 4 wire bridge input using internal excitation*

To measure	Use the command
bridge output	**1BGI(4W,** R_a**)**

B4 – 3-Wire BGI Input

In this configuration *Rc* may be either an active arm (preferred) or a completion resistor. The resistances of *Rc* and *Ra* should be equal, and must be specified as the channel factor (default is 350 Ω).

This configuration simulates a bridge by using the *DT80*'s 3-wire compensation circuit to "compensate" for the voltage drop across *Rc*. The end result is that, like a bridge, the *DT80* measures the difference between the voltages across the two arms.

This wiring configuration is also useful for reading the wiper position of a potentiometer.

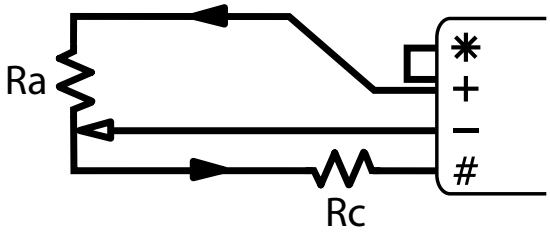


*Figure 61: **B3** Wiring for 3 wire bridge input using internal current excitation*

To measure	Use the command
bridge output	**1BGI(**R_a**)**

AD590-Series Inputs

These diagrams are applicable to the following channel types:

Channel Types	Description
AD590, **AD592**	1µA/K temperature sensors
TMP17	1µA/K temperature sensor

AD590 series devices produce a current that is proportional to temperature, which makes them useful with long lead lengths.

This configuration uses the *DT80*'s internal voltage excitation to power the sensor, and the internal shunt resistor to measure the output current.

External shunts and/or external power supplies (as per the wiring configurations for current) can also be used to allow more sensors to be measured per channel. For example configuration C3 could be used to allow three AD590s to be measured on the one channel.

A1 – 2-Wire AD590-Series Inputs

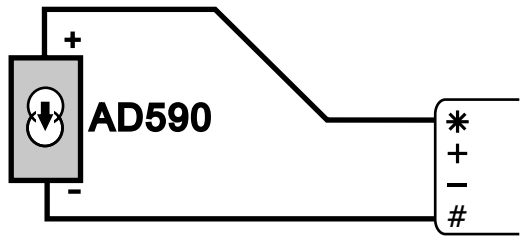


*Figure 62: **A1** Wiring for AD590 series input using internal shunt*

To measure	Use the command
temperature	**1AD590**

LM35-Series Inputs

These diagrams are applicable to the following channel types:

Channel Types	Description
LM34	10mV/°F
LM35, **LM45**	10mV/°C
LM50	10mV/°C + 500mV
LM60	6.25mV/°C + 424mV
TMP35	10mV/°C
TMP36	10mV/°C + 500mV
TMP37	20mV/°C

These sensors produce a voltage that is proportional to temperature.

The devices with a voltage offset (**LM50**, **LM60** and **TMP36**) are capable of measuring negative temperatures, the others are not.

L1 – 3 & 4-Wire LM35-Series input - full temperature range

This wiring configuration supports the full sensor operating temperature range. It requires some additional components (two resistors and two diodes). Refer to the manufacturer's data sheet for more details.

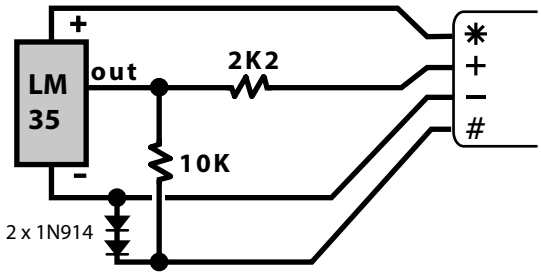


*Figure 63: **L1** for LM35 series input – full temperature range*

To measure	Use the command
temperature	**1LM35**

L2 – 3 & 4-Wire LM35-Series Inputs – restricted temperature range

This wiring configuration supports a restricted lower operating temperature range for the sensor. The temperature must be above ten degrees (10°C for LM35/45 & TMP35/37, 10°F for LM34). However this configuration does not require any additional components and requires only three wires to the sensor. Accuracy can be improved by replacing the link wire between the **#** and **–** terminals with a wire from the **–** terminal of the logger to the **–** terminal of the sensor.

Shared input configurations and/or external power supplies (as per the wiring configurations for voltage) can also be used to allow more sensors to be measured per channel. For example configuration V1 could be used to allow three LM35s to be measured on the one channel (assuming they are externally powered).

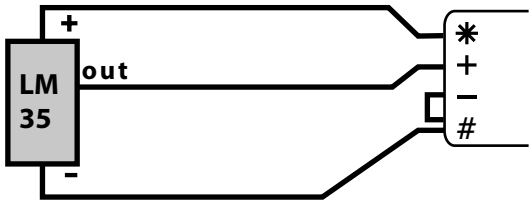


*Figure 64: **L2** Wiring for LM35 series input – restricted temperature range*

To measure	Use the command
temperature	**1LM35**

LM135-Series Inputs

These diagrams are applicable to the following channel types:

Channel Types	Description
LM135, **LM235**	10mV/°C
LM335	10mV/K

These sensors produce a voltage that is proportional to temperature.

4-Wire LM135-Series Inputs

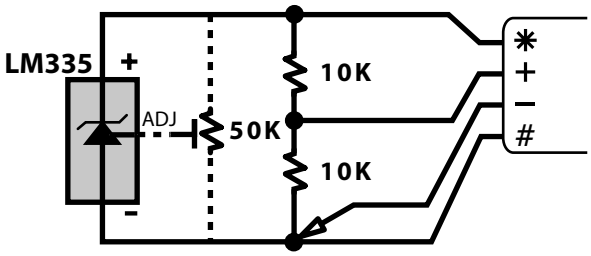
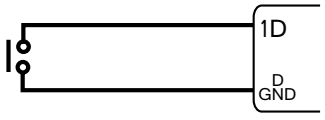


*Figure 65: **L3** Wiring for LM135 series input*

To measure	Use the command
temperature	**1LM335**

Wiring Configurations — Digital Channels

Digital Inputs

Digital Inputs	Notes	Wiring
D1 Wiring Voltage Free Contact	Inputs **1D**..**4D**, **1C**..**4C** (not **5D**..**8D**) 1DS=1 if contact open 1DS=0 if contact closed Count occurs when contact opens Channel Types: **DS**, **DN**, **DB**, **C**, **HSC**	

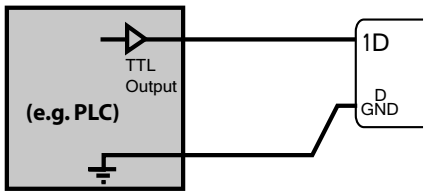
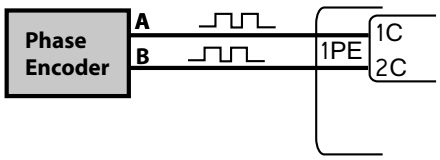
D2 Wiring Logic level digital state or counter input	Inputs **1D**..**8D**, **1C**..**4C** 1DS=1 if input is logic 1 (High) 1DS=0 if input is logic 0 (Low) Count occurs on rising edge Channel Types: **DS**, **DN**, **DB**, **C**, **HSC**	
D7 Wiring Phase Encoder	Inputs **1PE**..**2PE** **2PE** supports low threshold option Channel Types: **PE**	

Figure 66 Digital Input Wiring

Digital Outputs

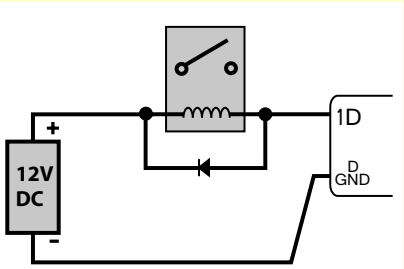
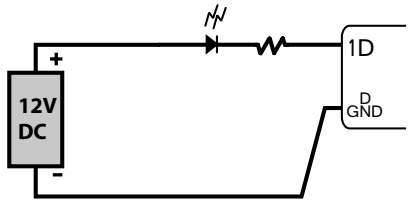
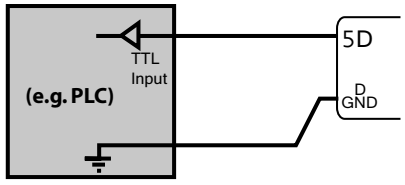
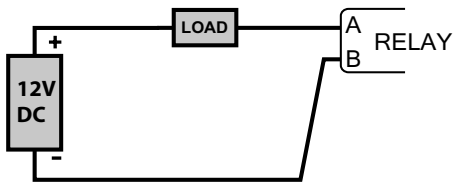
Digital Outputs		Wiring
D3 Wiring Digital output to drive relay	Outputs **1D**..**4D** (not **5D**..**8D**) **1DSO=0** to turn relay on **1DSO=1** to turn relay off	
D4 Wiring Digital output to drive LED	Outputs **1D**..**4D** (not **5D**..**8D**) **1DSO=0** to turn LED on **1DSO=1** to turn LED off	
D5 Wiring Logic level digital output	Outputs **1D**..**8D** **1DSO=0** provides logic 0 (Low) **1DSO=1** provides logic 1 (High)	
D6 Wiring Relay Output	**1RELAY=1** to turn load on **1RELAY=0** to turn load off	

Figure 67 Digital Output Wiring

Serial Channels

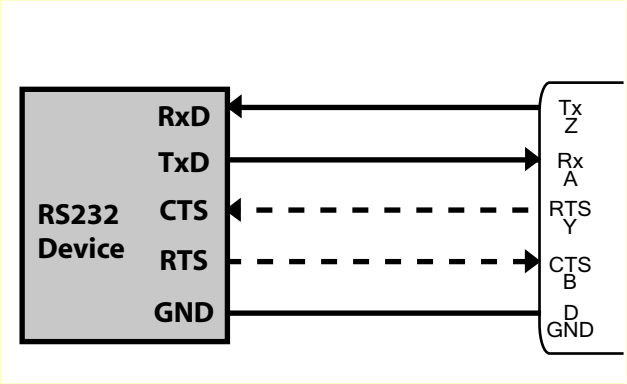
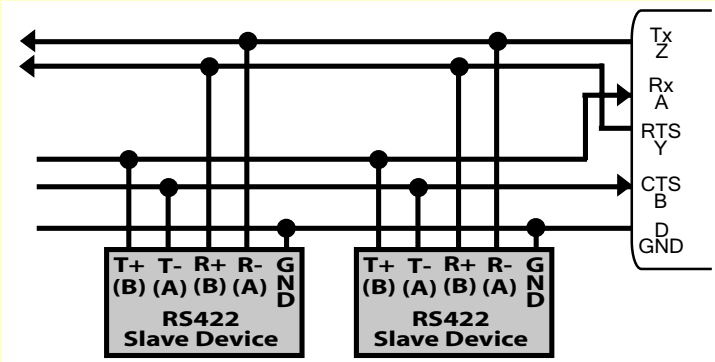
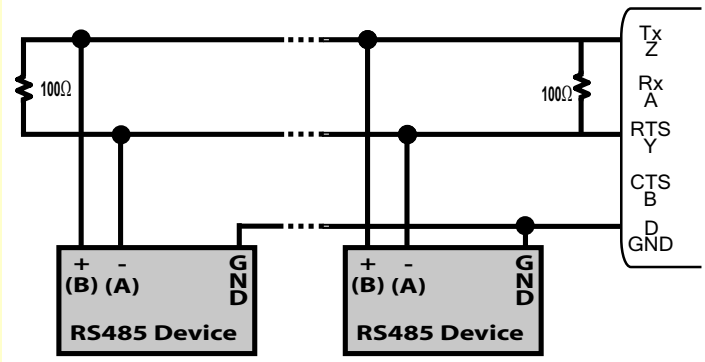
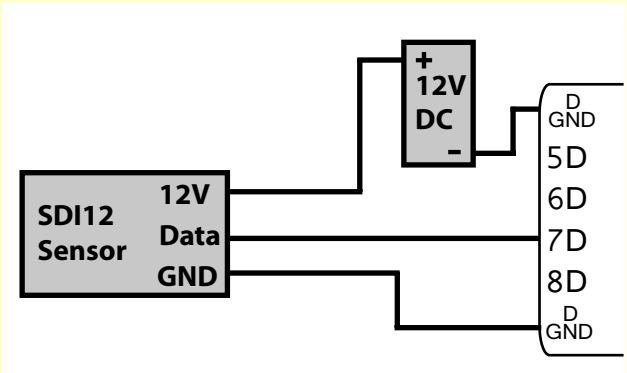
Serial Channels		Wiring
S1 Wiring Serial Channel RS232	RTS and CTS connections are optional	
S2 Wiring Serial Channel RS422	Diagram assumes that the *DT80* is the master device in the RS422 network.	
S3 Wiring Serial Channel RS485	For long cable runs or high baud rates a 100R termination resistor at each end of the cable is recommended.	
S4 Wiring Digital I/O SDI-12	If the *DT80* is externally powered, the 12V supply for the SDI-12 bus can be obtained from the *DT80*'s power input terminal block (+)	

Figure 68 Serial Channel and SDI-12 Wiring

Part O — Reference

Command Summary

The following table lists all **commands** supported by the *DT80*. It does not include:

- channel definitions (P27)
- schedule definitions (P42)
- alarm definitions (P71)

Command	Description	Page
'*comment text*	comment, remainder of line is ignored by *DT80*	55
*****	repeat last immediate schedule	49
/*switch*	set switch on (uppercase) or off (lowercase)	132
//	set all switches to their power-on default values	132
?ALL	return current values of all alarm tests	79
?*n*	return current value of alarm *n*'s test	79
?*sched*	return current value of all alarm tests for schedule *sched*	79
A *storefile-spec*	unload all logged alarms from the specified store file(s)	84
A *storefile-spec***(***from***)**	unload logged alarms from the specified store file(s), starting at timestamp *from*	84
A *storefile-spec***(***from***)(***to***)**	unload logged alarms from the specified store file(s), starting from timestamp *from* up to but not including timestamp *to*	84
A *storefile-spec***[***fromISO***]**	unload logged alarms from the specified store file(s), starting at timestamp *fromISO*	84
A *storefile-spec***[***fromISO***][***toISO***]**	unload logged alarms from the specified store file(s), starting from timestamp *fromISO* up to but not including timestamp *toISO*	84
ARCHIVE *storefile-spec*	copy logged data from the specified store file(s) to a new archive file in the same directory as the original store file	86
BEGIN	clear current job and begin entry of a new job called **UNTITLED**	54
BEGIN "*jobname***"**	clear current job and begin entry of a new job called *jobname*	54
CATTN	clear **Attn** LED	97
CERRLOG	clear error log	138
CEVTLOG	clear event log	138
CHARAC	return characterisation report	
CHARAC*n*	return line *n* of characterisation report	
CLOSEDIRECTPPP	manually close PPP connection on host RS232 port	
COPY "*srcfile***" "***destfile***"**	copy file *srcfile* to *destfile*	91
COPYDATA *storefile-spec*	copy logged data from the specified store file(s) to a new archive file on the USB memory device	87
CURJOB	return name of current job	56
DEL "*file***"**	delete file *file*	91
DELALARMS *jobspec*	delete all logged alarms for specified job(s)	88
DELDATA *jobspec*	delete all logged data for specified job(s)	88
DELJOB *jobspec*	delete program text and archive files for specified job(s)	56
DELONINSERT	delete **ONINSERT.DXC** for this *DT80* from USB memory device	57
DELONINSERTALL	delete global **ONINSERT.DXC** from USB memory device	57
DELONRESET	delete **ONRESET.DXC** file	57
DELTREE "*dir***"**	delete directory *dir* and any subdirectories	91
DELUSERINI	restore profile settings to defaults	134
DIAL	commence modem dialling	120

Command	Description	Page
DIR	return directory listing for **B:**	91
DIR "*dir***"**	return directory listing for *dir*	91
DIRJOB *jobspec*	return logging status for specified job(s)	83
DIRJOBS	return a list of all stored jobs	56
DIRTREE "*dir***"**	return directory listing for *dir* and all subdirectories	91
DT=*dtISO*	set *DT80* system date/time to *dtISO*	135
EAA	return *DT80* Ethernet adapter address (MAC address)	
END	complete definition of new job	54
FACTORYDEFAULTS	reset all settings to factory default values	136
FORMAT "*drive***"**	clear and re-format entire file system on *drive*	91
FUNCTION	return a list of all function menu definitions	95
FUNCTION*n*	return function menu item *n* definition	95
FUNCTION*n***=**	delete function menu item *n*	95
FUNCTION*n***="***label***{***actions***}**	define function menu item *n*	95
G	start all schedules in current job	51
G*sched*	start all schedules in schedule *sched* in current job	51
H	halt all schedules in current job	51
H*sched*	halt all schedules in schedule *sched* in current job	51
HANGUP	terminate active modem connection	121
IP	return *DT80* IP address	123
IPGW	return default gateway IP address	123
IPSN	return subnet mask	123
LOCKJOB *jobspec*	prevent specified job(s) from being overwritten when a new job is entered	56
LOGOFF	switch off data/alarm logging for all schedules	80
LOGOFF*sched*	switch off data/alarm logging for schedule *sched*	80
LOGON	switch on data/alarm logging for all schedules	80
LOGON*sched*	switch on data/alarm logging for schedule *sched*	80
MOVEDATA *storefile-spec*	copy logged data from the specified store file(s) to a new archive file on the USB memory device, then delete the data from the original store file	87
NAMEDCVS	return current values of all named channel variables	61
P*n*	return the current value of parameter *n*	129
P*n***=***value*	set parameter *n* to *value*	129
PASSWORD	return 1 if a comms interface password is set, otherwise 0	114
PASSWORD="*password***"**	set comms interface password	114
PAUSE *value*	delay for *value* ms	77
PH	return current host RS232 port parameters	116
PH=*baudrate***,***databits***,***parity***,***stopbits***,***flowcontrol*	set host RS232 port parameters (all parameters are optional)	116
PROFILE	return current settings for all profile keys	133
PROFILE "*section***"**	return current settings for all profile keys in *section*	133
PROFILE "*section***" "***key***"**	return current value of specified profile key	133
PROFILE "*section***" "***key***"=**	delete specified profile key	133
PROFILE "*section***" "***key***"="***keystring***"**	set specified profile key to *keystring*	133
PS	return current serial sensor port parameters	167
PS=*mode***,***baudrate***,***databits***,***parity***,***stopbits*	set serial sensor port parameters (all parameters are optional)	167
Q	terminate an unload	84
RAINFLOW:*maxcyc***:***rej%***:***m***..***n***IV**	return a rainflow analysis report	68
REMOVEMEDIA	stop using a USB memory device so it can be safely removed	90
RESET	clear current job and reset settings to power-on defaults	135
RUNJOB "*jobname***"**	replace current job with specified job	55

Command	Description	Page
RUNJOBONINSERT "*jobname*"	create **ONINSERT.DXC** for this *DT80* on USB memory device	57
RUNJOBONINSERTALL "*jobname*"	create global **ONINSERT.DXC** on USB memory device	57
RUNJOBONRESET "*jobname*"	create **ONRESET.DXC** file	57
S*n*=*a*,*b*,*c*,*d*"*units*"	define span #*n* (all parameters other than *a* and *b* are optional)	58
SATTN	set **Attn** LED	97
SDI12SEND *sdi-chan* "*sdi-cmd*"	send the specified SDI-12 command	160
SETDIALOUTNUMBER "*phonenum*"	set the number to call when **DIAL** command is issued	120
SETMODBUS *channels format scaling*	define data formats used when communicating with Modbus client system	109
SHOWPROG *jobspec*	return the program text for the specified job(s)	56
SIGNOFF	end comms session (password required to reconnect)	114
SINGLEPUSH	generate a hardware reset of the *DT80*	135
STATUS	return a status report	138
STATUS*n*	return line *n* of the status report	138
STATUS14 "*jobname*"	return internal details about specified job	138
T*n*=*a*,*b*,*c*"*units*"	define thermistor scaling function #*n* (all parameters other than *a* are optional)	59
TEST	return a test report	137
TEST*n*	return line *n* of the test report	137
TYPE "*file*"	return contents of a text file	91
U *storefile-spec*	unload all logged data from the specified store file(s)	84
U *storefile-spec*(*from*)	unload logged data from the specified store file(s), starting at timestamp *from*	84
U *storefile-spec*(*from*)(*to*)	unload logged data from the specified store file(s), starting from timestamp *from* up to but not including timestamp *to*	84
U *storefile-spec*[*fromISO*]	unload logged data from the specified store file(s), starting at timestamp *fromISO*	84
U *storefile-spec*[*fromISO*][*toISO*]	unload logged data from the specified store file(s), starting from timestamp *fromISO* up to but not including timestamp *toISO*	84
UERRLOG	return contents of error log	137
UEVTLOG	return contents of event log	137
UNLOCKJOB *jobspec*	allow specified job(s) to be overwritten when a new job is entered	56
X	trigger schedule X	49
X*sched*	trigger schedule *sched*	47
Y*n*=*a*,*b*,*c*,*d*,*e*,*f*"*units*"	define polynomial function #*n* (all parameters other than *a* are optional)	59

switch	a letter **A-Z** or **a-z**
m, *n*, *value*	an integer
a, *b*, *c*, *d*, *e*, *f*	a floating point value
sched	a schedule identifier: **A-K** or **X**
jobname	a job name (max 8 characters)
srcfile, *destfile*, *file*	a file name including drive and full path
dir	a directory name including drive and full path
drive	• **A:** (USB memory device), or • **B:** (internal file system)
storefile-spec	• "*file*" (a particular store file or archive file), or • "*jobname*"*sched* (store file for schedule *sched* in job *jobname*), or • *sched* (store file for schedule *sched* in current job), or • "*jobname*" (all store files for job *jobname*), or

Command	Description	Page
	• nothing (all store files for current job)	
jobspec	• ***** (all jobs stored on internal file system), or • **"***jobname***"** (the specified job), or • nothing (the current job)	
fromISO, *toISO*, *dtISO*	date/time in ISO format (*yyyy*/*mm*/*dd*,*hh*:*mm*:*ss*,**0.***ssssss*), all time fields are optional	
from	• *time*,*date* in P39, P31 format, or • **BEGIN**	
to	• *time*,*date* in P39, P31 format, or • **END**	
label	label to display in LCD function menu (max 16 characters)	
actions	command(s) to execute when function menu item selected (max 256 characters)	
password	comms interface password (max 10 characters)	
mode, baudrate, databits, parity, stopbits, flowcontrol	comms port parameters	
section, *key*	profile section and key names (max 80 characters)	
keystring	value of profile key (max 80 characters)	
maxcyc, *rej%*	rainflow analysis parameters	
sdi-chan	SDI-12 channel number (**5-8**)	
sdi-cmd	SDI-12 command (max 29 characters)	
phonenum	string of digits or other characters accepted by modem **ATD** command	
units	units string to use for scaled channels (max 7 characters)	

Table 7: DT80 Command Summary

Note that the spaces shown between commands and parameters are generally optional, eg.

```
BEGIN"LUPIN"
```

and

```
BEGIN "LUPIN"
```

are equivalent.

If multiple commands are specified on one line they should normally be separated by semi-colons (**;**), eg:

```
U; PAUSE 5000; LOGONA
```

although these may be omitted if the result is unambiguous, eg:

```
GA GB GC LOGONA
```

ASCII-Decimal Tables

Decimal	ASCII	Control	Description	Decimal	ASCII	Description	Decimal	ASCII	Description	Decimal	ASCII	Description
0	NUL	^A	null	32		space	64	@		96	`	backquote
1	SOH	^B		33	!		65	A		97	a	
2	STX	^C		34	"	dbl quote	66	B		98	b	
3	ETX	^D		35	#		67	C		99	c	
4	EOT	^E		36	$		68	D		100	d	
5	ENQ	^F		37	%		69	E		101	e	
6	ACK	^G	acknowledge	38	&		70	F		102	f	
7	BEL	^H	bell	39	'	quote	71	G		103	g	
8	BS	^I	backspace	40	(		72	H		104	h	
9	HT	^J	tab	41	)		73	I		105	I	
10	LF	^K	line feed	42	*		74	J		106	j	
11	VT	^L	vertical tab	43	+		75	K		107	k	
12	FF	^M	form feed	44	,	comma	76	L		108	l	
13	CR	^N	carriage return	45	-	dash	77	M		109	m	
14	SO	^O		46	.	period	78	N		110	n	
15	SI	^P		47	/	slash	79	O		111	o	
16	DLE	^Q		48	0		80	P		112	p	
17	DC1	^R	XON	49	1		81	Q		113	q	
18	DC2	^S		50	2		82	R		114	r	
19	DC3	^T	XOFF	51	3		83	S		115	s	
20	DC4	^U		52	4		84	T		116	t	
21	NAK	^V	negative acknowledge	53	5		85	U		117	u	
22	SYN	^W		54	6		86	V		118	v	
23	ETB	^X		55	7		87	W		119	w	
24	CAN	^Y		56	8		88	X		120	x	
25	EM	^Z		57	9		89	Y		121	y	
26	SUB	^[		58	:	colon	90	Z		122	z	
27	ESC	^\	escape	59	;	semicolon	91	[		123	{	
28	FS	^]		60	<		92	\	backslash	124	|	
29	GS	^^		61	=		93	]		125	}	
30	RS	^_		62	>		94	^	caret	126	~	tilde
31	US			63	?		95	_	underline	127	DEL	delete

Table 8: Standard ASCII Characters

This table lists the standard ASCII character set. The printable characters (codes 32-126) may be directly included in a *DT80* program.

For **text strings** enclosed by `""` within a *DT80* program (eg channel name, profile settings, alarm action text, etc. – but not file names), printable or non printable characters may also be entered using the "control character" notation (eg. `^M` for carriage return) or by entering a backslash followed by the decimal character code (eg. `\013` for carriage return, `\034` for double quotes). Use `^^` or `\\` to insert a single `^` or `\` character.

Note that if **DeTransfer** is used to send the command to the *DT80*, all backslashes must be entered as `\\` so that they are not interpreted by DeTransfer. So to output a single `\` in an alarm string you would need to enter eg.

```
DO"hello\\\\there"
```

in the DeTransfer send window.

Decimal	ASCII	Description	Decimal	ASCII	Description	Decimal	ASCII	Description	Decimal	ASCII	Description
128	€		160		not used	192	À		224	à	
129		not used	161	¡		193	Á		225	á	
130	‚		162	¢		194	Â		226	â	
131	ƒ		163	£		195	Ã		227	ã	
132	„		164	¤		196	Ä		228	ä	
133	…		165	¥		197	Å		229	å	
134	†		166	¦		198	Æ		230	æ	
135	‡		167	§		199	Ç		231	ç	
136	ˆ		168	¨		200	È		232	è	
137	‰		169	©		201	É		233	é	
138	Š		170	ª		202	Ê		234	ê	
139	‹		171	«		203	Ë		235	ë	
140	Œ		172	¬		204	Ì		236	ì	
141		not used	173	-		205	Í		237	í	
142	Ž		174	®		206	Î		238	î	
143		not used	175	¯		207	Ï		239	ï	
144		not used	176	°		208	Ð		240	ð	
145	‘		177	±		209	Ñ		241	ñ	
146	’		178	²		210	Ò		242	ò	
147	"		179	³		211	Ó		243	ó	
148	"		180	´		212	Ô		244	ô	
149	•		181	µ		213	Õ		245	õ	
150	–		182	¶		214	Ö		246	ö	
151	—		183	·		215	×		247	÷	
152	˜		184	¸		216	Ø		248	ø	
153	™		185	¹		217	Ù		249	ù	
154	š		186	º		218	Ú		250	ú	
155	›		187	»		219	Û		251	û	
156	œ		188	¼		220	Ü		252	ü	
157		not used	189	½		221	Ý		253	ý	
158	ž		190	¾		222	Þ		254	þ	
159	Ÿ		191	¿		223	ß		255	ÿ	

Table 9: Extended ASCII Characters - Windows CodePage 1252 / ISO-8859-1 (Latin1)

This table lists an "extended ASCII" character set. Any of these characters may be directly included in a *DT80* program, eg. by using a non-US keyboard mapping, or by holding down ALT and typing the 4-digit character code (eg. ALT-0197 for the Angstrom symbol).

Within text strings enclosed in **""**, you can also use the backslash notation (eg. **\176** for a degree symbol).

Note that many different character sets have been defined for the "extended ASCII" character codes (128-255). The character set shown above is the one most commonly used on Windows based computers, but be aware that some of these characters may appear differently or not appear at all, depending on the application program and font used to display them.

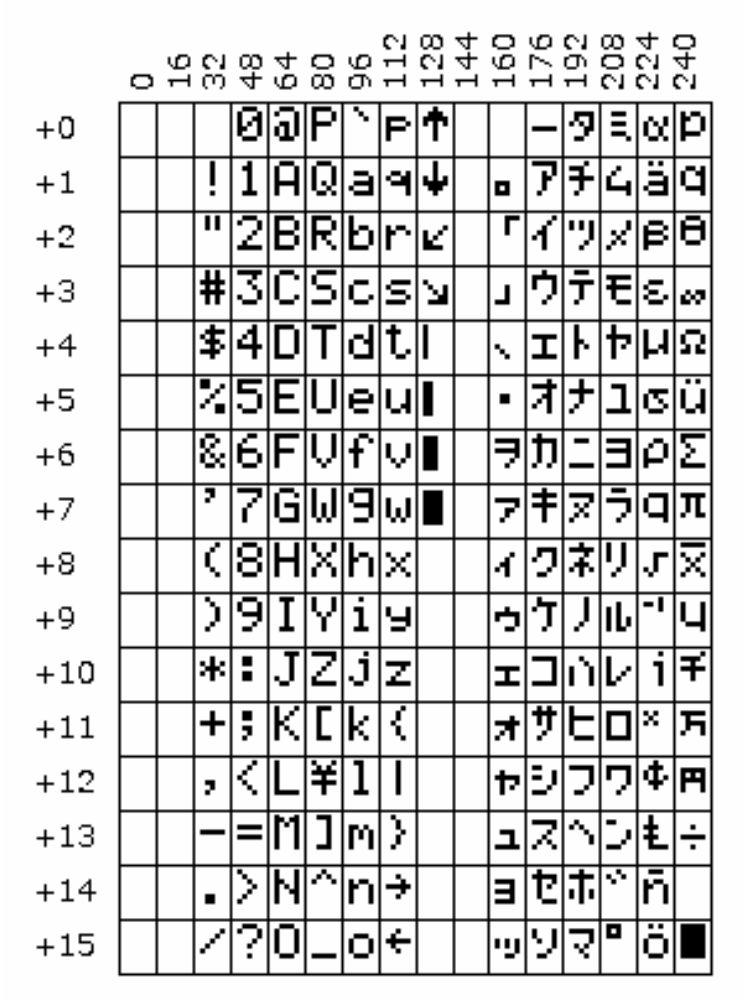


Table 10: LCD Character Set

This table lists the characters that can be displayed on the *DT80*'s LCD. Note the differences between this character set and the previous set. Thus to display a degree symbol on the LCD you would use **\223** (224+15) in an alarm text or channel name/units string.

Note that the *DT80* automatically translates the standard units strings `degC`, `degF`, `degR` and `Ohm` to `°C`, `°F`, `°R` and `Ω` when displaying on the LCD.

RS-232 Standard

The following table lists the standard RS-232 pinouts for both 9-pin (DB-9) and 25-pin (DB-25) DCE and DTE interfaces as used in *Figure 69* (P195) and *Figure 70*(P196)). Normally the DTE (*DT80*, computer) device's connector is male and the DCE (modem) device's connector is female.

Signal	Function	Direction	DB-9 Pin	DB-25 Pin
DCD	Data Carrier Detect	DTE ← DCE	1	8
RXD	Receive Data	DTE ← DCE	2	3
TXD	Transmit Data	DTE → DCE	3	2
DTR	Data Terminal Ready	DTE → DCE	4	20
GND	Signal Ground		5	7
DSR	Data Set Ready	DTE ← DCE	6	6
RTS	Request To Send	DTE → DCE	7	4
CTS	Clear To Send	DTE ← DCE	8	5
RI	Ring Indicator	DTE ← DCE	9	22

Table 11: RS-232 Pinouts

For applications where a DTE is connected to another DTE (eg. a *DT80* is connected to a host computer):

- the RXD and TXD signals must be "crossed over" so that one device's TXD is connected to the other device's RXD.
- The "Request To Send" output changes its meaning to "Clear To Send Output" (ie. a device sets it active when it is able to receive data). This allows hardware flow control to operate in both directions.
- DCD, DTR, DSR and RI are not normally used.

Cable Details

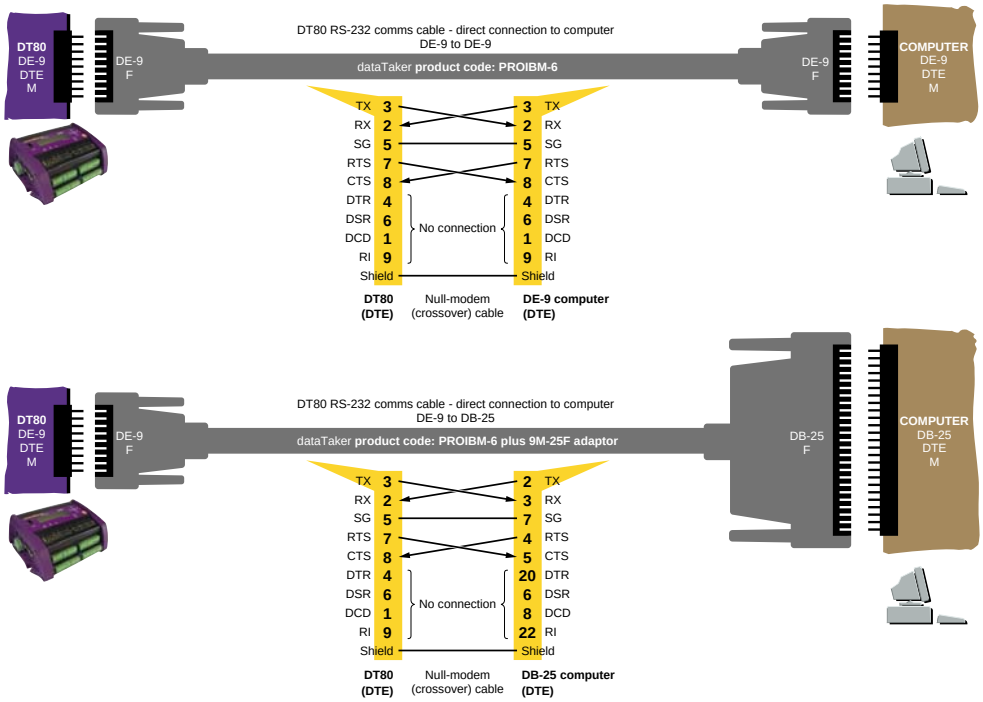


Figure 69: DT80-to-computer RS-232 comms cables

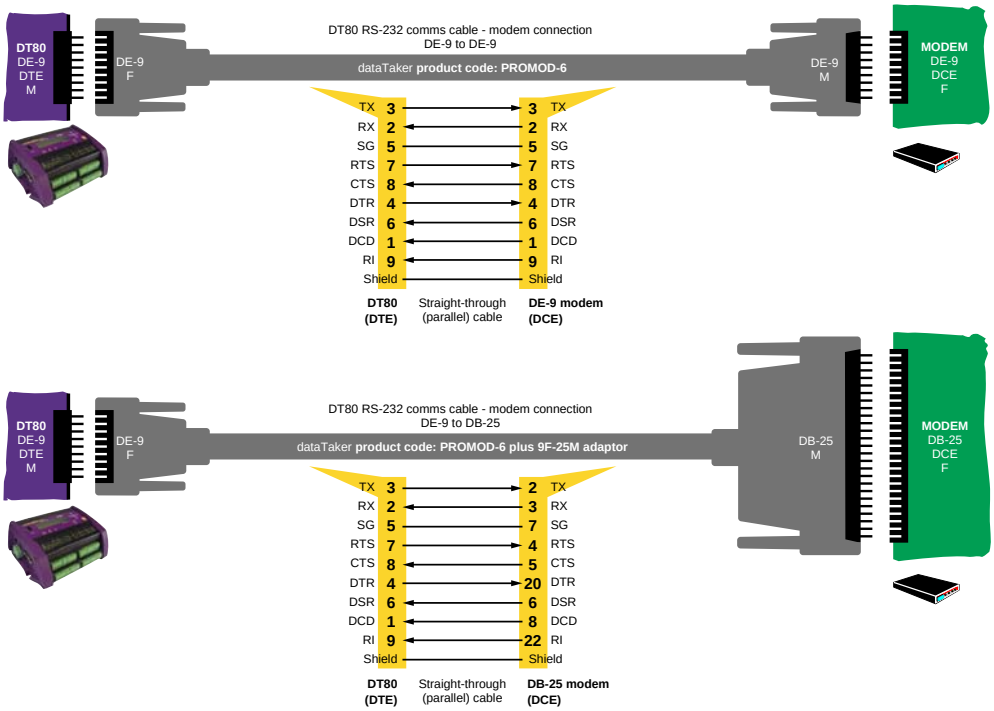


Figure 70: DT80-to-modem RS-232 comms cable

Upgrading *DT80* Firmware

The *DT80*'s "operating system" (or **firmware**) is stored in the *DT80*'s Flash memory. This means that you can easily upgrade your *DT80*'s firmware from a host computer running **DeLogger** (or **DeTransfer**).

It is strongly recommended that you keep your *DT80* up-to-date with the latest firmware. Firmware files are available for free download from www.datataker.com.

An upgrade typically takes between 2 and 15 minutes (depends on the method used).

Important Always check the **release notes** distributed with the new firmware version for any changes to the upgrade procedure documented here.

Recommended Preparation

We recommend that you carry out the following procedure before upgrading the *DT80*'s firmware to ensure that no compatibility problems arise.

This procedure returns the *DT80* to a completely unprogrammed state. Once the upgrade is complete you will have to restore any settings and programs.

1. Connect to the *DT80* and perform the following operations, most of which can be done from the text window interface in DeLogger, or from DeTransfer or other terminal software.

   Note if you are using DeLogger, it may ask if you want to upgrade the *DT80* when you connect to the *DT80*. If this occurs, answer **No** so that you can perform the following steps before the upgrade occurs.
2. Save any previously logged data stored in the *DT80*'s internal memory by unloading it to the host computer or copying to a USB memory device.
3. In DeLogger, select the Profile… option from the dataTaker menu and note any important profile settings, such as Ethernet IP Address. You must re-enter these once the upgrade is complete.

   If you are using DeTransfer or other terminal software, you can issue the **PROFILE** command to return the current profile settings.
4. Delete any ONRESET job stored in flash memory using the **DELONRESET** command.
5. Delete any PROFILE settings stored in flash memory using the **DELUSERINI** command.
6. Format the internal disk using the **FORMAT"B:"** command.

   Note This will erase all jobs and data stored in the *DT80*.

You are now ready to perform the firmware upgrade.

Recommendation — Power Before carrying out a firmware upgrade, we recommend that you charge the *DT80*'s main internal battery for 12 hours. Furthermore, if at all possible, power the *DT80* from an external source as well during the upgrade. These two precautions minimise the possibility of a power failure to the *DT80* during the upgrade.

Firmware Upgrade — Host USB or RS232 Port

Here's the procedure for upgrading the firmware of a *DT80* by using DeLogger (version 2 revision 16 or later) on a computer directly connected to the *DT80*'s Host RS-232 port. You can also use DeTransfer Version 3.18 or later.

There are also other ways you can upgrade your *DT80*'s firmware (remotely by FTP transfer, or using a USB memory device, for example). If you have a special requirement such as this contact your *dataTaker* representative.

Note If you attempt to make a connection to a *DT80*, DeLogger checks the firmware version of the *DT80* against the latest version that it can see in the **C:\Program Files\datataker\DeLogger\Firmware*DT80*** directory and offers to initiate a firmware upgrade if it finds that a later version is available. You should ensure you have properly prepared the *DT80* (as described in *Recommended Preparation* (P196) above) before either allowing this automatic upgrade to occur, or using the **Upgrade Firmware…** menu option to initiate the upgrade process, as described in the following steps:

1. Obtain the appropriate firmware ("Flashware") upgrade zip file from www.dataTaker.com or your *dataTaker* representative. Extract all files from the zip file.

   The firmware upgrade file is named according to its firmware version number and has a .DXF extension — for example, **DT80-5080002.dxf**. The size of a typical .DXF file is 1 to 2MB.

   If you are using DeLogger to perform the upgrade then the .DXF file must be copied to the **C:\Program Files\datataker\DeLogger\Firmware\dt80** directory.
2. Review the release notes (which were in the zip file). These identify the changes that have been made to the firmware, may dictate a change to the upgrade procedure described here.
3. Power the *DT80* as described in *Recommendation — Power* above.
4. Connect the host computer to the *DT80* using USB or RS232.
5. Start DeLogger and check that you're using version 2 revision 16 or later (from DeLogger's Help menu, choose About DeLogger…).

   Alternatively, start DeTransfer and check that you have version 3.18 or later.
6. In DeLogger:

   a) Do not open a connection to the *DT80*

   b) Choose **Upgrade Firmware** from the dataTaker menu.

   c) Select the correct firmware file and COM port, then press OK

   In DeTransfer:

   a) Open a connection to the *DT80*

   b) Choose **Up grade Firmware (*DT80*/800)** from the File menu.

   c) Select the correct firmware file, then press OK
7. The upgrade process will now proceed automatically. The *DT80*'s LEDs will flash and the LCD display should indicate that the upgrade is progressing. You will notice that the upgrade proceeds through three phases:

   a) A special "loader" program is downloaded (**Attn** LED flashes and display shows: ***DT80* Bootstrap**)

   b) The main firmware is downloaded (**Sample** LED flashes and display shows: ***DT80* Loader**)

   c) A hard reset is performed (display shows the normal "sign-on" screen)

   Important During the upgrade, do not remove any cables, or reset or power-down the *DT80*.
8. Once the upgrade is complete, check that the version number displayed on the sign-on screen is correct. For example if the file ***DT80*-5080002.dxf** was loaded then the display should indicate ***DT80* V5.08**.
9. Connect to the *DT80* and configure it with your preferred settings and programs.

   The upgraded *DT80* is now ready for use.

Note The above procedure can also be used to revert back to an earlier version of the firmware, should that be required.

In Case of a Failed Upgrade

In the unlikely event that something goes wrong during an upgrade (eg. power to the *DT80* or host computer is lost, or the firmware file is corrupted), use the following recovery procedure.

1. Reset the *DT80* by inserting a paper clip or similar into the reset hole (*Figure 41 DT80 Side Panel* (P140))
2. If the old firmware starts correctly, simply repeat the above upgrade procedure.
3. If the firmware does not start correctly (ie. the normal sign-on display is not shown and the *DT80* does not respond to commands) then hold down the **Edit/OK** key and reset the *DT80*. The **Attn** LED should now be flashing slowly (5s on, 5s off) and the display should show **DT80 Bootstrap**. Release the **Edit/OK** key.
4. At this point you should now be able to repeat the above upgrade procedure.

Error Messages

Standard Messages

The *DT80* returns a message when it detects an error in a command, or an operational difficulty. The form of the error report is controlled by the **/U** switch. The default is the verbose form shown in the table below. If the switch is set to **/u** the error message is reduced to an error number (e.g. E3). (Note this Switch also reduces the verbosity of other returned data).

Error messages can be switched off by the **/m** switch. The default is for errors to be reported (**/M**). During an unload operation, error reporting is disabled until the unload is complete.

If an error is detected during job entry (ie. between **BEGIN** and **END**) then the remainder of the job is ignored.

The table below lists all of the *DT80* errors, along with an explanation of their likely cause and/or correction.

Error Number and Description Cause/Action	Error Category Syntax	Operation	Memory	Reading	Hardware
E1 – Time set error • Must be in format defined by P39 and P40 • Illegal separator or non-digits entered	☑				
E2 – Command line too long • Command too long (maximum 250 characters)		☑			
E3 – Channel option error • Illegal channel option used • Incompatible options used • Option invalid for channel type	☑				
E7 – Day set error • Illegal day number entered	☑				
E8 – Parameter read/set error • Parameter index out of range • Parameter value out of range	☑				
E9 – Switch error • Illegal switch command character	☑				
E10 – Command error • Unrecognised keyword	☑				
E12 – Channel list error • Channel number outside the legal range • Incomplete channel sequence • Invalid channel type • Polynomials or spans index out of range	☑				
E18 – STATUS command error • STATUSn outside the range 1 to 14	☑				
E23 – Scan schedule error • Schedule ID not A-K, X or S • Scan time interval too large • Scan interval type invalid • Event or counter channels invalid	☑				
E24 – Unload command error • Schedule ID is not one of A-K or X	☑				
E25 – Channel table full • Internal acquisition and alarm table filled (maximum 800 entries). Additional channels cannot be declared			☑		
E29 – Poly/span declaration error • Polynomial or span index out of range (1 to 50) • Individual terms not separated by a comma • Range of terms outside 1.0e–18 to 1.0e18	☑				
E30 – Analog measurement fault • Faulty circuit board, circuit board connector, or circuit board power supply					☑

Error Number and Description Cause/Action	Error Category Syntax	Operation	Memory	Reading	Hardware
• If fault persists after a hard reset contact your *dataTaker* representative.					
E32 – Job not found • The named job cannot be found.	☑				
E37 – No current job • A command was entered that operates on the current job, but there is no job currently loaded.	☑				
E39 – Channel list fixed /F • Changes to program are not allowed		☑			
E42 – USB device not ready • No USB memory device inserted • *DT80* has not yet read the required system information from the device (wait a few seconds) • USB device is faulty or not a memory device		☑			☑
E50 – Job locked • A job that ahs been locked cannot be modified		☑			
E51 – ALARM/IF command error • Setpoint character <, >, <> or >< missing • AND, OR, XOR incorrectly entered • Setpoint not specified or too large • Delay incorrectly specified	☑				
E52 – Alarm text memory full • Memory for storage of alarm and expression text is full			☑		
E54 – Expression error • Syntax error • Expression too complex	☑				
E55 – Expression memory full • Memory for storage of expressions text is full (total is 16384 characters, shared with alarms text)			☑		
E65 – ALARM undefined • Alarm n does not exist	☑				
E74 – Serial sensor CTS detect timeout • Serial sensor: CTS did not go to the required state within timeout period				☑	
E80 – Serial device not responding • no response received from SDI-12 sensor • check cabling and sensor address				☑	☑
E81 – Serial device invalid response • garbled response received from SDI-12 sensor • check for address conflict • check for electrical noise				☑	☑
E82 – Serial device data not available • SDI-12 sensor doesn't support the requested measurement • sensor doesn't support continuous mode				☑	
E89 – Serial sensor receive time out • Serial sensor: expected characters were not received within timeout period				☑	
E90 – Serial sensor scan error • Serial sensor: could not convert the received text as specified in the control string				☑	
E98 – Flash writing error • ONRESET job too large (max 16k) • Flash memory faulty			☑		☑
E104 – Drive format failed • Unable to format the specified drive. It may be		☑			☑

Error Number and Description Cause/Action	Error Category Syntax	Operation	Memory	Reading	Hardware
write-protected or damaged.					
E106 – Pn(s) in USER.INI out of range • An out-of-range parameter has been specified in the PROFILE. It must be changed for correct operation.	☑				
E107 – Counter is already used as a trigger • The specified counter is already used as a schedule trigger and cannot be used again.		☑			
E109 – File IO error: *detailed description* • An error occurred while reading or writing to a file on one of the drives (A: or B:). The detailed description will contain exact details of the error type. For example, a write-protected file cannot be written to.		☑			
E113 – Schedule option error • Invalid schedule option specified	☑				
E114 – Command parameter error • Invalid parameter specified for a command	☑				
E115 – Serial sensor string error • Invalid syntax within serial sensor control string	☑				
E116 – Cannot log: *detailed description* • The *DT80* cannot log data for one or more schedules for the indicated reason • If the problem was that an existing job of the same name had logged data/alarms then you need to give the new job a different name, or delete the old job's data using DELDATA/DELALARMS		☑	☑		☑
E117 – Incompatible schedule store units and trigger • You can only specify storefile size by time (eg 12 hours data) if the schedule trigger is time based.	☑				
E118 – Error accessing drive *x*: *detailed description* • The indicated drive (A: or B:) could not be accessed • Media may be absent, not formatted or faulty.		☑			☑
E119 – No matching storefiles • There are no storefiles for the specified job/schedule		☑			

Table 12: DT80 Error Messages

Data Errors

Errors may also occur even though the *DT80*'s measurement system is operating normally – for example if an analog input is out of range or a connected sensor does not perform correctly. In some cases, the *DT80* returns an error message (see "Reading" error category in the above table), but mostly there is no message returned. Instead, the *DT80* flags the logged and/or returned data value as invalid.

Each logged reading has an associated "data state", which identifies whether that particular reading is valid or invalid. The following data states are possible:

Logged data state	Value when unloaded	Description
OK	measured value	No error
Overrange	`99999.9`	• input voltage exceeds max positive input voltage • input voltage exceeds positive linearization limit for sensor • input frequency too high (**F** channel type)
Underrange	`-99999.9`	• input voltage exceeds max negative input voltage • input voltage exceeds negative linearization limit for sensor • input frequency too low (**F** channel type)
Not Yet Set	`-9.0e9`	• *DT80* analog hardware fault • insufficient samples have been taken to calculate a statistical value • could not read a valid value from a serial device
Reference Error	`99999.9`	Thermocouples and some bridges require a separate reference channel measurement as part of the measurement process. This error indicates that although this channel's raw measurement was OK, the associated reference channel measurement was not.

DT80 Abnormal Resets

If a serious internal hardware or firmware failure occurs, the *DT80* will normally force a hardware reset (equivalent to a **SINGLEPUSH** command). A message will be displayed on the LCD (eg. `SW Exception`) and the **Attn** LED will flash until a keypad button is pressed.

Additional technical details about the cause of the reset will generally be logged to the *DT80*'s event and error log files. You can view these files using the **UEVTLOG** and **UERRLOG** commands.

Abnormal resets should never occur, but if you do experience one please **contact dataTaker support**. It will assist us if you can provide the following details:

- the contents of the event and error logs (cut and paste the text from the DeTransfer window)
- the job that was running at the time
- any other details regarding how the *DT80* is set up (comms connections? host applications? power?), and the circumstances leading up to the failure.

Glossary

4–20mA loop

A common industrial measurement standard. A transmitter controls a current in the range of 4 to 20mA as a function of a measurement parameter. Any receiver(s) or indicator(s) placed in series can output a reading of the parameter. Main advantage is 2-wire connection and high immunity to noise pick-up. Usually powered from a 24V supply.

50/60Hz rejection

The most common source of noise is that induced by AC power cables. This noise is periodic at the line frequency. *DT80*s are able to reject most of this noise by integrating the input for exactly one line cycle period (20.0 or 16.7ms).

Ω

ohm, a unit of resistance

µA

microamp, 10^{-6} A

µs

microsecond, 10^{-6} s

µStrain

microstrain, strain expressed in parts per million (ppm). Strain is a measure of the stress-induced change in length of a body.

µV

microvolt, 10^{-6} V

A

Ampere or amp, a unit of current

actuator

A device that converts a voltage or current input into a mechanical output.

ADC

Analog-to-Digital Converter. Part of the *DT80*'s input circuitry that converts an analog input voltage to a digital number (in other words, it converts a smoothly-varying signal to a quantised digital value). The *DT80* is a digital instrument, and therefore requires an ADC to convert analog sensor signals into digital form prior to processing. Important characteristics of an ADC are its linearity, resolution, noise rejection and speed.

ADC settling time

See *channel settling time* (P203).

Ah

Ampere-hour, a unit of electrical charge, often used when referring to battery capacity

analog

a quantity that can vary continuously through a potentially infinite number of values — for example, the time swept out by the hands of a clock, or the output of a thermocouple. Compare with digital.

append

To add a new record or data to the end of a file, database, string or list.

ASCII

American Standard Code for Information Interchange. A coding system designed for standardising data transmission to achieve hardware and software compatibility. It assigns a 7-bit code to each of the 128 standard characters: 96 visible characters — letters, numbers and punctuation marks (including the space character); 32 hardware control characters — sounding a bell, advancing a printer page, carriage return, line feed and so on.

asynchronous

Not synchronised, not occurring at pre-determined or regular intervals. A telephone conversation is asynchronous because both parties can talk whenever they like. In an asynchronous communications channel, data is transmitted intermittently rather than in a constant stream.

autoranging

The process of changing amplifier gain automatically so that the signal is amplified as much as is possible without exceeding output limits.

autozeroing

A stabilization method for removing errors due to a drift in the input offset of a measuring system. Also called **zero correction**.

base date and time

The *DT80*'s base date is 0:00:00 on 01/01/1989. All timestamps are stored as offsets from this point in time.

bit

The smallest unit of information in a computer. A bit has a single value: either 0 or 1. Computers generally store information and execute instructions in bit-multiples called **bytes**.

brackets and braces

Delimiters:

( )	Round brackets (parentheses)	Used to identify channel options, eg. **1V("Flow Rate",Y1)** group terms within expressions, eg **1CV=3.14*(2CV+3CV)**
[ ]	Square brackets	Used to enclose channel variable to be used within a serial prompt or parse command, eg. **1SERIAL("%f[17CV]")**
{ }	Braces	Used to enclose channels and commands to be conditionally executed within **ALARM** and **IF** statements , eg. **ALARM1(2CV>3){1CV=1CV+1 HB}**

signify output actions in the serial sensor command, eg.
1SERIAL("{MD004}%f[1CV]")

bps

bits per second, a measure of data transfer rate

bridge

A sensitive and stable means of measuring small changes in resistances. They are particularly useful when applied to strain gauges (as found in pressure sensors and load cells). See Bridges (P151).

buffer

An area of memory where data is held temporarily until the system is ready for it, or in case it is needed in the future.

byte

A unit of information that is eight bits long

carriage return

Also known as a **return**, usually abbreviated to **CR**. The *DT80*'s default is to suffix each scan's data with a carriage return. Symbol: ↵
In the ASCII character set, a carriage return has a decimal value of 13.

channel definition

A channel's ID followed by any channel options (in round brackets). See *Figure 8*(P42).

channel ID

A channel's number and type (eg. **3TK**). See Components of a typical schedule command (P42).

channel list

A list of channel definitions within one report schedule.

channel settling time

The time allowed for the input signal to the ADC to stabilise before it is measured. This can be controlled using the measurement delay (MDn) channel option.

channel table

An internal *DT80* data structure that stores details of all defined channels. The channel table is limited to a maximum of 800 entries.

A channel table entry is used each time a channel is referenced in the current job. For example, the job
RA10S T 4V 1CV(W)=1CV+1 ALARM2(1CV>10)"boo"{1DSO=0}
uses 5 channel table entries (for **T**, **4V**, **1CV**, **1CV** and **1DSO**).

clock

The *DT80* a real-time clock/calendar, which you can set to your actual time

CMRR

Common-Mode Rejection Ratio. A measure of the influence of common-mode voltage (unwanted) on the output of the *DT80*'s instrumentation amplifier (see common-mode voltage (P203) below).

More precisely, CMRR is the ratio of the common-mode voltage at the amplifier's input to the common-mode voltage at the amplifier's output, expressed in dB. It indicates the quality of a measuring system's input electronics. Relevant to basic (differential) inputs only.

$$CMRR = 20 \log\left(\frac{V_{CM}}{V_{out} \times A_V}\right)$$

where

V_{CM}	is an applied common-mode voltage
V_{out}	is the resulting output voltage
A_V	is the amplifier's voltage gain

command line

One or more *DT80* commands typed one after the other, separated by tab or space characters, and ending with a return character. Limited to a maximum of 250 characters (including spaces, tabs, underscores,…). For example
RA10S T 4V 5TK
is a command line made up of four *DT80* commands (separated by spaces).

common-mode voltage

An unwanted AC and/or DC voltage that offsets both inputs to the *DT80*'s instrumentation amplifier (with respect to amplifier ground). It is unwanted because it usually originates from nuisance sources such as electrical noise, DC offset voltages

caused by the sensors or the equipment being measured, or from ground loops.

Typically in industrial measurement, the sensor signals you apply to the *DT80*'s input terminals consist of

- the small component you want to measure (a few mV to a few tens of mV), PLUS
- a large unwanted component (a few V to a few tens of V) — the **common-mode voltage**.

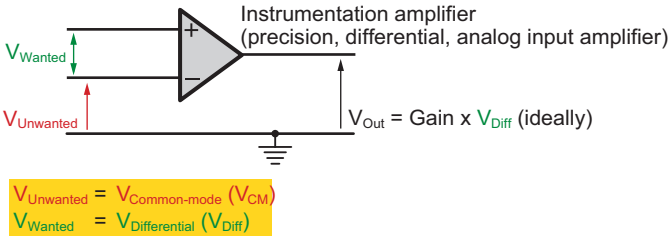


Figure 71: Common-mode voltage VCM and Differential voltage (VDiff) - 1

When the *DT80* makes a measurement, both of these components are applied to the inputs of the its instrumentation amplifier. Then, when configured for basic (differential) use, the amplifier does two things:

- It rejects most of the common-mode voltage (the unwanted signal). How well the amplifier does this is indicated by its common-mode rejection ratio — see *CMRR* (P203).
- It amplifies the difference between the signals on its two inputs. This is the wanted signal and is called the **differential voltage** — see *differential voltage* (P205).

Common-mode voltage is calculated as the average of the voltages between the measurement system's ground and the two input terminals:

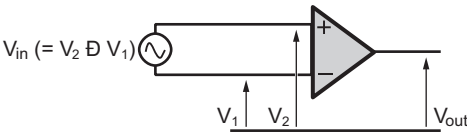


Figure 72: Common-mode voltage VCM and Differential voltage (VDiff) — 2

CR

See *carriage return* (P203).

crest factor

The peak-to-RMS voltage ratio of an AC signal (*Crest factor* (P204)).

A pure sine wave has a crest factor of 1.414. If the crest factor is less than 1.4, the waveform is flattened; if the crest factor is greater than 1.4, the waveform is peaked.

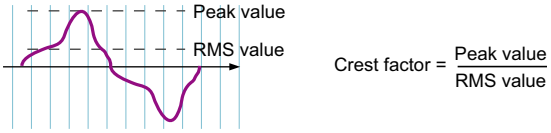


Figure 73: Crest factor

DAC

Digital-to-Analog-Converter

data acquisition system

A measurement system that scans a range of analog and digital channels, converts the readings to digital format, and forwards the data to a host. The host does any storage or data manipulation required. See also logging (P207).

data logging system

A data acquisition system with its own on-board data storage and manipulation facilities. See also logging (P207).

dataTaker

The name of the family of stand-alone data logging, acquisition and associated equipment manufactured by *dataTaker* (Aust.) Pty Ltd.

dataTaker releases:
1983 *dataTaker* DT100
1987 *dataTaker* DT200
1990 *dataTaker* DT500 series, DT600 series, and the DT50
2000 *dataTaker DT800*
2005 *dataTaker DT80*

DCE

Data Communications Equipment. A DCE device (a modem, for example) enables a DTE device (such as a computer or a *DT80*) to communicate over phone lines or data circuits. A DCE device connects to the RS-232 interface of a DTE device. See DTE (P205).

default

An attribute, value or option that is assumed if none is explicitly specified. A state or group of operating conditions (determined by the manufacturer and factory-set) to which the *DT80* automatically reverts after a reset.

differential input

An analog input where the difference between two voltages is measured, without reference to ground or any other common point. For example the **1V** command measures the differential voltage between the 1+ and 1- terminals.

differential voltage

The difference between the voltages on the two inputs of the *DT80*'s instrumentation amplifier (the *dataTaker*'s precision, differential, analog input amplifier). See common-mode voltage (P203).

digital

a quantity that is represented by a number that has a finite number of possible values. The number of bits used to store a digital value determines the resolution, ie how close two values can be and still be resolved (distinguished). Some quantities are inherently digital, eg. a logic signal or switch (whose state can be represented by 1 bit)

direct commands

Commands that perform direct tasks within the *DT80* the moment they are sent (for example, switch, parameter, unload, alarm, job and delete commands).

directory

an area on a data storage device used to store related files. Also known as a *folder*.

DTE

Data Terminal Equipment. The information source and/or destination in an RS-232 communications link. The *DT80*'s Host RS-232 port and Serial Channel are DTE devices, as is a PC's RS-232 port (serial port).

The RS232 standard was originally designed for connecting a DTE to a DCE (eg a modem). However, a DTE can also be directly connected to another DTE by means of a null-modem cable

echo

A communications option for commands you send to the *DT80*. When echo is turned on see Table 5: *DT80* Switches (P132), commands you send to the *DT80* are automatically returned to the host computer screen.

Echo is useful for troubleshooting: when the echo is on, you can see by the returned commands that the *DT80* is actually receiving them. (Once you're confident that it is receiving, you can turn the echo off.) Also, any error message appears right under the echo of the erroneous command, making the error obvious.

EEPROM

Electrically-Erasable Programmable Read-Only Memory. A special type of PROM that can be erased by exposing it to an electrical charge. Requires data to be written or erased one byte at a time (compare with Flash (P205) below). Retains its contents even when power is unavailable.

enable

Turn on or activate

Ethernet

A standard method for connecting a network of computers so that they can share information. The *DT80* supports "10 Base-T" Ethernet, that is it operates at a data rate of 10Mbps and uses Twisted-pair cable. See *Ethernet* Communications (P141).

firmware

The "operating system" software stored inside the *DT80*. The *DT80*'s firmware is semi-permanent, and you can upgrade it with a simple file transfer.

Flash

A special type of EEPROM that can be erased and reprogrammed in blocks (instead of one byte at a time — compare with EEPROM (P205) above). Flash memory is therefore much faster to erase and re-write. Retains its contents even when power is unavailable. The *DT80*'s firmware is stored in Flash memory. See also *Upgrading DT80 Firmware* (P196).

flow control

The process of controlling the flow of information between communications devices. For example, if data is being sent too quickly from a *DT80* to its host computer, the computer tells the *DT80* to temporarily stop sending data; then when the computer has caught up, it tells the *DT80* to resume sending data. See Flow Control (P118). Hardware handshaking (hardware flow control; RTS/CTS) and software handshaking (software flow control; XON/XOFF) are alternative mechanisms of flow control.

folder

Another name for **directory**

format

A specific way of organising related information. For example, the *DT80*'s internal data memory is formatted as a DOS/Windows compatible file system.

FTP

File Transfer Protocol. A TCP/IP protocol for copying files from one computer to another.

ground

A common return path that is the zero voltage reference level for the equipment or system. It may not necessarily be connected to earth.

ground loops

More often that not, grounds in a measurement system are not at the same electrical potential — differences may be from microvolts to many volts. Then, if signal wires happen to connect different grounds together, currents can flow and result in unpredictable measurement errors. These unintended conduction paths are referred to as **ground loops**. The *DT80* has been designed for maximum immunity to ground loops — see *DT80* Analog Sub-System (P154).

guard

An actively-driven shield around input signal conductors that is maintained at the common-mode voltage of the input signal.

Signal guarding is used when a sensor has a high output impedance and the cable's capacitance and insulation leakage are significant.

host computer

The computer you use for supervising the *DT80*

host software

The software you run on the host computer to supervise the *DT80*. See *DT80-Friendly Software* (P14)

hunting

An undesirable oscillation

HWFC

Hardware flow control (RTS/CTS). Also known as **hardware handshaking**. See flow control (P205).

A device using hardware flow control monitors its Clear To Send (CTS) input and will not send data until the signal is active. Conversely, a device indicates that it can receive data by driving its RTS output active (which is then connected to the other device's CTS input)

Hz

Hertz, a unit of frequency

instrumentation amplifier

A precision differential amplifier for amplifying the *DT80*'s analog input signals (wanted) and rejecting any common-mode voltage (unwanted). See Figure 43 (P154).

IP address

A device's address on a TCP/IP Ethernet network. Every device connected to an Ethernet network must be assigned its own unique IP Number. An IP address is written as four decimal numbers eg. 192.168.1.209

ISO

International Organization for Standardisation

job

A logical "hold-all" for a group of schedules and other commands, and related data and alarms. Each job has a name and a directory structure that organizes this information. See Jobs (P21).

kB

kilobyte, 1024 bytes

kbps

kilobits per second, 1024 bps

Kelvin sense point

A particular point in a measurement circuit where a measurement should be made to ensure the best possible accuracy by ensuring that unwanted voltage drops due to current flows are minimized. Symbol 

LED indicator

Light-emitting diode indicator. The *DT80* has three LEDs on the front panel, which light to indicate Sampling, Internal Disk Activity, and Attention Required. See (P93) for more details.

logging

Recording or storing data. The *DT80* logs data to its internal memory and/or an external USB memory device. Logging is a separate, user-configurable operation that the *DT80* performs in addition to its basic function of data acquisition (taking measurements from sensors connected to its inputs). See also data logging system (P204) and data acquisition system (P204).

lsb

least significant bit (within a byte)

LSB

Least Significant Byte (within a multi-byte word)

m□

milliohm, $10^{-3}\,\Omega$

mA

milliamp, 10^{-3} A

MB

megabyte, 1048576 bytes

Mbps

Megabits per second

Modbus

A widely used control and automation communications protocol, often used in SCADA systems.

monolithic sensors

Also called **IC** (Integrated Circuit) sensors. Sensors that are constructed on a single piece of silicon using integrated circuit fabrication techniques. Available sensors include those for measuring temperature (see IC Temperature Sensors (P151)), pressure, acceleration and concentration of various compounds in gases and liquids.

ms

millisecond, 10^{-3} s

msb

most significant bit (within a byte)

MSB

Most Significant Byte (within a multi-byte word)

multidrop

In communications, a multidrop configuration allows multiple devices to be connected in parallel by means of a single twisted-pair cable. This requires that each device switch off (tri-state) its transmitter when it is not actively transmitting.

multiplexer

A "many-in, one-out" switching network that allows many input signals to time-share one analog input circuit. It sequentially routes multiple channels to a single signal processing system.

noise

Unwanted voltage or current (generally with an AC component) superimposed on the wanted signal.

null-modem cable

A communications cable for connecting two DTE devices together (for example, a PC to another PC, or a *DT80* to a PC) (P195). Also known as a **crossover cable**.

nybble

Half a byte (four bits)

parse

To identify components of a command string

PC

A personal computer of the IBM or IBM-compatible type. (Although the Macintosh is technically a PC, it is not referred to as such.)

PCB

Printed Circuit Board

peak-to-peak

The value of an alternating quantity measured from its negative peak to its positive peak.

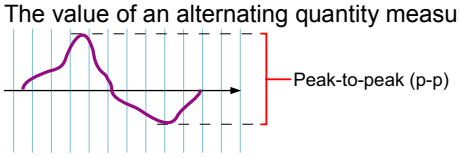


Figure 74: Peak-to-peak measurement

period

The time taken for a cyclic event to repeat itself. Reciprocal of frequency:

$$Period = \frac{1}{Frequency}$$

PID

Proportional, Integral, Derivative. A three-mode control algorithm commonly used in industrial control. A PID control loop automatically adjusts its response according to how close the measured value is to the target value. Deals with system hysteresis more effectively than simple on/off controls.

A PID loop with two-state output can be programmed on the *DT80* using the difference, integration and calculation facilities.

PLC

Programmable Logic Controller. Used to automate monitoring and control of industrial equipment.

plug-and-play

A device whose characteristics are automatically determined when it is plugged in. All USB devices are plug-and-play.

polling

Requesting information

port

A plug, socket or interface that enables connection to another device for information transfer. For example, the *DT80* has three ports for communicating with a host computer: Ethernet, USB and RS232.

PPP

Point-to-Point Protocol. A low-level protocol that allows TCP/IP based protocols to be used over an RS232 connection.

process list

The part of a schedule command that follows the schedule header and trigger, and lists the processes you want the schedule to carry out. It may include, for example, a channel list and an **IF** command.

program

A *DT80* program is a group of one or more jobs or commands that you send to the *DT80*.

protocol

The language (or set of rules) that devices use to communicate over a network. For the Information Superhighway, think of protocols as the "rules of the road". All devices on a network must use the same protocol to communicate with each other. See TCP/IP *(P210)*.

RAM

Random Access Memory. Memory that allows the storage locations within it to be accessed (written to or read from) directly (non-sequentially). This characteristic makes RAM very fast. Often simply called **memory**.

RAM disk

An area of RAM configured by a software program to emulate a disk drive.

real-time

As it happens. The *DT80* can return data directly to the host computer in real time — that is, as each scan is made, its resulting data is returned to the host computer straight away and displayed on-screen immediately.

resolution

The smallest detectable increment of measurement — that is, the smallest change in input that produces a detectable change in output. In the field of data acquisition, resolution is the number of bits that the ADC uses to represent the analog signal — the greater the resolution, the smaller the changes in input signal that can be resolved/detected.

retrieve

To unload or return data and other information from the *DT80* to the host computer, either by:

- Unloading through one of the *DT80*'s communications interfaces
- Unloading by temporarily inserting a USB memory device into the *DT80*

return

See carriage return (P203).

ROM

Read Only Memory. Memory that can be randomly read from but not normally written to. The *DT80* uses flash ROM.

RS-232

A common communications and interface standard for connecting serial devices.

RS232 uses a negative voltage (typically –5V) to represent a logic "0" and a positive voltage (typically +5V) to represent a logic "1". These signals are with respect to a common ground terminal, hence RS232 is said to use *single-ended* signalling.

RS-422

Another communications interface standard for connecting serial devices. RS-422 uses *differential* signalling (a pair of wires for each signal, no signal-ground connection) which provides improved noise immunity and allows operation over longer distances than RS-232.

RS-485

Yet another communications interface standard. Like RS-422, RS-485 uses differential signalling. RS-485 is designed for multi-drop operation over a single shared pair of wires.

RTD

Resistance Temperature Detector. A resistive sensor that changes resistance with changes in temperature. See RTDs (P150).

sampling speed

The maximum rate at which analog-to-digital conversions can be done. This includes any channel selection time, settling time (for the signal to stabilise) and processing time (if required).

SCADA

Supervisory Control and Data Acquisition. SCADA systems are used to monitor and control plant status and provide data logging facilities.

schedule

Full name: **scan schedule command**. A scan that automatically triggers whenever specified condition(s) and/or event(s) occur. For example, whenever 5 seconds have elapsed (repeating every 5 seconds), whenever a door closes (scan on digital event), or whenever an alarm occurs. This is the command you'll send to the *DT80* most often. There are several flavours of schedule command.

schedule header

The schedule's ID and trigger, eg. **RA1S** — see *Figure 8*(P42).

SDI-12

Serial Digital Interface – 1200 baud. A 3-wire multi-drop serial sensor interface, and associated protocol.

serial

One by one. In serial data transfer, data is sent in a single stream of bits, one bit at a time, one after the other. The opposite of serial is parallel. In parallel data transfer, several streams of bits are sent concurrently.

settling time

The time allowed for an input signal to stabilise after the *DT80* selects the channel, selects the gain, and applies excitation (if required). See channel settling time (P203).

shared-terminal inputs

Analog inputs where a common reference is used. Also called single-ended inputs. For example, the **1*V**, **1+V** and **1-V** commands all measure single-ended voltages relative to a common point (the 1# terminal)

See Shared-Terminal (P19).

shield

A conductor surrounding input signal wires that is generally connected to a data *dataTaker*'s ground. The purpose is to shield the input signal from capacitively-coupled electrical noise. Such a shield provides little protection from magnetically-induced noise.

SRAM

Static Random Access Memory. An extremely fast and reliable type of RAM. "Static" derives from the fact that it doesn't need to be refreshed like other types of RAM. See (P141).

S-Record

A printable ASCII format consisting of strings of hexadecimal digits; used for transferring binary data between computers.

stand-alone

Not connected to a host computer. The *DT80* is designed to operate in stand-alone mode: once programmed, you can disconnect the *dataTaker* from the host computer leaving the *dataTaker* operating totally independently. Later, to download data or reprogram the *dataTaker*, you reconnect the host computer.

state

Of an alarm: the true/false result of an alarm test.

SWFC

Software flow control (XON/XOFF). Also known as **software handshaking**. See flow control (P205).

A device using software flow control will stop transmitting if an XOFF character is received and will resume when an XON character is received.

switch

Full name: **switch command**. A software control. A two-state (ON or OFF) command that changes a *DT80* internal setting. For example, sending the switch command **/R** to the *DT80* turns ON the return-of-data-to-the-host-computer switch, and sending **/r** turns it OFF. See Switches (P132) for a complete listing.

syntax error

An error in the order, arrangement or spelling of the components of a command.

TCP/IP

Transmission Control Protocol / Internet Protocol. A commonly-used family of communication protocols. TCP/IP protocols are used on the *DT80*'s Ethernet interface, and can also be used on an RS232 link if PPP is enabled.

All TCP/IP protocols allow data to be transported across a local area network or the Internet.

TCP

Transmission Control Protocol. TCP is the default TCP/IP protocol used by the *DT80* to communicate over an Ethernet or PPP link.

TCP provides:

- flow control (prevents data being sent faster than it can be received)
- reliable data transfer (errors are detected and data is automatically re-sent)
- support for application protocols, such as e-mail and FTP

thermocouple

A temperature-sensing device constructed from dissimilar metals. See Thermocouples (P148).

transducer

A device that converts a physical parameter (temperature, for example) into an electrical voltage or current. It is usually a sensor with additional electronics for signal conditioning and scaling.

UART

Universal Asynchronous Receiver/Transmitter. A hardware component that provides an RS232 serial interface. The *DT80* uses two UARTs – one for the host RS232 port, one for the serial sensor.

UDP

User Datagram Protocol. A component of the TCP/IP suite of protocols. UDP is a simple "connectionless" protocol that operates in a similar way to an RS232 link, except that the link can be across a LAN or the Internet.

Unlike TCP, UDP does *not* guarantee that all data will be delivered.

unshared input

a differential input.

USB

Universal Serial Bus. A standard method of connecting peripheral devices to a host computer. The *DT80* operates both as a USB *device* (when talking to a host computer) and as a USB *host* (when talking to a USB memory device).

See *USB Communications* (P114)

USB Memory Device

A memory device designed to be connected to USB. These devices can either be hard disk drives or flash memory devices. They are generally powered from the USB port.

V

volt, a unit of electrical potential (voltage)

version number

The version number of the *DT80*'s firmware consists of a major number, a minor number and a build number — see Figure 75 (P211).

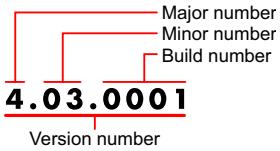


Figure 75: Version number components

XON/XOFF

Transmitter on / transmitter off. Control characters used for software flow control (SWFC), instructing a device to start transmission (XON) and end transmission (XOFF).

YSI

Yellow Springs Instruments — YSI Incorporated, 1725 Brannum Lane, Yellow Springs, Ohio 45387 (800 765-4974, 937 767-7241, Fax 937 767-9353, www.ysi.com)

zero correction

See autozeroing

Index