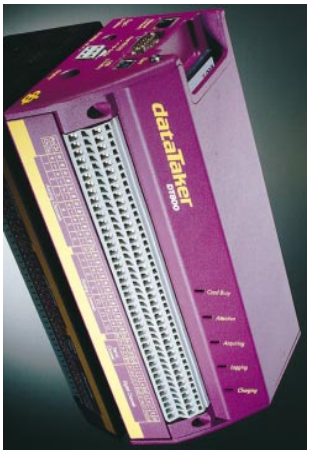
dataTaker®

User's Manual

***dataTaker* DT800 data logger**

A complete Guide to DT800:

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


dataTaker
DT600

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 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 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 Companies

Datataker Pty Ltd
Datataker Ltd
Datataker, Inc.

Related Software Products

DeLogger
DeLogger Pro
DeTransfer
DePlot
DeTerminal for Windows
DeTerminal for DOS

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.

UM-0068-A2

DT800 Firmware Covered in This Manual

This version of the *DT800 dataTaker User's Manual* (UM-0068-A2) applies to DT800s running **version 4.00 (or later)** firmware.

New in This Manual

This version of the *DT800 dataTaker User's Manual* (UM-0068-A2) contains the following changes from the previous version (UM-0068-A1):

- Minor corrections and improvements throughout
- Updated sampling speeds in tables in "Sampling Modes" (page 51)
- Updated DT800 System Variables table (page 69)
- Revised "Rainflow Cycle Counting" (page 87)
- Updated "Alarm Action Text" (page 99) — additional substitution characters
- Revised "Logging and Retrieving Alarms" (begins on page 103) — alarm action text log is no longer supported
- Updated DT800 Parameters table (page 107)
- Updated DT800 PROFILE Details table (page 113)
- Revised "Password Protection — Comms Ports" (page 124)
- Updated "DT800 Modem (Remote) RS-232 Connection" (begins on page 129)
- Revised "Digital State Outputs" (page 154)

Note Modem support is now different from that of previous firmware versions (see "RS-232 Communications" beginning on page 125).

List of Major Tables

Contents

Part C — Power

Part D — Schedules

Part E — Logging and Retrieving Data

Part F — Manipulating Data

Part G — Alarms

Part H — Housekeeping

Part I — Communications

Part J — Sensors and Channels

Part K — Reference

Part A — The DT800

DT800 Concepts

What is the DT800?

The *dataTaker* DT800 data acquisition and logging instrument (Figure 1) is a tool to measure and record a wide variety of quantities and values in the real world.

With the DT800 simple tasks are easy. For example, sending the command line

```
RA5S 1..5TJ LOGON↵
```

declares a report schedule (**RA**) that reports every five seconds (**5S**) the temperatures on five type J thermocouples connected to the DT800's analog input channels 1 to 5 (**1..5TJ**), and stores the results in memory (**LOGON**).

Commands are executed by the DT800 only after it receives a carriage-return character (↵). *dataTaker* software (see "DT800-Friendly Software" on page 10) automatically adds these carriage returns for you.

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

```
U↵
```

(the **UNLOAD** command) to the DT800 returns data to your computer in a format ready to be imported into your favourite program.

You can program the DT800 to carry out extremely powerful tasks, for which you'll need to become familiar with more of the set of *dataTaker* commands. Explore the features that are of most interest to you.

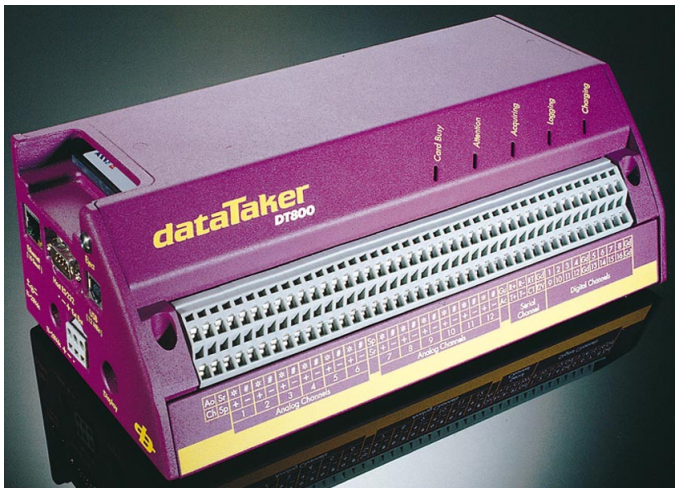


Figure 1 The *dataTaker* DT800

Connect the DT800's Internal Battery

Important The DT800 is shipped with its main internal battery disconnected. We recommend that, at your earliest convenience, you open the DT800 and connect the battery so that it can charge from the mains adapter or other external power source. This procedure is described in "Inside the DT800" beginning on page 31.

See also "Main Battery is Disconnected for Shipping" on page 40.

DT800-Friendly Software

Although you can use any terminal software to communicate with your DT800, *dataTaker* DT800-friendly software packages incorporate so many productivity features specific to data acquisition, data logging and the DT800 that make it pointless to use anything else. For example:

DeLogger has a totally graphical interface, which means that knowledge of the *dataTaker* programming language is not required. Instead, you supervise the DT800 just by clicking on icons and making selections from menus and dialog boxes. And, in addition to standard text output, you can display and print real-time and logged data in dynamic table, chart and mimic (meter) views, load data into a fully-featured spreadsheet, and replay saved data to any of the dynamic views.

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 is the easiest host software to use with the DT800 programming language. Its non-graphical interface provides complete access to all of the DT800's capabilities, and it has separate send and receive windows, which are the basis of its exceptional and unique functionality. If you prefer a command-line interface, DeTransfer is for you.

DePlot works in conjunction with DeTransfer. It graphs real-time data and unloaded data on the computer, like traces on a chart recorder.

We recommend starting with DeLogger. It's included on the CD provided with your *dataTaker* and introduced in the *Getting Started with DT800 dataTaker* user's guide (also provided). Then graduate to DeLogger Pro if you find you need its extra capabilities.

Three Modes

Normal Mode

In its **normal mode** of operation, the DT800 is designed to reject hum (from the mains electricity supply) that can be induced in the sensor wires and other wires you're using to connect to it.

You can also put the DT800 into either of two "higher gears", which cause it to scan faster than can be achieved in normal mode. These two higher gears are called **fast mode** and **burst mode**.

Fast Mode

When you apply fast mode to a DT800 input channel or group of channels, various settings are altered so that the DT800 reads those channels faster, while continuing to read other channels at normal speed. Of course, as the sampling speed is increased, the DT800 compromises by reducing resolution and noise rejection. See "Fast Mode Sampling" on page 52 for full details.

Burst Mode

In burst mode, the DT800 can sample at very high speeds, but for short periods of time. And when carrying out a burst, all of the DT800's resources are used to service the burst schedule. This means that functions such as internal housekeeping and running other schedules are not attended to during the burst. This may or may not matter depending on your application.

When a burst is armed (by the burst schedule's "normal" trigger), the DT800 begins scanning the channels listed in the schedule at the specified burst clock speed (100kHz by default). Because the amount of data collected at this rate can be very large (200kB/second), a circular capture buffer (called **burst memory**) is used to temporarily hold the measurements until the burst trigger occurs. The burst trigger defines the "snapshot" of the burst memory's measurements that are kept as the final burst samples.

Once the specified burst of readings has been taken, the DT800 automatically returns to normal operating mode.

See "Burst Mode Sampling" on page 53 for full details.

Typical Sampling Speeds

To give you an idea of how quickly the DT800 can take measurements in each of the modes, the following table compares sampling speeds for one type of DT800 input, voltage measurement with zero correction (**V** channel type):

Mode	Sampling Speed in Hz (samples/second)	
Normal	40Hz for one **V** channel 7Hz for each of 10 **V** channels sampled simultaneously	Best resolution and noise rejection
Fast	200Hz for one **V** channel 70Hz for each of 10 **V** channels sampled simultaneously	Reduced resolution and noise rejection
Burst	25kHz for one **V** channel 1.5kHz for each of 10 **V** channels sampled simultaneously	

"Sampling Modes" beginning on page 51 covers these modes in detail.

Uppercase, Lowercase

The DT800 only responds to uppercase characters, except for **switch** commands (see "Switches" on page 111).

Use lowercase characters to document and add clarity to commands. For example, `Time` is the same as `T`, and `Report_schedule_A_every_15_Minutes` is the same as `RA15M`. (Use the underscore character to improve the readability because the "space" character is a command separator.)

User Defaults and Startup

The DT800 can store your preferred configuration settings and automatically load them every time it is reset or powered up. The settings are stored in the DT800's USER.INI file, and you use `PROFILE...` commands to modify this file. See "User Startup Profile" on page 113. You use this method to set defaults for time format, date format and mains frequency — see the PARAMETERS section in the DT800 PROFILE Details table on page 113.

In addition, the DT800 can automatically run a particular job every time it is reset or powered up. See "Startup Job" on page 115.

Power

"Powering the DT800" beginning on page 40 discusses the ways you can provide power to the *dataTaker*.

For applications where power consumption is critical, the DT800 has a sleep mode that reduces battery current drain from approximately 400mA (maximum) or 150mA (typical) to just 300µA. The DT800 automatically wakes from sleep mode when input channels are due to be scanned. Plan your DT800 program to ensure that the DT800 does not wake more often than is necessary. This applies particularly to the statistical sub-schedule (page 49) and alarms (page 96). See "Low-Power Operation" on page 43.

Operating Environment

The DT800 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 your DT800 gets wet, immediately disconnect and remove all power sources (including the main internal battery), and dry the DT800 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 DT800 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 DT800's exposure to temperature extremes.

Ways of Using the DT800

You can deploy the DT800 in many ways depending on factors such as location, data volume and power availability:

- on-line to a host computer, with the DT800 as a front end
- 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 DT800
- data recovery (and programming) using removable memory cards

The method of deployment influences the fine tuning of the DT800'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.

Operating System Upgrade

The DT800's operating system exists in the DT800 as "firmware". You can easily upgrade this

- from a connected (RS-232 or Ethernet) host computer running appropriate *dataTaker* host software, using an upgrade file obtained from your *dataTaker* representative or **www.datataker.com**
- by inserting a specially-prepared PC Card into the DT800.

See "Upgrading DT800 Firmware" beginning on page 193.

Fundamental Inputs and Ranges

The DT800 can measure

- voltage
- resistance
- frequency.

It derives all other measurements from these **fundamental inputs**.

The following table lists the DT800's input ranges for these fundamental signals, along with their resolutions:

	Full Scale			Resolution
V	±10mVdc	or	mVac p-p	1µV
	±20mVdc	or	mVac p-p	2µV
	±50mVdc	or	mVac p-p	5µV
	±100mVdc	or	mVac p-p	10µV
	±200mVdc	or	mVac p-p	20µV
	±500mVdc	or	mVac p-p	50µV
	±1Vdc	or	Vac p-p	100µV
	±2Vdc	or	Vac p-p	200µV
	±5Vdc	or	Vac p-p	500µV
	±10Vdc	or	Vac p-p	1mV
	±13Vdc	or	Vac p-p	2mV

	Full Scale	Resolution
R	20Ω	100µΩ
	50Ω	25µΩ
	100Ω	500µΩ
	200Ω	1mΩ
	500Ω	3mΩ
	1000Ω	5mΩ
	2000Ω	100mΩ
	5000Ω	25mΩ
	10000Ω	50mΩ
F	10kHz	0.01Hz

Accuracy of the DT800

	25°C	45°C to 70°C
DC voltage measurement	±0.02%	±0.10%
DC resistance measurement	±0.04%	±0.20%
	% of reading ±0.01% of full scale	

The temperature coefficient for the channel factor is 20ppm/°C maximum.

Analog Channels — Introduction

Connecting Sensors

You must know the output signal type and magnitude for each sensor. Make sure that the input signal to the DT800 does not exceed the *dataTaker*'s ratings. As a general rule, the voltage on any analog input terminal should be within –13 to +13 volts relative to DT800 ground.

Then go to

- "Wiring Configurations — Analog Channels" beginning on page 168, or
- "Wiring Configuration — Digital Channels" beginning on page 184

and connect the sensors to the DT800 as shown there.

Analog Input Channels

An analog input channel on a DT800 is a 4-wire connection (Figure 2) 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.

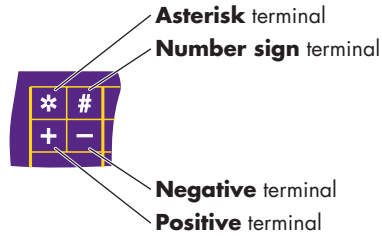


FIGURE 2 Analog input channel terminal labels

Channel Pairs

For some measurement purposes — see especially "Shared-Terminal Analog Inputs" below — the following adjacent analog channels are each considered as a **channel pair** — that is, a group of eight terminals where some functions are shared across the two adjacent channels. See, for example, Figure 115 on page 173. Each group of two analog channels shares a multiplexer (six multiplexers in total — see Figure 3, and Figure 92 on page 148):

Analog input channels **1 and 2**	Analog **channel pair 1**	Analog input multiplexer 1
Analog input channels **3 and 4**	Analog **channel pair 2**	Analog input multiplexer 2
Analog input channels **5 and 6**	Analog **channel pair 3**	Analog input multiplexer 3
Analog input channels **7 and 8**	Analog **channel pair 4**	Analog input multiplexer 4
Analog input channels **9 and 10**	Analog **channel pair 5**	Analog input multiplexer 5
Analog input channels **11 and 12**	Analog **channel pair 6**	Analog input multiplexer 6

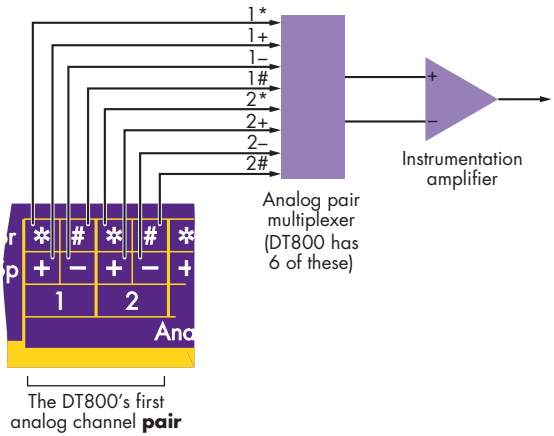


FIGURE 3 The first of the DT800's six channel pairs

Multiplexers

The DT800's multiplexers are essentially patch-boards that direct signals from the channel terminals to the instrumentation amplifier input (see Figure 3). Many different connections are possible.

Autoranging

The DT800's default is for its instrumentation amplifier to automatically change range to suit the input signal applied to it by the multiplexers.

P59 determines the maximum gain for automatic gain-ranging (default is 10mV) — see page 110.

If you're certain of the amplitude of your input signals, you may want to set the gain manually. You 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 — see page 72.

Analog Input Configurations

Your application determines how you connect sensors and signals to the DT800's analog channels. There are two configuration options:

- **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.

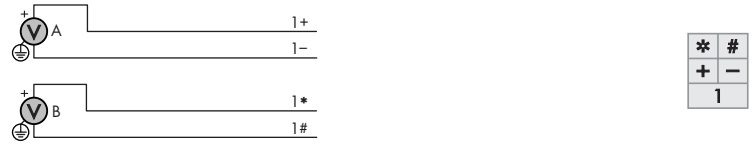


FIGURE 4 Wiring one or two independent inputs to a single channel (voltage inputs used as example)

An independent input that requires more than four terminals is connected to a channel pair — see Figure 117 (page 174).

An independent input is one in which the signal is the voltage between two wires and neither wire is necessarily at ground potential. On the DT800, each channel's **+** and **–** terminals, and each channel's ***** and **#** terminals, provide for a dedicated independent analog input.

The DT800's multiplexer patches each scanned channel's **+** terminal to the DT800's instrumentation amplifier **+** input, and the **–** terminal to the amplifier's **–** input. This patching is achieved by defining the **channel number** and **channel type**. For example, a **V**oltage on channel **1** is patched by the channel definition **1V**.

The section "Wiring Configurations — Analog Channels" beginning on page 168 contains many sensor/signal types that are wired using independent input configurations.

See also "Which Analog Input Configuration Should I Use?" below.

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, the three sensors share channel 1's **#** terminal. Each of the three inputs is a shared-terminal input.

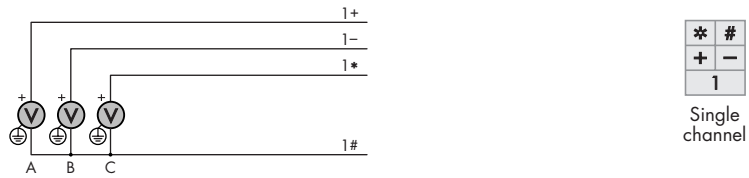
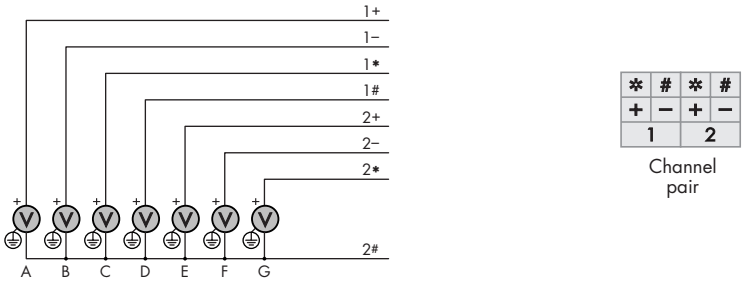


FIGURE 5 Shared-terminal voltage inputs sharing a channel's **#** terminal (voltage inputs used as example)

A shared-terminal input simply provides a signal voltage between two wires. On the DT800, 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) is recognized by the channel definition **1+V**.

Inputs can also be wired to the terminals of a channel pair (see "Channel Pairs" on page 12) so that they share the # terminal of the even channel of the pair. For voltage sensors, for example, this allows the connection of up to seven shared terminal inputs as shown in Figure 6.



Channel pair

FIGURE 6 Shared-terminal voltage inputs sharing a channel pair's common return terminal, which is the even channel's # terminal (voltage inputs used as example)

When shared-terminal inputs are connected to a channel pair, each input's channel definition is suffixed by **(#)**, the common terminal's symbol enclosed by round brackets. For example, the channel definition for sensor A in Figure 6 is **1+V(#)**, where **(#)** indicates that the voltage sensor on channel **1**'s **+** terminal is also connected to the common/shared return terminal of a channel pair. And, because the other half of channel 1's pair is channel 2, you know that 2# is the common terminal.

The section "Wiring Configurations — Analog Channels" beginning on page 168 contains many sensor/signal types that are wired using shared-terminal input configurations on a single channel and on a channel pair.

See also "Which Analog Input Configuration Should I Use?" below.

Which Analog Input Configuration Should I Use?

Consider the following:

- Shared-terminal inputs reduce the number of wires necessary (because of the shared point) and are adequate if you have several input signals with a readily-shared common point.
- Independent inputs are preferred if
  - your system does not have a readily-shared common point, or
  - the input signals are low level, or
  - the input signals have measurable/significant voltage differences between their return wires.
- In most cases, independent inputs give greater data accuracy than shared-terminal inputs. (But you may not always need the increased accuracy — beware of adding unnecessary complexity to your acquisition and logging system.)

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 DT800 can provide

- 0–10Vdc voltage excitation (12-bit resolution = 2.44mV steps)
- 0–20mA current excitation (12-bit resolution = 4.88µA steps)
- 0–200mW power excitation (12-bit resolution = 48.8µW steps).

These are output on the appropriate terminal of each channel when the channel is read. This action is automatic for most sensor types, but can also be initiated by a channel option. See the Excitation category in the channel options table beginning on page 71.

Digital Channels — Introduction

To see where the DT800's digital channels are located physically and conceptually, see Figure 11 (page 24) and Figure 92 (page 148).

Note that you can also use analog channels as (digital) state inputs — see "Analog Logic State Inputs" on page 147.

The main functional groups of the DT800's 16 channels are shown in Figure 7.

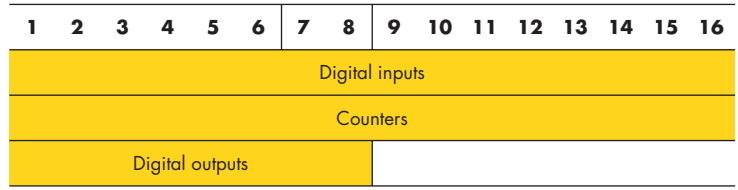


FIGURE 7 Main functions of the DT800's digital channels

By sending commands to the DT800, you configure each digital channel for use as either

- a standard TTL digital input, or as
- a low-level digital input — channels D7[1] and D8 only, or as
- a digital state output — channels D1 to D8 only, or as
- a 32-bit counter.

This is covered in detail in "Digital Channels" beginning on page 152.

You use the 16 digital channels to monitor digital states, or use them as counters. The two low-level digital inputs are intended for use with inductive-pickup (magnetic) sensors. The eight digital state outputs (open-drain FET, 30V 100mA) can be used to drive relays and other control devices.

Counters and Sleep Mode

Two of the three groups of counters (see Figure 97 on page 152) continue to operate when the DT800 is in sleep mode.

There are steps you can take to maximize battery life when using digital counters. See "Counter Rollover Rate" on page 157.

[1] In this manual, digital channel numbers are prefixed by the letter **D** to distinguish them from analog channel numbers, which are not given a prefix.

Programming the DT800

When creating a program to send to the DT800, you'll typically work in the following order:

Specify Channel Types

The input channels are very versatile, but the DT800 does not automatically know what type of sensor is connected — you must tell it. A channel is defined by a channel type that determines how the multiplexer is patched and how the readings are to be processed. There are more than thirty different channel types (see the table beginning on page 63).

A particular 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.

In very general terms, when working with your DT800, you firstly select the most appropriate **channel type** for each sensor from the table beginning on page 63. The Wiring Configuration column shows appropriate wiring configurations; connect the sensors accordingly.

Add Channel Options

Then, you use **channel options** to modify channel function. In a channel definition these are listed in round brackets immediately after the channel type. The table beginning on page 70 describes the channel options.

Test Each Sensor

Next, we recommend that you test each sensor by declaring a simple schedule. For example

```
RA1S 2PT385(4W)
```

returns every one second (**RA1S**) the temperature of a platinum resistance temperature sensor (**PT385**) connected as a 4-wire resistance (**4W** channel option) on channel **2**.

For the next level of programming detail, go to "Programming the DT800" on page 15 and "Working with Schedules" on page 58.

Schedule Commands

You can program the DT800 by sending individual commands to it, by sending several commands all on the one line, or by sending a program. Figure 8 shows a sample DT800 program.

You program the DT800 by sending **schedules** and other commands to it from any of the following:

- a host computer
- a memory card (any program present on the memory card can be automatically downloaded to the DT800 when the card is inserted)
- an alarm (the DT800 can re-program itself if an alarm occurs)

Sent commands are not processed by the DT800 until it receives a carriage return (*dataTaker* supervision software inserts this character automatically — you don't have to type it every time). Note the following:

- The input buffer is 254 characters, so command lines must not exceed this length.
- Each command must be separated by one or more spaces, tabs or carriage returns.
- All schedules must be entered on one line or placed between the BEGIN and END keywords.

Schedules in More Detail

A **schedule** is a list of channels preceded by a scan trigger specification — see Figure 40 on page 44.

As a general rule when creating schedules, don't instruct the DT800 to read channels more frequently than is really necessary. For example, temperatures generally change slowly so rapid reading does not provide extra useful information.

You can declare up to eleven different schedules, 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. This allows the trigger to be modified by the program itself (see "Alarm Action Text" on page 99).

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. Schedules are discussed in detail beginning on page 44.

Important A schedule's 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" beginning on page 58").

Jobs

A DT800 **job** is a logical "hold-all" for a group of schedules and other commands, and related data and alarms. Each job has a directory structure that organizes these components. The command **BEGIN** signifies the start of a job, and the command **END** signifies the end of the job — see Figure 8 on page 17. A job comprises all statements beginning with and including **BEGIN**, up to and including **END**.

The DT800 can store more than one job, but only one can be the current/active job. A job remains current in the DT800 until

- you reset the DT800 (see "Resetting the DT800" beginning on page 118), or
- you send a new job to the DT800, or
- you use the **RUNJOB"*JobName*"** command to make ***JobName*** the current job (see the DT800 Job Commands table on page 16).

Job Name

A job must always have a name. If you don't supply a name, the DT800 assigns the default name **UNTITLED** .

Recommendation Because the job name is used to determine if data should be appended to existing data files, we recommend that you name each job, and that you make each name unique. This avoids confusion that could arise from having several jobs with the name **UNTITLED**, or several jobs with the same user-assigned name.

You assign a name to a job by enclosing it (maximum of 8 characters, no spaces, not case-sensitive) between straight quotation marks (") immediately after (no space) the **BEGIN** command. (If you omit **"*JobName*"** and simply send **BEGIN**, the DT800 assigns the default name **UNTITLED** to that job.)

You create a job when you send

```
BEGIN
```

or

```
BEGIN"JobName"
```

to the DT800. (Remember, if you omit **"*JobName*"**, the DT800 assigns the default name **UNTITLED** to the job.)

Program: a Holder for Jobs

You can create several jobs, one after the other, in DeTransfer's send window or DeLogger's text window and send them to the DT800 all at once (with one action). This group of jobs is called a DT800 **program**.

UM-0068-A2

When you send the program, all its jobs are stored in the DT800 for future recall, and the last one sent becomes the DT800's active job. (If you send the jobs one-by-one, each new job replaces the previous one so that there is only ever one active job in the DT800.) You can change the DT800's current job by sending the **RUNJOB"*JobName*"** command (see the DT800 Job Commands table below).

Subroutines

When you send two or more jobs in a program (that is, all at once), they can work like subroutines. For example, if an out-of-range measurement triggers an alarm in the current job, it can "call" (load and run) another job that you've written to deal with alarm situations.

Job Commands

The DT800 supports these job-related commands:

BEGIN"*JobName*" ⇓ **END**	When sent to the DT800, creates a job called ***JobName***	
BEGIN ⇓ **END**	When sent to the DT800, creates a job called **UNTITLED**	
DIRJOBS	Returns a report of all jobs stored in the DT800 (+ indicates locked job, * indicates current job)	See Figure 9 (page 18).
DIRJOB"*JobName*"	Returns a report comprising various ***JobName*** details (by schedule)	
DIRJOB	Returns a report of the current job and its details (by schedule)	
DIRJOB*	Returns a report of all jobs and their details (by schedule)	
SHOWPROG	Returns the current job's program file	
SHOWPROG"*JobName*"	Returns ***JobName***'s program file	
SHOWPROG*	Returns all currently-defined jobs' program files	
LOCKJOB	Protects the current job from deletion, its program from modification, and its data from deletion	
LOCKJOB"*JobName*"	Protects ***JobName*** from deletion, its program from modification, and its data from deletion	
LOCKJOB*	Protects all jobs from deletion, their programs from modification, and their data from deletion	
UNLOCKJOB	Allows the current job to be deleted, its program to be modified, and its data to be deleted	
UNLOCKJOB"*JobName*"	Allows ***JobName*** to be deleted, its program to be modified, and its data to be deleted	
UNLOCKJOB*	Allows all jobs to be deleted, their programs to be modified, and their data to be deleted	
DELDATA	Deletes the current job's data from the DT800	
DELDATA"*JobName*"	Deletes only ***JobName***'s data from the DT800	
DELDATA*	Deletes all jobs' data from the DT800	

Table: DT800 Job Commands (sheet 1 of 2)

DELJOB	Deletes the current job from the DT800
DELJOB"*JobName*"	Deletes only ***JobName*** from the DT800
DELJOB*	Deletes all jobs from the DT800
DELALARMS	Deletes the current job's alarms from the DT800
DELALARMS"*JobName*"	Deletes only ***JobName***'s alarms from the DT800
DELALARMS*	Deletes all jobs' alarms from the DT800
U	See "Unload Commands" beginning on page 82.
U(*from*)(*to*)	
U[*from*][*to*]	
Ux	
Ux(*from*)(*to*)	
Ux[*from*][*to*]	
U"*JobName*"	
U"*JobName*"(*from*)(*to*)	
U"*JobName*"[*from*][*to*]	
U"*JobName*"x	
U"*JobName*"x(*from*)(*to*)	
U"*JobName*"x[*from*][*to*]	
A	See "Retrieving Logged Alarm States" beginning on page 105.
A(*from*)(*to*)	
A[*from*][*to*]	
Ax	
Ax(*from*)(*to*)	
Ax[*from*][*to*]	
A"*JobName*"	
UA"*JobName*"(*from*)(*to*)	
A"*JobName*"[*from*][*to*]	
A"*JobName*"x	
A"*JobName*"x(*from*)(*to*)	
A"*JobName*"x[*from*][*to*]	
CURJOB	Returns the current job's name
RUNJOB"*JobName*"	Makes ***JobName*** the DT800's current job. (Of course, ***JobName*** must already exist in the DT800 — see "Program: a Holder for Jobs" on page 15.) New data created is appended to any existing data for that job.
RUNJOBONINSERT"*JobName*"	See "Startup Job" on page 115.
RUNJOBONINSERTALL"*JobName*"	
RUNJOBONRESET"*JobName*"	

Table: DT800 Job Commands (sheet 2 of 2)

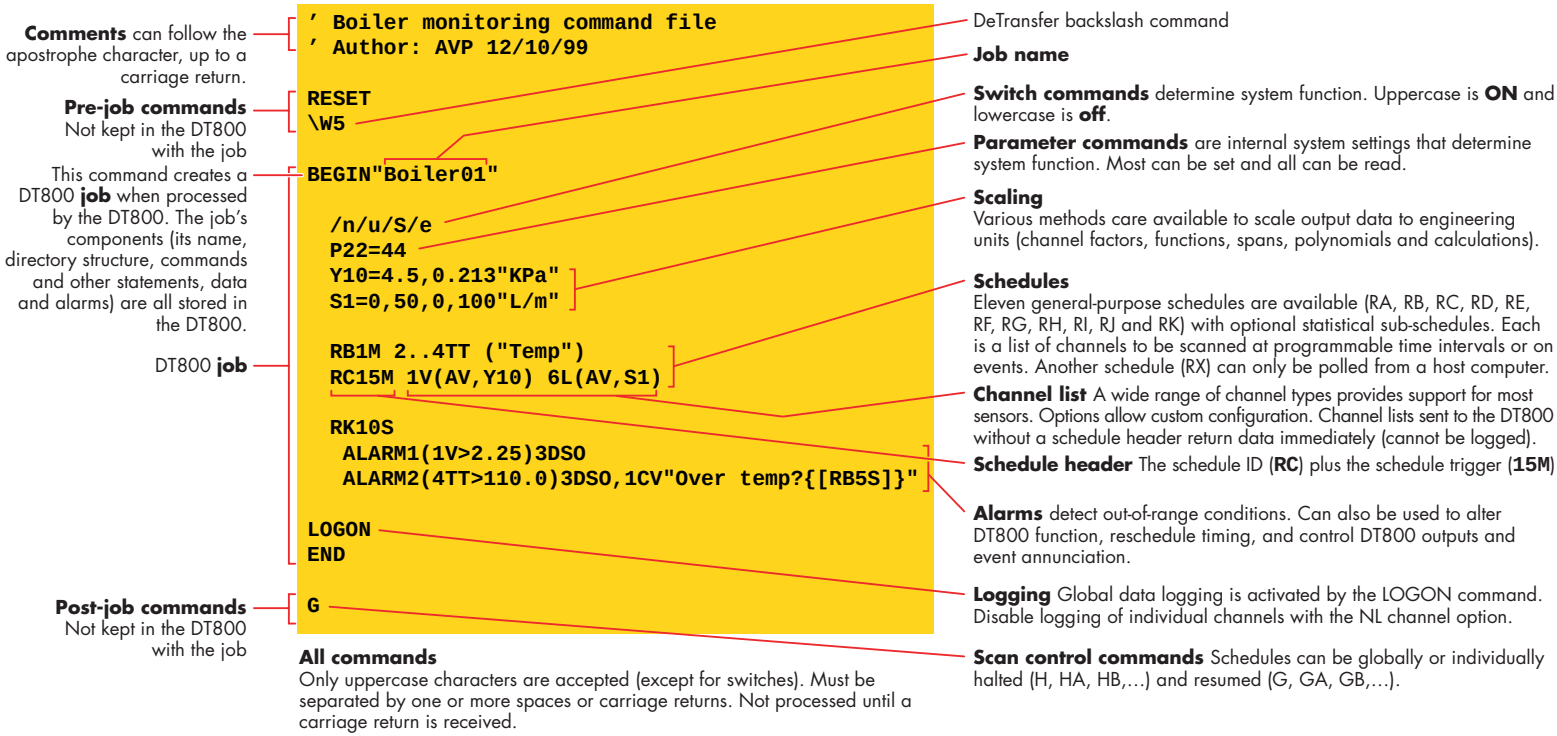


FIGURE 8 Anatomy of a sample DT800 program

Pre-Job and Post-Job Commands

See Figure 8. These are not stored in the DT800 along with the job. They are run once when you send the job to the DT800, then discarded. Only the job is kept in the DT800 for re-use later.

Deleting Jobs

To delete a job from the DT800, you must firstly delete the job's alarms and data. But before doing this, you must unlock the job (if it's locked). Therefore, the sequence for deleting a job from the DT800 is

- use the appropriate **UNLOCKJOB** command if the job is locked, then
- use the appropriate **DELALARMS** command if the job has logged alarms in the DT800, then
- use the appropriate **DELDATA** command if the job has logged data in the DT800, then
- use the appropriate **DELJOB** command to finally delete the job.

These commands are presented in the DT800 Job Commands table (page 16), and the DT800 Delete Commands — Summary table (page 185).

Scaling and Calculations

The DT800 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 "Channel Options — Scaling" on page 90.

Reducing Data

In many instances you can reduce the volume of the data recorded by taking averages, maximums, minimums, standard deviations, histograms or integrals. See "Channel Options — Statistical" on page 85.

You can also use conditional statements to define when data is to be logged. See ""Trigger While" on page 48" and "Alarm Condition" on page 98.

DIRJOBS report

```
DIRJOBS
   UNTITLED
   JOB1
   JOB2
 +*JOB3
```

+ indicates locked job
* indicates current job

Logging status (**On** or **Off**)

Number of data records

Number of alarm records

DIRJOB"*JobName*" report

```
DIRJOB"JOB2"
Job JOB2
S  SchedID  Log  Data Records (approx)  First               Last                Alarm Records (approx)  First               Last
=============================================================================================================================
A           Off                    17  08/01/2004 09:26:20  08/01/2004 09:26:36                    10  08/01/2004 09:26:20  08/01/2004 09:26:36
```

DIRJOB report

```
DIRJOB
Job JOB3 - Locked
S  SchedID  Log  Data Records (approx)  First               Last                Alarm Records (approx)  First               Last
=============================================================================================================================
A           On                     63  08/01/2004 09:26:38  08/01/2004 09:27:40
```

DIRJOB* report

```
DIRJOB*
Job UNTITLED
S  SchedID  Log  Data Records (approx)  First               Last                Alarm Records (approx)  First               Last
=============================================================================================================================
Job JOB1
S  SchedID  Log  Data Records (approx)  First               Last                Alarm Records (approx)  First               Last
=============================================================================================================================
A           Off                    31  08/01/2004 09:25:50  08/01/2004 09:26:19
Job JOB2
S  SchedID  Log  Data Records (approx)  First               Last                Alarm Records (approx)  First               Last
=============================================================================================================================
A           Off                    17  08/01/2004 09:26:20  08/01/2004 09:26:36                    10  08/01/2004 09:26:20  08/01/2004 09:26:36
Job JOB3 - Locked
S  SchedID  Log  Data Records (approx)  First               Last                Alarm Records (approx)  First               Last
=============================================================================================================================
A           On                     63  08/01/2004 09:26:38  08/01/2004 09:27:40
```

FIGURE 9 Sample DIRJOB reports

Alarms

The DT800's alarm facility is flexible and powerful. Alarms are used to warn of error conditions and to control the DT800's operation. Alarms can

- allow logical comparisons with setpoints
- control DT800 digital state outputs
- initiate execution of *dataTaker* commands
- trigger the sending of messages to the host computer.

Executing DT800 commands from an alarm can be particularly useful in modifying the DT800's programming in response to changes in input(s).

See "Alarms" beginning on page 96.

IFs

The DT800's **IF** facility allows powerful program control. See "Conditional Processing — IF… Command" on page 60.

Data Logging

The DT800 stores measurements in its internal data store and in removable PC Card memory cards. The internal memory acts as a buffer for the memory card, so that data is not lost during card changes.

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

By default, the DT800 stops logging when both its internal and card memories are full. But an overwrite mode allows continued logging, with the oldest data being overwritten by new readings — see "Overwrite Mode (/O)" on page 77 and **/O** in the Switches table on page 111.

Selective Logging

You can selectively log channels and schedules:

- For channels, use the **NL** channel option — see "Disabling Data Logging for Specific Channels" on page 76.
- For schedules, use the **LOGON*x*** command — see "LOGON and LOGOFF Commands" on page 76.

Handle With Care

Important The DT800 does everything possible to avoid data loss caused by careless use. However, it does respond to resets and **DEL...** commands instantly (see the DT800 Delete Commands — Summary table on page 185).

These erase information without question the moment you send them to the *dataTaker*, so use them with care. (See "Deleting Logged Data" on page 78.)

Retrieving Data

The DT800 can do two things with the data it measures:

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

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

Retrieving Real-Time Data

The DT800's default is to return data to a connected host computer instantaneously — that is, as it is measured. (You can override this by sending the **/r** switch to the *dataTaker*; see **/R** in the DT800 Switches table, page 111). You can store this real-time data as a file on the computer.

Data Return Modes — Real-Time Data Real-time data can be returned to the host in either free-format mode (the DT800's default) or fixed-format mode. See "Two Format Modes for Returned Data" on page 21.

Retrieving Logged Data

Data stored in a DT800's internal memory or memory card can be retrieved (returned, unloaded) by means of the Host RS-232 port, the Ethernet port, or the USB port. You can retrieve data for an individual schedule or all schedules, or for all jobs or an individual job. Here are a few of the commands you'll find useful when retrieving logged data:

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

See

- "Retrieving Logged Data" beginning on page 81
- the DT800 Retrieval Commands — Summary table on page 186.

Data Return Mode — Logged Data The DT800 always returns logged data to the host in fixed-format mode. See "Fixed-Format Mode /H" on page 21.

Analog-to-Digital Conversion

The DT800 uses a fast (100kHz) successive approximation ADC (Analog-to-Digital Converter). Many ADC sampling characteristics can be adjusted by means of

- channel options (page 70)
- parameters (page 107)
- switches (page 111).

These include calibration, settling time, sampling time and extended or multiple sampling. The default values of these characteristics are suited to the majority of sensors.

See "Optimizing for Speed" on page 23.

Examples of Things You Can Do with Channels

■ Read a channel once. For example, sending the command

```
2TT
```

to the DT800 instructs it to read channel 2 as a type T thermocouple. It returns data in the standard format

```
2TT 449.3 DegC
```

■ Read channels repeatedly. For example, sending the command

```
RA1S 2..4TT
```

to the DT800 instructs it to read channels 2, 3 and 4 as type T thermocouples (**2..4TT**) every second (**RA1S**) and return data in the standard format

```
2TT 451.5 degC
3TT 563.2 degC
4TT 487.8 degC

2TT 451.9 degC
3TT 569.8 degC
⇓
```

Important We recommend that you wire sensors to channels, calibrate them and test them using a schedule command such as this one above prior to entering your full program.

■ Change the format of the returned data. For example, sending the parameter and switch commands

```
P22=44 /n/u
```

to the DT800 instructs it to change the data separator to ASCII 44, the comma, and disable channel number and units. The returned data looks like

```
452.0,565.4,451.0
452.3,566.2,450.5
⇓
```

Memory Cards

The DT800's PC Card slot (Figure 12 on page 25) supports ATA Flash memory cards and ATA hard disk cards, which you can use

- as removable data storage in addition to the DT800's internal memory (see "Logging to Memory Card" on page 76
- as a medium for transferring logged data from the internal memory of a DT800 to a computer (see "Retrieving Logged Data — Memory Card Transfer" on page 81)
- to upgrade a DT800's operating system (see "Upgrading DT800 Firmware" on page 193)
- to load a startup job into a DT800 (see "Startup Job" on page 115).

See also

- "Storage Capacity" on page 30
- "Memory Card Commands" on page 30
- "What Happens When Memory is Full?" on page 77.

Data is stored on PC Cards in a Windows-compatible file structure — see "Directory Structure of Memory Cards" on page 79.

Format of Returned Data

The DT800 can return the following types of data to the host computer:

- data returned as it is measured — that is, real-time data
- data unloaded from the DT800's internal memory or from a memory card — that is, logged data
- data returned by the **TEST** and **STATUS** commands

The format of returned data is controlled globally by the following parameters and switches (see "Configuring the DT800" beginning on page 107" for full details):

P22	Data delimiter in free-format mode (see page 107)	Default = **32** (space character)
P24	Scan delimiter in free-format mode (see page 108)	Default = **13** (carriage return character)
P31	Date format — see "Date" on page 68	Default = **1** (DD/MM/YY) or **2** (MM/DD/YY)
P32	Maximum number of significant digits (0 to 9)	Default = **5**
P33	Defines a fixed field width for output data (variable)	Default = **0** (off)
P38	Decimal point locator character for floating-point numbers	Default = **46** (full stop character)
P39	Time format — see "Time" on page 68	Default = **0** (hh:mm:ss.sss)
P40	Time separator character	Default = **58** (colon character)
/H	Fixed-format mode — see "Two Format Modes for Returned Data" on page 21	Default = off
/U	Include Units text appended to the data	Default = on
/N	Include channel Number and type before data	Default = on
/L	Include *dataTaker* serial number before scan data	Default = off
/C	Include Channel type (/C) or number only (/c)	Default = on
/D	Include scan Date at beginning of returned data	Default = off
/T	Include scan Time at beginning of returned data	Default = off
/I	Include schedule ID	Default = off

Character Pairs — Carriage Return + Line Feed

The DT800's default is to automatically add a carriage return character (**CR**, ASCII 13) and a line feed character (**LF**, ASCII 10) to the end of appropriate chunks of information it returns to the host computer. These return-and-line-feed pairs (**CRLF**) make data and other returned information easy to read on the host screen or a printout.

Two Format Modes for Returned Data

The DT800 has two modes for the format of data and other information it returns to the host computer:

- free-format mode (enabled by **/h**)
- fixed-format mode (enabled by **/H**)

Logged data is always returned to the host computer in fixed format, but real-time data can be returned in either free format or fixed format.

Free-Format Mode /h

Also known as "unformatted mode". This is the DT800's default mode for real-time data return (enabled by the **/h** switch command), in which data and other information is returned to the host computer in a verbose (descriptive, conversational) style suitable for on-screen display and printing. For example, when the DT800 is in free-format mode, the schedule command

```
RA5S 1V 3PT385 1C("Widgets")
```

returns each data item in the form

```
1V 2.490 mV
3PT385 395.0 degC
Widgets 3498 Counts
```

to the host computer screen. The switches listed in the table above default to **/U/N/C**, and parameters P22 and P24 are not used as delimiters while units text is enabled (**/U**).

User-Definable In free-format mode, you can use format-related parameter and switch commands to alter the format of returned data and other information to suit your requirements.

Fixed-Format Mode /H

Also known as "formatted mode". Recommended for those writing drivers to interface host software with the DT800 (that is, advanced users only). Logged data is always returned to the host in fixed-format mode.

In fixed-format mode (enabled by the **/H** switch command), the following parameters and switches are forced to the states shown in order to ensure a predictable, repeatable, comprehensive format for returning data ready to be imported into spreadsheets and other data analysis software:

States Forced by /H	Effect on Returned Data
P22=44	Each measurement is separated by a comma (**,**).
P24=13	CR (and LF) is applied to the end of each scan's data.
P38=46	Decimal point character = period (**.**)
/c	Channel type does not appear in returned data.
/u	Units text does not appear in returned data.
/n	Channel number does not appear in returned data.
/e	Echo (automatic return of sent commands) is turned off.
/r	Real-time data return is turned off.

For example, when the DT800 is in fixed-format mode, the schedule command

RA5S 3PT385

returns data as shown in Figure 10.

See also Figure 65 on page 84.

The parameters and switches listed above are restored to their previous values when the DT800 receives **/h** (sets free-format mode).

Not User-Definable In fixed-format mode, you cannot change the format of returned data and other returned information: format-related parameter and switch commands have no effect.

Numeric Format

You can set the numeric format of free-format-mode returned data for individual channels by the following channel options (see "Output Data Format" in the DT800 Channel Options table — page 75):

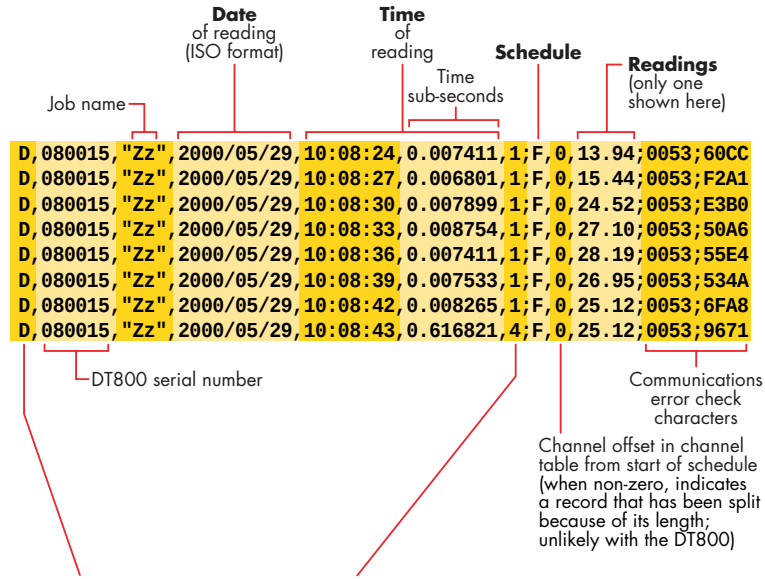
FF*n*	Fixed-point format, ***n*** = number of decimal places (0 to 7)
FE*n*	Exponential format, ***n*** = number of significant digits (0 to 7)
FM*n*	Mixed FF or FE formats. Uses FE format if exponent is less than –4 or greater than ***n***.

For example:

Default	FF1	FE3	FM1	FM2
23.456	23.5	2.346e1	23.5	23.46
–0.025	–0.0	–2.542e–2	–0.0	–0.03
1034.6	1034.6	1.035e3	1e3	1034.64

The default format depends on the channel type returning the data (see the DT800 Channel Types table beginning on page 63, especially the Resolution column). Formatting options are not applied to the 99999.9 error data code (see "Error Messages" on page 197).

Fixed Field Width Parameter 33 (page 108) allows returned data to be in fixed fields. All data is placed into fields of the same width (defined by P33), by space-padding to the left. If the field width is not sufficient, least significant characters are truncated from the right. Fixed fields are useful when returned data is to be tabulated, or forwarded to software with a simple string parser.



Record Type	Record Index
A **A**larm record	Alarm number
B **B**urst record	**0** = pre trigger **1** = trigger **2** = post trigger **4** = discontinuity
C Program **C**hange record	**0** = change of card **1** = change of program
D Returned **D**ata record	**0** = real-time data **1** = logged data **2** = Unused **3** = end of data return (end of unload for logged data) **4** = data discontinuity record (caused by sending one of the LOGOFF or Halt commands, or by changing jobs) **5** = end of schedule during unload
E **E**rror record	Error number
I **I**nformation record	Information number
P **P**arameter record	Parameter number
S **S**tatus record	Status message number
T **T**est record	Test message number
W Pass**W**ord query record	Password message number

FIGURE 10 Typical logged data records returned in fixed-format mode

Guidelines for Successful Data Gathering

The Procedure

Data acquisition and 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 and number of channels.
- Determine sensor output scaling.
- Determine how data is to be processed and reported.
- Decide on the sample frequency — minimize redundancy.
- Calculate the volume of data to be collected.
- Decide on the method of data recovery and archiving.
- Consider the power consumption.

Having defined the task, you then connect sensors and program the DT800.

Sampling Multiple Channels

To attain good performance over many channels, the high speed input multiplexer can interleave different channels within a line cycle period. In this way, 5 to 10 different channels can be sampled every 16 or 20ms, all with good mains hum rejection.

Ground Loops

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.

DT800 Solves Ground-Loop Problems

There are three general areas of any measurement system that can give rise to ground loops (described in greater detail in "Grounds, Ground Loops and Isolation" on page 151 — see Figure 95). The analog section isolation built in to the DT800 removes the likelihood of ground-loop problems between sensors and the *dataTaker*.

Of course, other ground-loop combinations are possible (sensor-to-computer, for example), but the DT800's isolation blocks most of these as well.

Other Ground-Loop Solutions

Many ground-loop problems can also be overcome by

- using independent inputs instead of shared-terminal inputs to remove the effects of sensor-to-sensor loops, and/or
- connecting all grounds in a measurement system to a single common point (although this is not always practical).

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 (see Figure 135 on page 183).

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 DT800 is designed to reject mains noise (hum; see "Normal Mode Sampling" on page 51). For best noise rejection, set the DT800's parameter 11 to your local mains frequency, 50Hz or 60Hz — see **P11** on page 107.

You can force the DT800 to load this parameter setting every time it restarts by using a **PROFILE** command — see "User Defaults and Startup" on page 11, and the **PARAMETERS** section of the DT800 Parameters table (page 113).

Self-Heating of Sensors

Sensors that need excitation power to be read are heated by power dissipation. This can be particularly acute with temperature sensors and some sensitive bridges. Minimize error by minimising the excitation power (see the **V**, **I** and **P** excitation channel options on page 72).

Optimizing for Speed

Although the DT800's "scanning engine" — its ADC — is capable of scanning at 100kHz, the actual rate achievable depends on several factors: ADC settings, channel type, number of channels, data storage and return options, and data formatting.

See "Getting Optimal Speed from Your DT800" on page 188 for more information.

Part B — Hardware

Inputs and Outputs

Terminals, ports, connectors and sockets

DT800 Front Panel

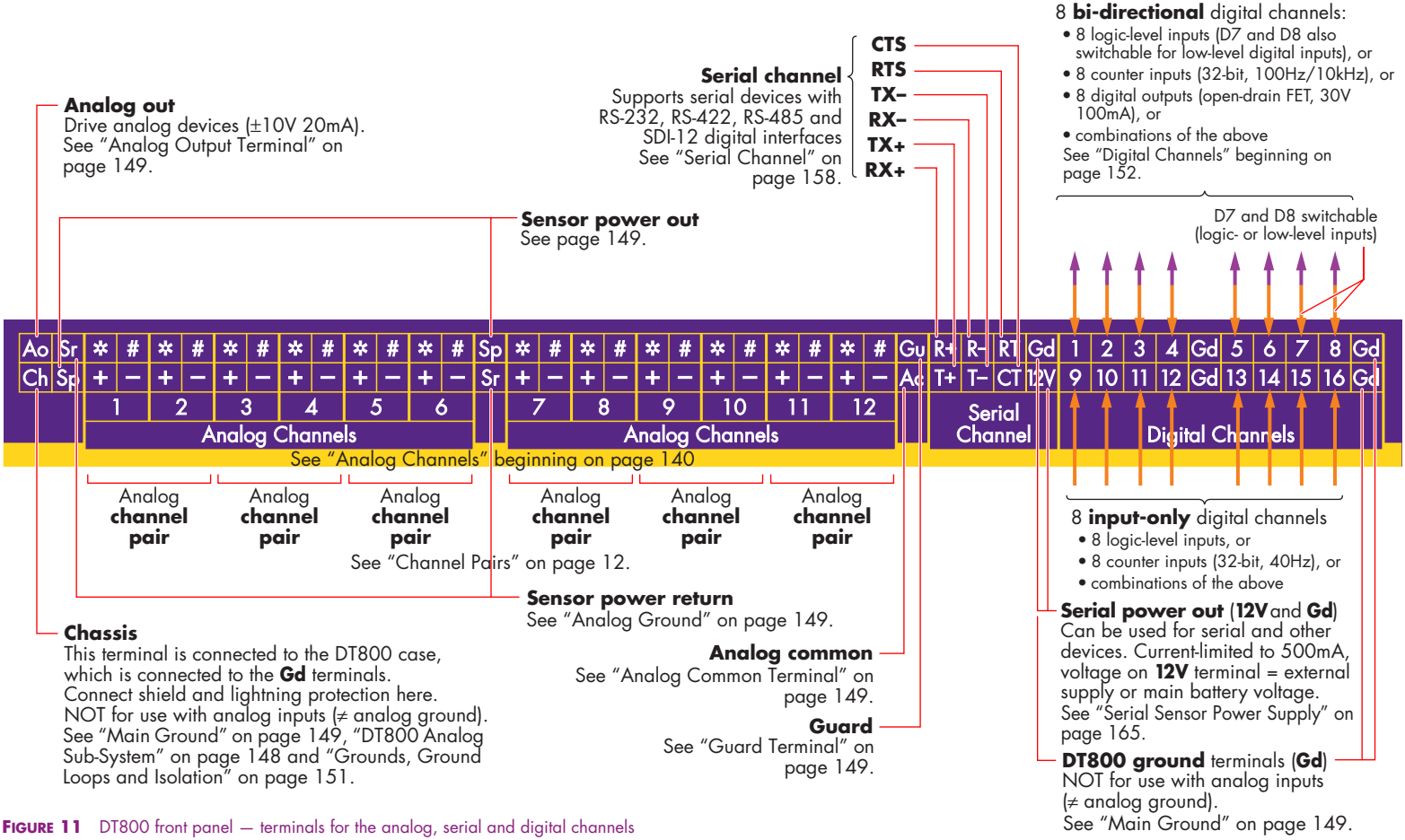
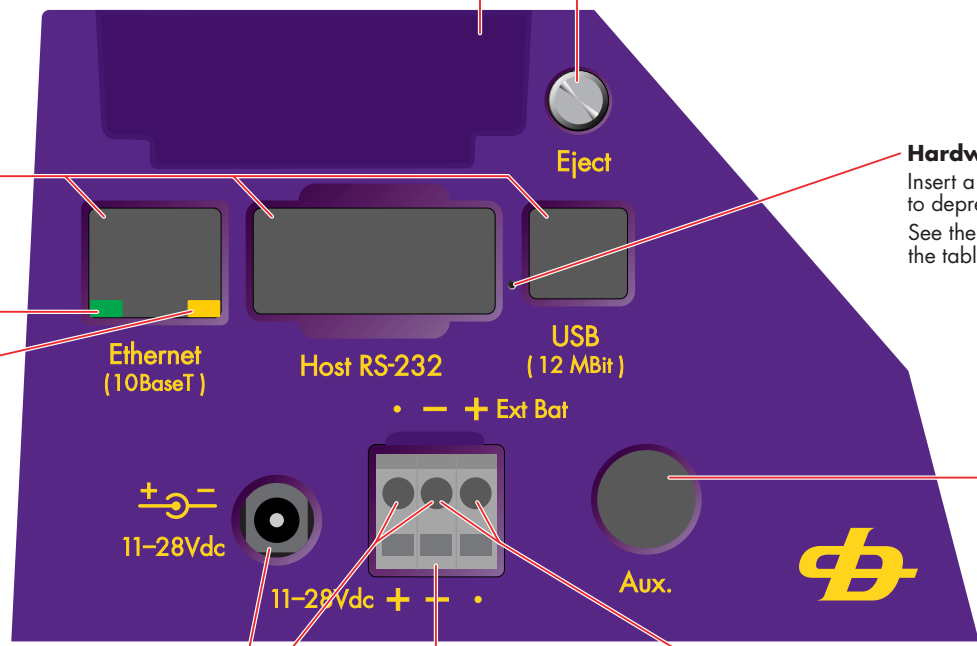


Figure 11 DT800 front panel — terminals for the analog, serial and digital channels

DT800 Side Panel



PC Card slot
Supports Type I, II or III PC Cards (ATA Flash memory cards, ATA hard disk cards and modem cards)

PC Card eject button

Communications ports
See page 124.

Ethernet **transmit activity** LED (green)
Ethernet **network activity** LED (amber)

Hardware reset hole
Insert a straightened paperclip to depress the reset button.
See the Hardware Reset row in the table on page 118.

Auxiliary port
(for future expansion)

External power (11–28Vdc) inputs — coaxial socket or cage-clamp terminals are provided.
Either input powers the DT800 and charges the main internal battery.

Removable terminal block
Pry out with flat-bladed screwdriver or coin

External battery (12V) terminals
Powers the DT800 but does not charge the main internal battery.
DT800 does not charge the external battery through these terminals.

FIGURE 12 DT800 side panel — memory card slot, communications ports and power connectors

Connecting Sensors to Channel Terminals

The 84 terminals on the front panel of the DT800 are of the "cage-clamp" type. To connect a sensor wire to one of these:

1. Strip approximately 8mm of the insulation from the end of the wire.
2. Firmly press the tip of one of the cage-clamp tools (supplied with your *dataTaker*) into the terminal's tool entry slot at a 45° angle (Figure 13 and Figure 14).
   This opens the spring clamp in the terminal's wire entry slot. You can remove your hand from the cage-clamp tool while you insert the wire — the tool stays in place, wedging the clamp open.
3. Insert the bared wire into the wire entry slot and pull the cage-clamp tool from its slot.
   The wire is now firmly clamped in the terminal.

Note A double-pronged cage-clamp tool is provided to facilitate the connection of pairs of wires to two adjacent terminals at the same time.

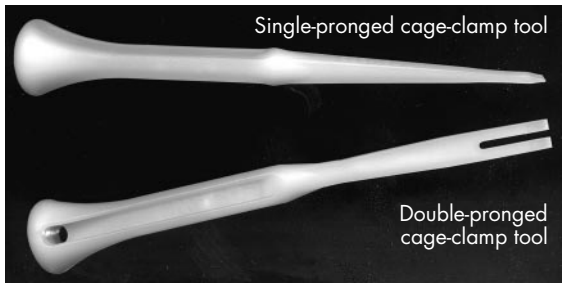


Figure 13 Cage-clamp tools

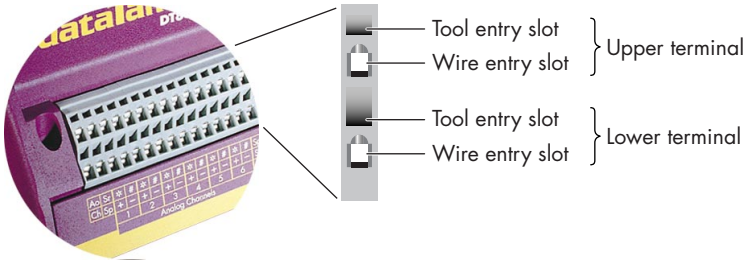


Figure 14 Two of the DT800's cage-clamp terminals on the front panel

Connecting to the Removable External Power Terminals

Use a cage-clamp tool or flat-bladed screwdriver as described in "Connecting Sensors to Channel Terminals".

Note The terminal block can be unplugged by prying it out with a flat-bladed screwdriver or coin.

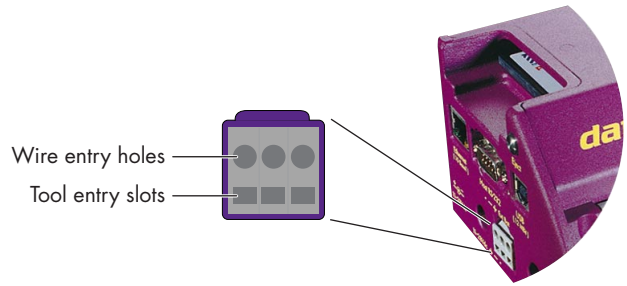


Figure 15 The DT800's external power cage-clamp terminals

Terminal Layout Sheet

Figure 16 is a blank slide-in layout sheet for the DT800's sensor terminal block. Having made one or more of these, you can

- write on or color-code the sheet to make paper records of your terminal wiring
- use the sheet to plan wiring layouts.

The sheet is designed to be kept with the DT800 — slid in behind its sensor terminals. It has two folds, which make three panels. Before inserting it behind the terminal block, you can

- leave it unfolded — the upper two panels are visible when inserted (useful if the terminal screenprinting below the terminal block is obscured by wires, and to read your wiring layout)
- fold the lower panel behind — leaves just the upper panel (your wiring layout) visible when inserted
- fold the upper panel behind — leaves just the middle panel visible when inserted (useful if the terminal screenprinting below the terminal block is obscured by wires)
- fold both the upper and lower panel behind — leaves just a finger tab visible when inserted (for storing the sheet with the DT800 — turn the sheet over so that the name and date show).

To make a layout sheet:

1. Photocopy this page (or print it if you have the Adobe Acrobat PDF file of this manual).
2. Fold the sheet at the two fold lines ( ). Be sure to do this before the next step.
3. Cut out the sheet ( ).

IMPORTANT: MAKE FOLDS FIRST, THEN CUT

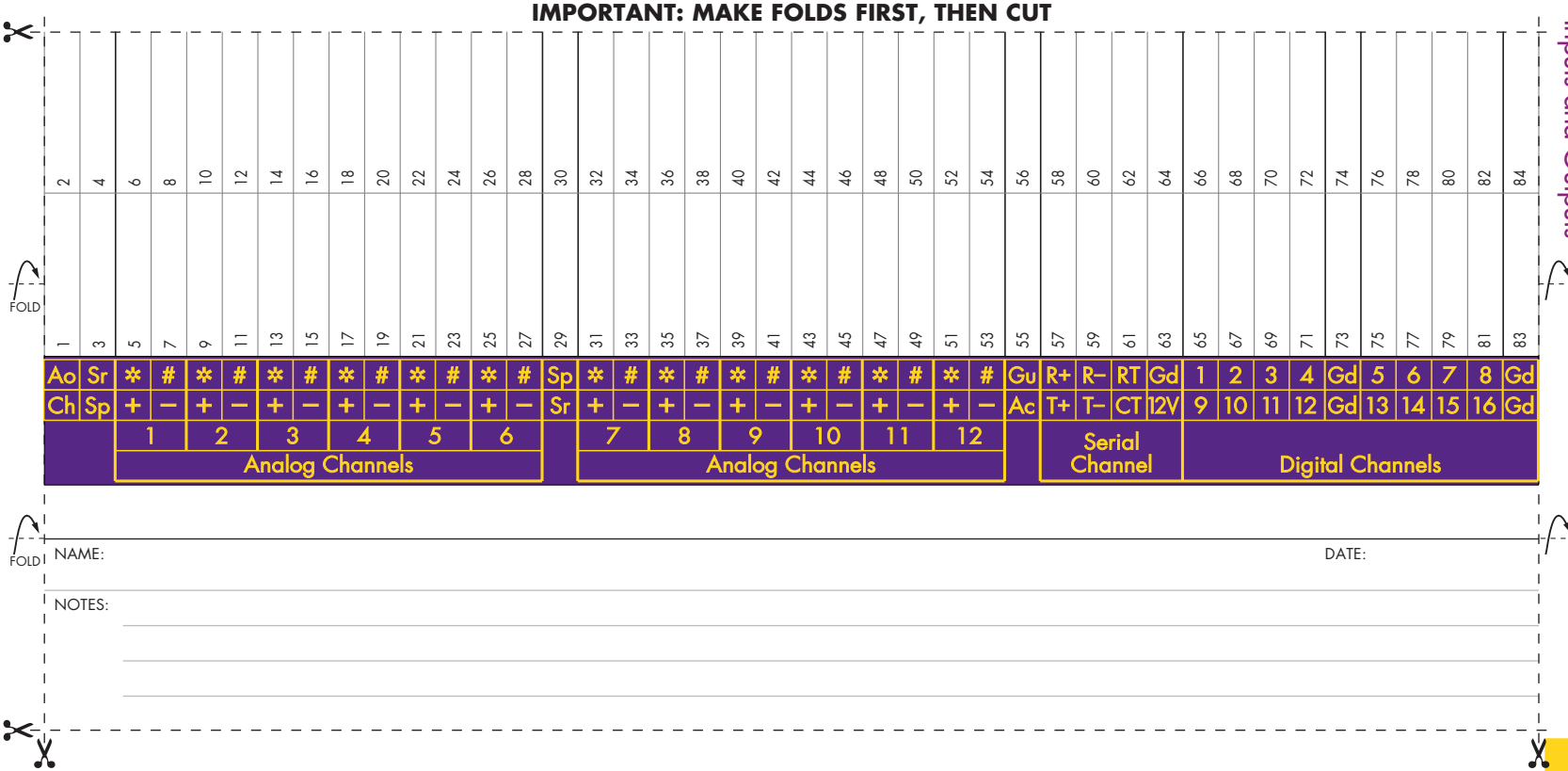
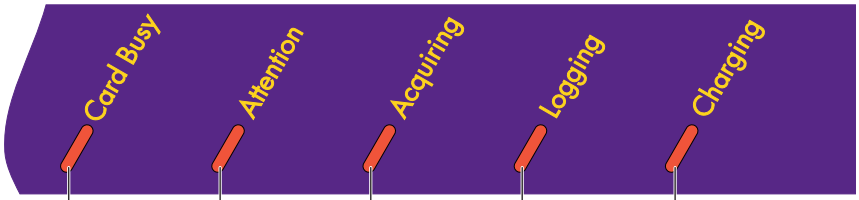


FIGURE 16 Terminal layout sheet — blank

INDICATORS

The DT800 has indicator LEDs (light-emitting diodes) on its front panel, and an internal buzzer.

LEDs



- Lights when the DT800 is charging its main internal battery from an external supply
- Flashes whenever the DT800 stores a measurement — that is, whenever the DT800 acquires data (makes a measurement) and is in LOGON mode
- Flashes
  - whenever the DT800 makes a measurement
  - every three seconds (the DT800's "heartbeat" — see step 3 below)
- Lights — and stays lit — when any of the following occurs:
  - the DT800's data storage area is full (in stop-when-full logging mode: **/o**)
  - a significant event (for example, a power failure) has occurred — check the event log
  - the DT800's internal temperature specification has been exceeded (that is, internal temperature <–20°C or >60°C) — check the event log
  - one of the DT800's internal maintenance channels has gone out of range — send the TEST command (look for **FAIL**ed lines in the test report), or check the event log
  - you send the `SATTN` command to the DT800 (clear by sending the `CATTN` command) — see "Attention LED Commands" on page 29
  - you send the `2WARN=1` channel type to the DT800 (clear by sending `2WARN=0`) — see page 66
- Indicates memory card activity

FIGURE 17 DT800 front panel — LED indicators

LED Sequence on Startup

Here's what the DT800 LEDs do during a restart:

1. All LEDs flash rapidly four times.
2. The Attention LED lights momentarily to indicate that the DT800 has restarted.
3. The Acquiring LED flashes every three seconds. This is the DT800's internal housekeeping "heartbeat". (The DT800 carries out internal housekeeping operations every three seconds when awake.)
   For special circumstances that require less-frequent or more-frequent internal checking you can alter this housekeeping period (see P61 on page 110). But, in general, we recommend that you don't change it.
4. If the DT800 is externally powered, the Charging LED lights.

See also "LEDs and Messages After a Reset" on page 119, and "Upgrading DT800 Firmware" on page 193).

No LEDs?

No LED activity does not necessarily mean that the DT800 has a fault or that it is unpowered.

It may simply mean that the DT800 is asleep. To test this, you can wake the DT800 by doing things such as communicating with it or inserting a PC Card.

See also "Low-Power Operation" on page 43.

Attention LED Commands

Turn the Attention LED on or off using these commands:

SATTN	Set the Attention LED (LED on)
CATTN	Clear the Attention LED (LED off)

To send the **SATTN** and **CATTN** commands to the DT800, include them as ***actionProcesses*** in the **DO** command— see "Unconditional Processing — DO... Command" on page 61.

Compare with the **2WARN** channel type (page 66).

Buzzer

You use the **1WARN** channel type (see page 66) to turn the DT800's internal buzzer on and off. To do this, include the following in your schedule:

- **1WARN=1** to activate the buzzer
- **1WARN=0** to deactivate the buzzer

MEMORY

The DT800 contains internal Flash and SRAM memory, as well as the option to insert a removable memory card. Each of the main circuit boards also contains its own EEPROM memory. See Figure 18.

Storage Capacity

The DT800 stores logged data and alarms in CRC-16 integrity-guaranteed format, which allows 60,000 readings per megabyte. Therefore

- the DT800's 2MB internal memory (B: drive RAM disk) can store 120,000 readings
- an inserted memory card can store an additional 60,000 readings per megabyte.

See also "Data Storage Capacity — Readings/MB" on page 78.

Memory Card Commands

`CARDCLEAR` `FORMAT"A:"`	These commands reformat the inserted memory card and create the minimum directory and file structure. **Warning** These commands delete all data from the card.

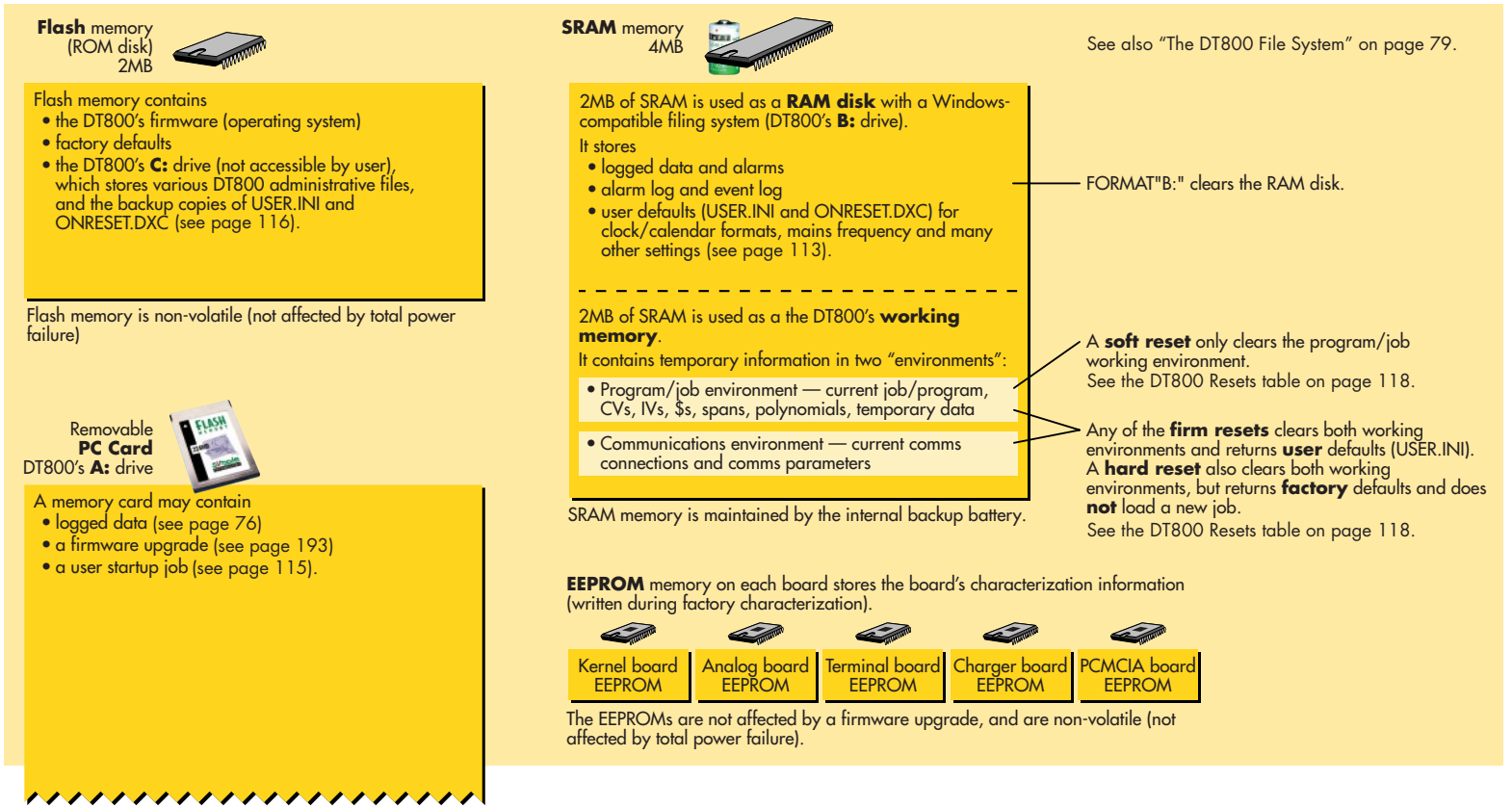


FIGURE 18 The DT800's memory — Flash, SRAM, PC Card (memory card) and individual EEPROMs

Inside the DT800

Opening the DT800 Case

To open the DT800's case:

1 Using a flat-bladed screwdriver, undo the two top-cover screws (one each side of the terminal block on the front panel).

Figure 19 Opening the DT800 case — 1

2 Grasp under each side with your fingertips and use your thumbs to press down on the top of the terminal block. Lift hard enough with your fingertips to unplug the full-width connector strips inside.

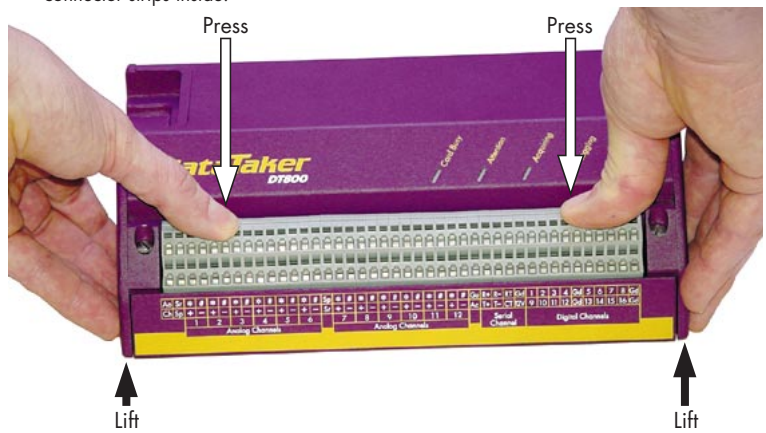


Figure 20 Opening the DT800 case — 2

3 Swing the front of the top cover up and back.

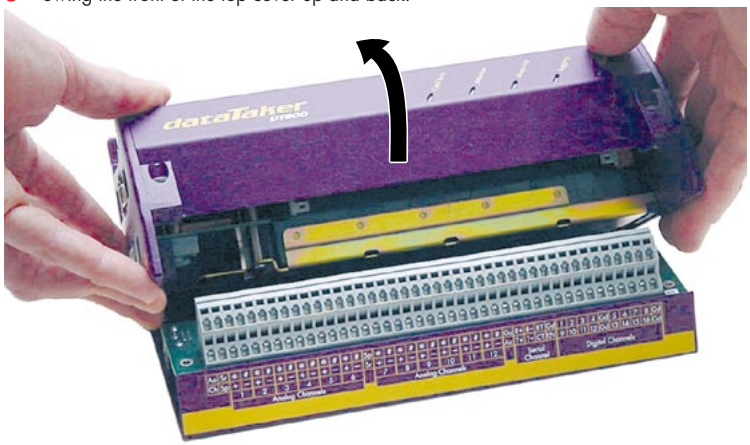


FIGURE 21 Opening the DT800 case — 3

4 Lift the top of the DT800 away from its base.

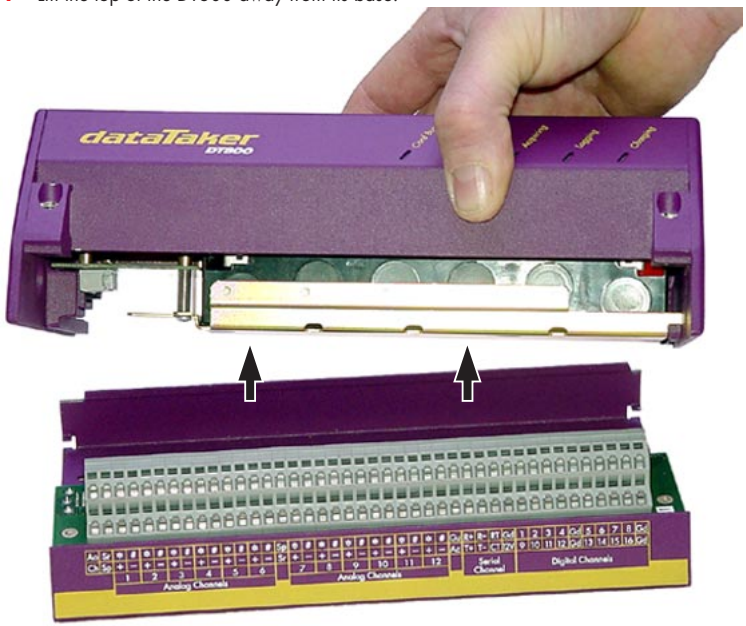


FIGURE 22 Opening the DT800 case — 4

5 Go to "Accessing the Internal Batteries" on page 33, "Mounting the DT800" on page 35, or "Closing the DT800 Case" on page 38.

Accessing the Internal Batteries

Warning If you remove both internal batteries from the DT800 at the same time, all SRAM memory contents (data, alarms, alarm log, event log, program, clock/calendar and configuration settings) will be lost.

To gain access to the DT800's main battery and memory-backup battery:

1 After opening the DT800 case (see page 31), remove the two battery-cover screws and lift the battery cover-plate out. Note the two prongs on the end of the cover-plate that locate the plate into the end casting of the DT800.

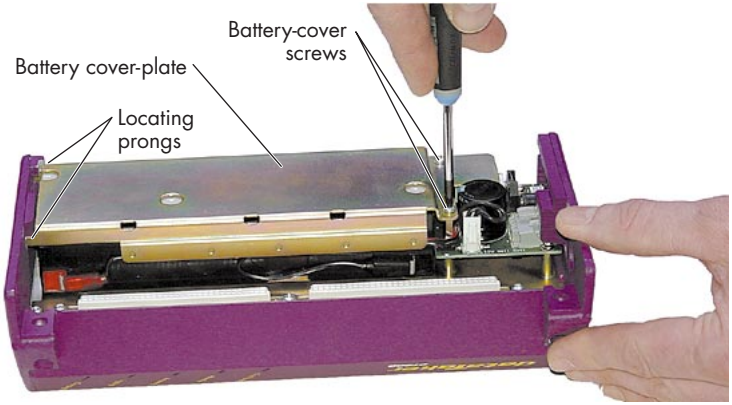


FIGURE 23 Removing the battery cover-plate

2 You can now lift the internal main battery out of the DT800 and disconnect it at the spade terminals or the cable connector, or…

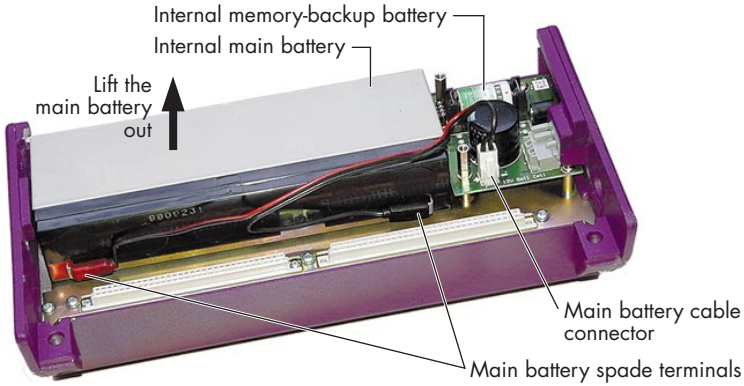


FIGURE 24 The DT800's two internal batteries

3 …remove the internal memory-backup battery from its holder.
For replacing the memory-backup battery in its holder, the polarity is marked on the inside of the holder and on the circuit board at the end of the holder.

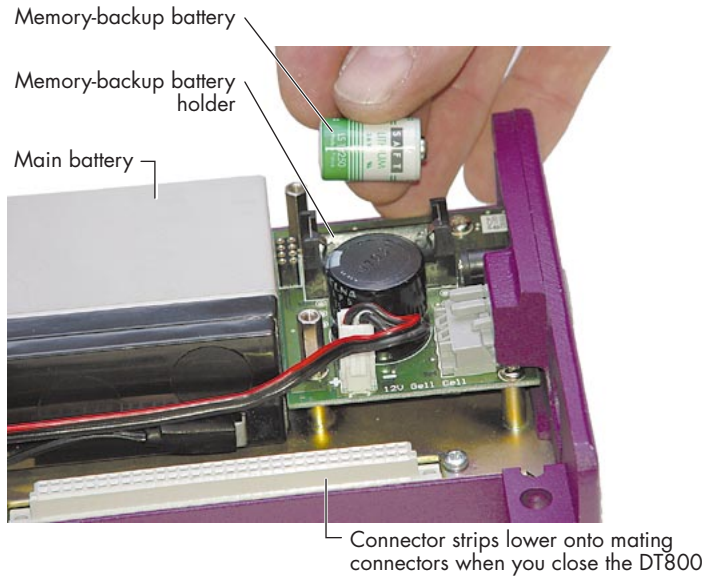


FIGURE 25 Removing/replacing the internal memory-backup battery

4 When replacing the battery cover-plate, make sure that the main battery's cable is well out of the way of the two connector strips by tucking the cable up under the edge of the battery cover-plate. If necessary, use a screwdriver to do this.

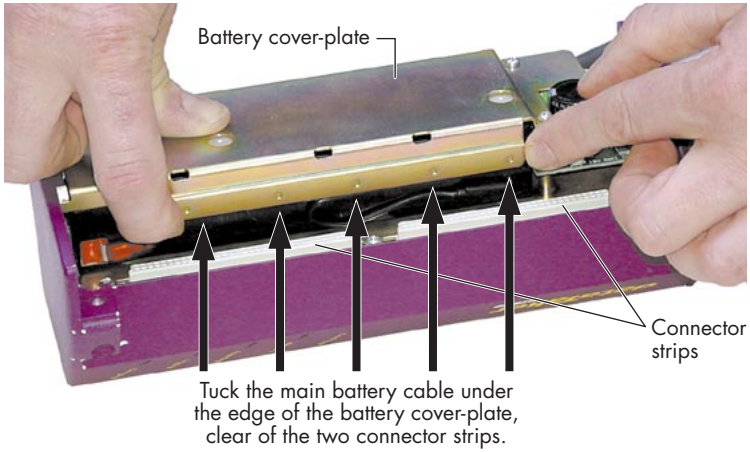


FIGURE 26 Fitting the main battery cable under the edge of the battery cover-plate

5 Go to "Mounting the DT800" on page 35, or "Closing the DT800 Case" on page 38.

Mounting the DT800

Dimensions, Clearances

Figure 27 shows dimensions you need to be aware of when screw-mounting or rail-mounting the DT800. In particular, note the clearance required for connectors and memory card insertion and removal.

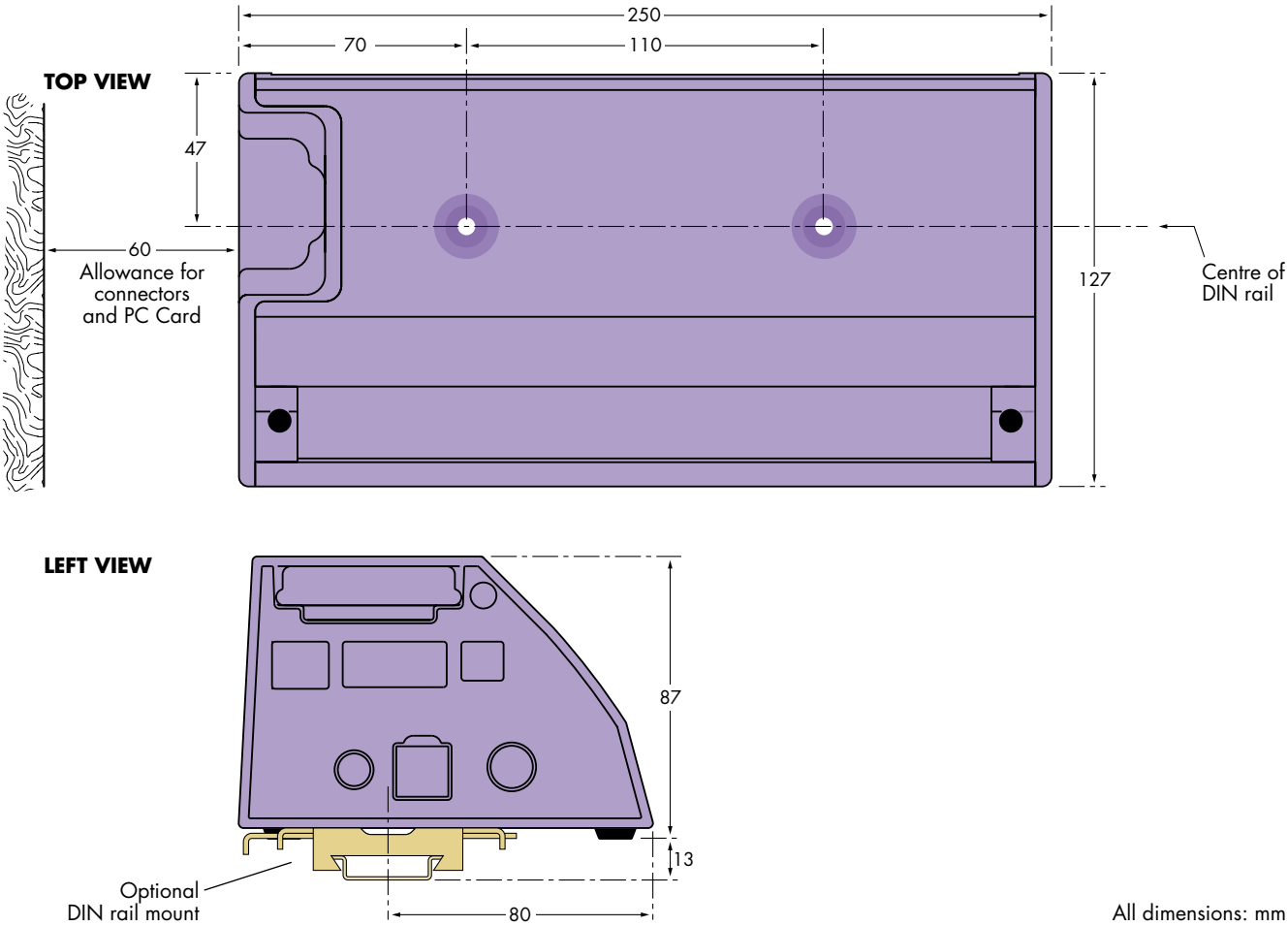


FIGURE 27 DT800 mounting dimensions

Screw Mounting

To mount the DT800 using screws:

1 After opening the DT800 case (see page 31), screw its base to the mounting surface (wall, benchtop, cabinet,...) through the two recessed mounting holes.

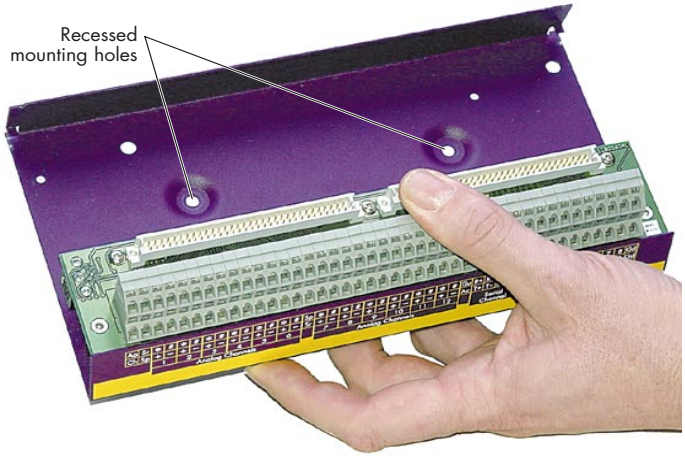


FIGURE 28 Mounting the DT800 using screws

2 Close the case as described on page 38.

DIN Rail Mounting

To mount the DT800 using DIN rail (Figure 29):

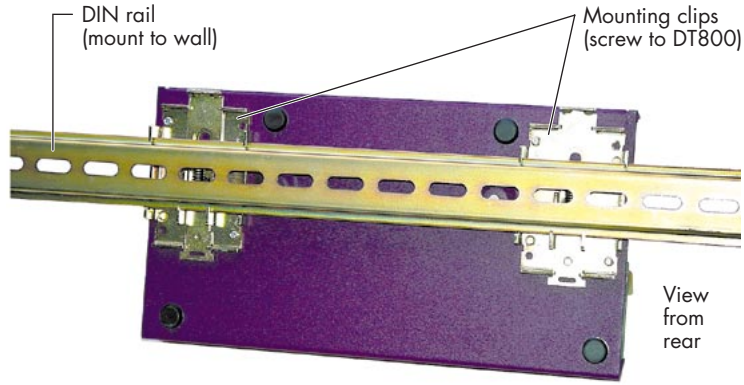


FIGURE 29 DIN rail and mounting clips in position

1 After opening the case (see page 31), screw two DIN rail mounting clips to the rear of the DT800.
Two screw holes for each clip are provided in the DT800's base. The finger-press of each clip's spring-loaded latch points upwards.

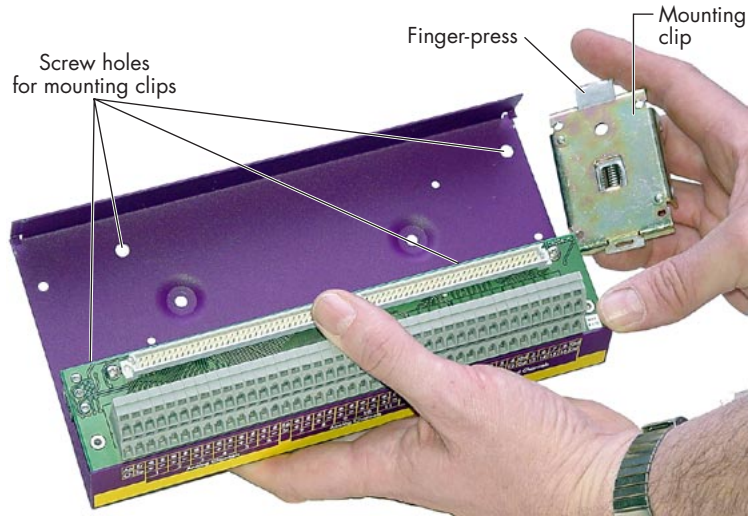


FIGURE 30 Fitting the DIN rail mounting clips

2 Close the DT800 case as described on page 38.

3 Fit the DIN rail to the mounting surface and clip the DT800 to the rail.
When attaching or removing the DT800, depress each clip's spring-loaded latch that protrudes above the case.

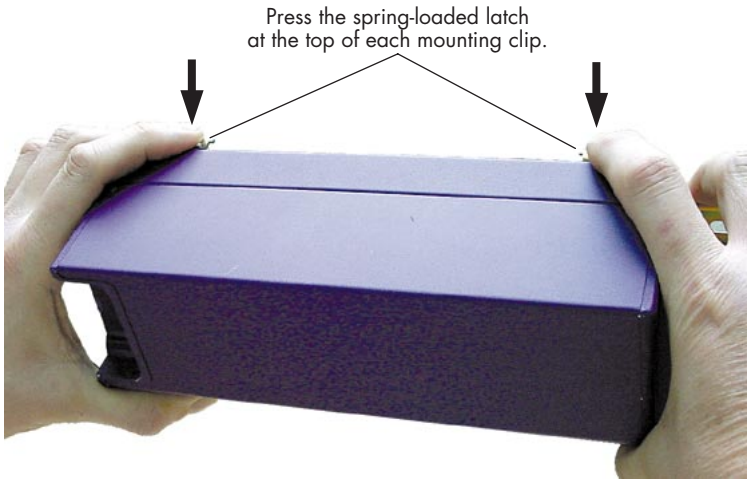


FIGURE 31 Clipping the DT800 to the rail

Closing the DT800 Case

To close the DT800's case:

1 Lie the rear of the top of the DT800 along the rear of the base (Figure 32). Align the top directly over the base and "hinge" the two backs together (Figure 33).

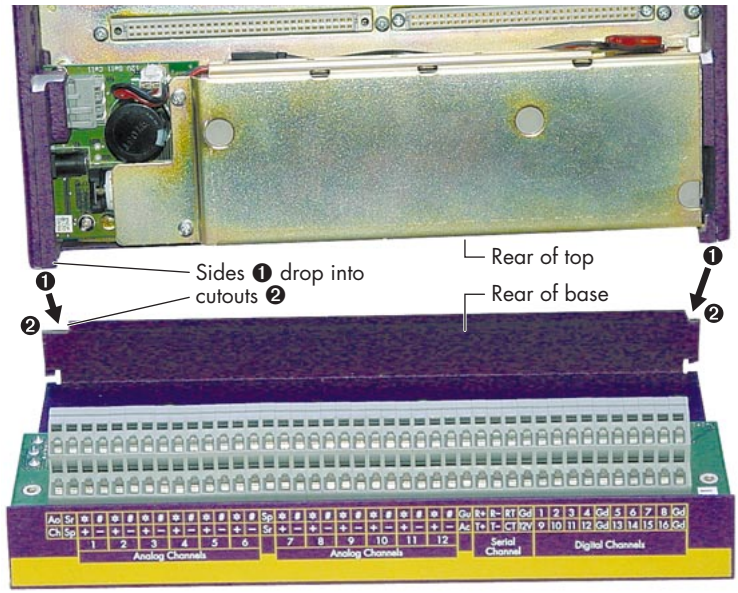


FIGURE 32 Aligning — 1

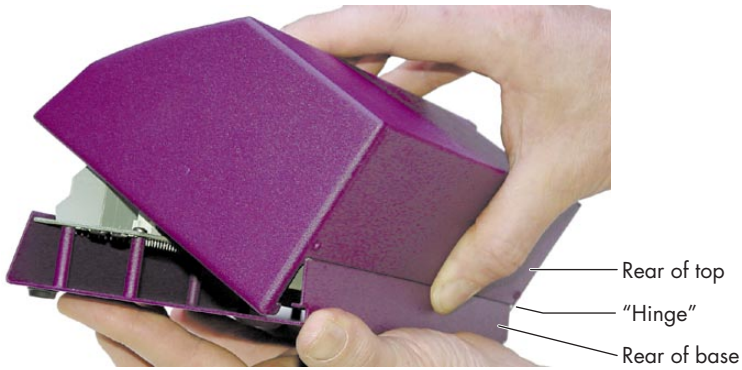


FIGURE 33 Aligning — 2

2 Swing the front of the DT800 down onto the base (Figure 34). Press firmly to mate the connector strips inside (Figure 35).
If the top of the DT800 won't press home, check that the main battery's cable is not trapped between the connector strips — refer to Figure 26 on page 34.

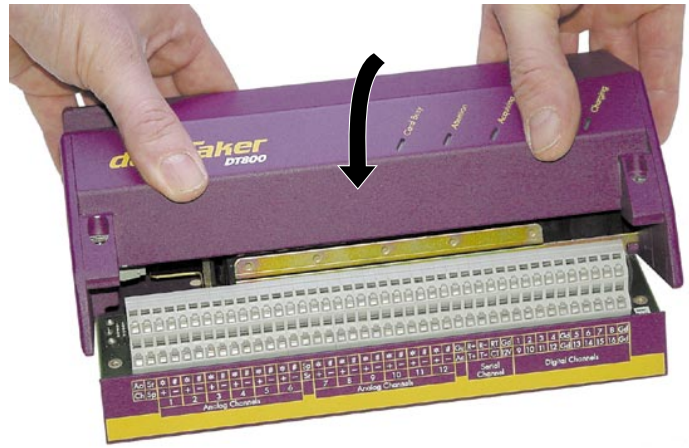


FIGURE 34 Closing — 1

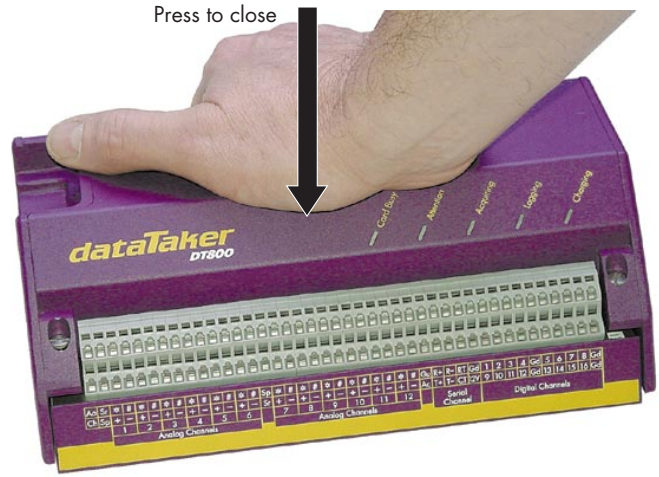


FIGURE 35 Closing — 2

3 Using a flat-bladed screwdriver, tighten the two top-cover screws.

FIGURE 36 Closing — 3

Part C — Power
Powering the DT800

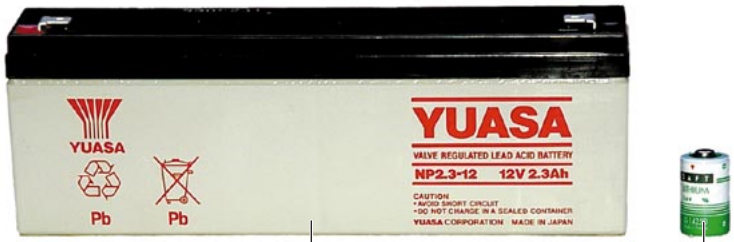
You can supply power to the DT800 from internal and external sources:

Internal See "Internal Power (Main Battery)" below.	12V lead acid gel cell battery supplied with the *dataTaker*	
External See "Main Battery Life" below.	"Standard" supply **11–28Vdc** inputs	A DC mains adapter (plug pack) or other "unlimited" DC power source — that is, a source with no special energy conservation or optimization requirements
	"Low-drain" supply **Ext Bat** terminals (12Vdc)	External batteries, solar panels, vehicle power supplies and other sources for which energy conservation may be critical — that is, applications requiring minimum battery drain by the *dataTaker* so that the longest possible life is obtained from the power source

Internal Power (Main Battery)

The DT800 is fitted with an internal 12V 2.2Ah sealed lead-acid gel-cell battery (dimensions 178mm x 60mm x 33mm) — Figure 37. It's known as the DT800's "main" battery to distinguish it from the DT800's other internal battery, the "memory-backup" battery.

The main battery is completely maintenance-free and rechargeable, being automatically charged by the *dataTaker*'s inbuilt battery charger whenever an external power supply is connected to either of the *dataTaker*'s **11–28Vdc** inputs (Figure 12 on page 25). The battery is fitted with spade terminals.



Internal main battery
12V lead-acid gel-cell

Internal memory-backup battery
3.6V lithium
(see page 42)

Figure 37 The DT800's two internal batteries

Main Battery is Disconnected for Shipping

The DT800 is shipped with its main battery disconnected. Therefore

- if you intend to use the internal main battery as the *dataTaker*'s power source, connect the main battery as described in "Inside the DT800" beginning on page 31
- if you intend to power the *dataTaker* from an external source, you don't need to connect its internal main battery. **HOWEVER...**

Recommendation No matter how you intend to power the DT800, we recommend you connect its internal main battery. By doing this, you guarantee uninterrupted data acquisition and logging because the internal main battery is then always available to continue powering the *dataTaker* if the primary/external supply is accidentally disconnected or fails. The topic "Inside the DT800" beginning on page 31 explains how to connect the internal main battery.

In addition, the main internal battery is a gel-cell type: if you let a gel-cell remain flat for any length of time, its service life will be significantly reduced.

Main Battery Life

The life of the DT800's internal main battery depends on

- the scan interval
- the number of analog channels being scanned
- the number of digital channels being scanned
- the number of alarms
- excitation power drawn by sensors
- the complexity of any calculations
- communications activity.

See "Always Trying to Sleep" and "Extending Battery Life" on page 43.

Counters and Main Battery Life

See "Counter Rollover Rate" on page 157.

External Power

Three Options

The three options for powering a DT800 externally are shown in Figure 38.

Your DT800 is supplied with a DC mains adapter, ready to be powered from a wall outlet.

Note The DT800 can not charge an external battery from any of its terminals.

Maximum Reliability

In situations where the reliability of external power to the DT800 is crucial, we recommend that you use the removable external power terminal block (option 2 in Figure 38) instead of the coaxial socket. The cage-clamp terminals of the removable block are exceptionally reliable.

See Figure 12 (page 25) and Figure 15 (page 26).

Solar Charging

You can charge the DT800's internal main battery from a 12V solar panel connected to the *dataTaker*'s standard power inputs (option 1 or option 2 in Figure 38). The *dataTaker* provides current and voltage limiting to protect both the panel and battery.

The size of the solar panel required depends on the hours of full sunlight that can be expected. As a general rule, only one day in seven should be regarded as a "charge day", and the charge must be able to fully replenish the batteries on that one day. The solar panel rating is calculated as follows:

$$\text{Panel rating} = \frac{I_w}{T_w \times \eta} \text{ Amps (in full sunlight)}$$

where

I_w	is the Ah per week consumed by the *dataTaker*
T_w	is the hours per week of full sunlight
η	is the efficiency; a combination of battery charge absorption and the cosine effect — typical value 0.65

Sending **P15=1** ensures that the DT800 sleeps whenever possible to conserve power.

See also Figure 12 on page 25.

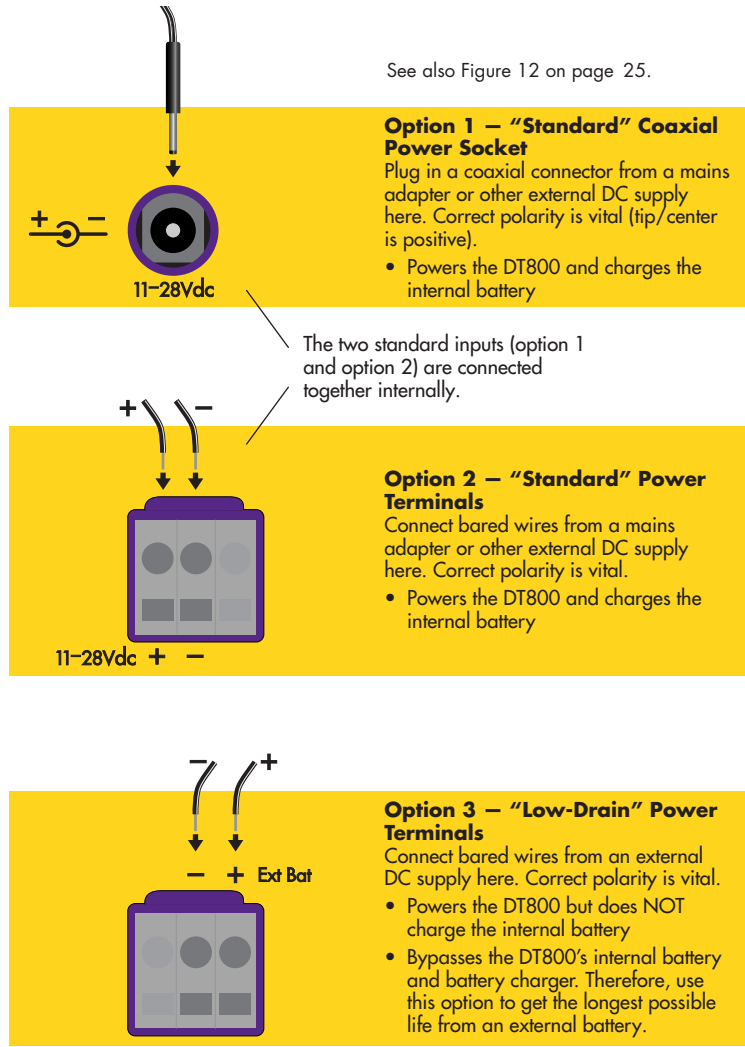


FIGURE 38 The DT800's external power inputs

Internal Memory-Backup Battery

In addition to the internal main battery, the DT800 contains a small, cylindrical, lithium "memory-backup" battery. Figures 24 and 25 show its location inside the DT800.

3.6V lithium
$^1/_2$ AA size

FIGURE 39 The DT800's memory-backup battery

The memory-backup battery ensures that

- your data
- the DT800's clock/calendar
- the DT800's primary configuration settings (mains frequency P11, date format P31 and time format P39.

are not lost if power to the *dataTaker* is interrupted. The memory-backup battery can maintain this information for at least 12 months if necessary.

Just like the backup battery in your computer, you'll need to replace the DT800's memory-backup battery every five years or so if you want to maintain this safeguard capability. The topic "Inside the DT800" beginning on page 31 explains how to replace the internal memory-backup battery.

The memory-backup battery is a $^1/_2$AA 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).

Self-Testing the Memory-Backup Battery

You can use the DT800 to test its memory-backup battery by

- sending a **TEST** command — see Lithium Battery Voltage (line 7) in "TEST Commands" on page 120
- including the **VLITH** channel type in a schedule (or alarm) — see **VLITH** in the DT800 Channel Types table (page 67).

Getting Maximum Life from the Memory-Backup battery

The memory-backup battery is not activated until you connect the main battery to the *dataTaker*. This means that the memory-backup battery is effectively "on-the-shelf" (it has a 10-year shelf life) until the main battery is connected. This is one of the reasons the *dataTaker* is shipped with its main battery disconnected; to prolong the overall life of the memory-backup battery. (The other reason is safety during shipping.)

Once you activate the memory-backup battery, you can de-activate it at a later time, and thereby return it to an on-the-shelf situation of zero current drain, by following the simple procedure described in "Internal Memory-Backup Battery During DT800 Storage" in the next topic.

Battery Guidelines for Long-Term Storage

To look after your DT800's batteries if it is to be out-of-use for longer than a month or two, do the following:

Internal Main Battery During DT800 Storage

Charge the main battery periodically so that it never goes flat. Eight hours charge every three to six months is safe; a battery in good condition need only be charged every twelve months.

You can use the DT800 to do this by applying power to one of its External Power inputs (see Figure 12) with the main battery connected inside, or you can remove the battery from the *dataTaker* (see "Inside the DT800" on page 31) and charge it from a suitable external charger.

Internal Memory-Backup Battery During DT800 Storage

Remove the memory-backup battery from the *dataTaker* (described in "Inside the DT800" on page 31).

This, of course, returns the memory-backup battery to a zero-drain/shelf-life situation (with an overall shelf life of at least 10 years). Then, as long as the main internal battery is disconnected, you can refit the memory-backup battery in its holder inside the *dataTaker*, ready to be automatically reactivated when you reconnect the main battery (refer to "Getting Maximum Life from the Memory-Backup battery" on page 42).

If you leave an activated memory-backup battery in the DT800, the memory-backup battery has a life of

- approximately five years if the main battery is present and powering the DT800
- approximately one year if no other power is present.

Low-Power Operation

When supplied from its internal main 12V battery, the DT800 has two power-related modes of operation — wake mode and sleep (low-power) mode:

- In wake mode, the DT800 is fully active and draws 150mA (typical) or 400mA (maximum) from the battery.
- In sleep mode, only the "bare essentials" remain active and current drain is reduced to approximately 300µA.

Once asleep, the DT800 only wakes, for example, when

- a scheduled scan becomes due
- an immediate scan is sent
- a memory card is inserted or removed
- communication arrives at the Host RS-232 port (see "Comms Wakes the DT800" on page 128).

Exceptions

If the DT800 is externally-powered or connected to an Ethernet network, it never sleeps. There are also other exceptions. See "No-Sleep Conditions" below.

Always Trying to Sleep

Sleep Conditions

For maximum life when the DT800 is powered only by its internal main battery, the DT800 goes to sleep when ALL of the following conditions exist:

- No external channel measurement is scheduled for the next four seconds[2].
- No communication has arrived at the DT800's Host RS-232 port within the last 30 seconds[3].
- The DT800's Host RS-232 port RI (Ring Indicator) line has not been asserted within the last 30 seconds[3].
- No reset of the DT800 has occurred within the last 30 seconds[3].

You can override the default operation listed above by

- setting **P15=2** (see "Controlling Sleep" below)
- rapid scanning (see "Extending Battery Life" below)
- setting P55 so that specific schedules do not wake the DT800 (see page 110).

No-Sleep Conditions

The DT800 is designed to not go to sleep when any of the following conditions exist:

- When the DT800 is externally-powered. That is, when power is provided to either of the standard power inputs (options 1 and 2 in Figure 38 on page 41). You can override this — see "Controlling Sleep" next.
  When externally-powered, the DT800 draws 50mA to 400mA (depends on the state of charge of the internal main battery) from the external supply in addition to the 150mA (typical) or 400mA (maximum) required for normal operation. This occurs even if the DT800 is forced to sleep by setting **P15=1**.
- When the DT800's Ethernet port is connected to an Ethernet network, or directly to a computer's Ethernet port. (But connecting Ethernet to a sleeping DT800 does not wake it.)
- When the DT800 is unloading data.
- When a modem connected to the DT800's Host RS-232 port is in the process of establishing a call or has a call in progress.

[2] Four seconds is the DT800's default — see **P3** on page 107.
[3] 30 seconds is the DT800's default — see **P17** on page 108.

Controlling Sleep

Use parameter 15 (page 108) to control the DT800's sleep:

P15=	Sleep Entry Condition	
0	Auto-sleep: • Sleep when not busy only if powered from internal main battery • Never sleep if externally-powered P15=0 is the DT800's default.	See P17 (page 108) and P55 (page 110).
1	Force sleep — that is, sleep when not busy regardless of how powered	
2	Force normal operation — that is, keep the DT800 awake (never enters sleep mode)	

Extending Battery Life

Therefore, to extend battery life, do not sample channels more frequently than your data-gathering requires. You can also save power by minimizing comms activity (set the reporting switch to **/r**), and using digital counter channels D7 and D8 (see "Counter Rollover Rate" on page 157).

Low-Power Programs

Sleep Program

You may find this framework useful when designing low-power programs. Reset the DT800 before sending this program:

```
BEGIN

  P15=1'Sleep if not busy
  P17=5'Go to sleep quickly
  /u/n 'Disable channel ID and units
  S1=0,100,0,1.000"%RH"'Define spans, etc. here

  RS15M'Scan slowly for statistical schedules
  RA1H'Especially for reporting schedules
  1V("Humidity",S1,AV) 'Define channels
  2PT385("Air temp.",4W,AV,=1CV)
  IF(1CV>25)[LOGON]
  IF(1CV<20)[LOGOFF]

END
```

Conserve Serial Channel Power

You can turn the Serial Channel's transceiver on only when you need it. For example:

```
RA10M 1SSPORT=1 DELAY=500 1SERIAL(xyz) 1SSPORT=0
```

where

1SSPORT=1	turns the Serial Channel's transceiver on (see page 164)
DELAY=500	delays schedule execution for 500ms (see page 64) to allow transceiver to stabilize
1SERIAL(xyz)	is your Serial Channel command (see page 158)
1SSPORT=0	turns the Serial Channel's transceiver off again

Part D — Schedules

Schedule Concepts

Tell the DT800 what to do and when to do it

What are Schedules?

Schedules (full name: **schedule commands**) are the workhorses of the DT800. They are the underlying structures that you use to manage the repetitive **processes** of the DT800 such as

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

Figure 40 shows the components of a typical schedule.

The DT800 supports the following schedules:

11 general-purpose schedules	**Report schedules**	Most commonly-used of the schedules
3 special-purpose schedules	**Polled schedule**	Triggered by command from a host computer
	Immediate schedule	For running processes immediately, once
	Statistical sub-schedule	For collecting primary data for statistical summaries of input channel data

Schedule ID

Each schedule has a unique identifier, which is used when programming the DT800. The schedule IDs are summarized in the following table:

	Schedule ID	Quantity
Report schedules See page 46	RA, RB, RC, RD, RE, RF, RG, RH, RI, RJ, RK	11 available
Polled schedule See page 49	RX	1 available
Immediate report schedule See page 49	No schedule ID	You may create any number, but only one can be sent to the DT800 at a time.
Statistical sub-schedule See page 49	RS	1 available, but is applied to one or more of the other report schedules

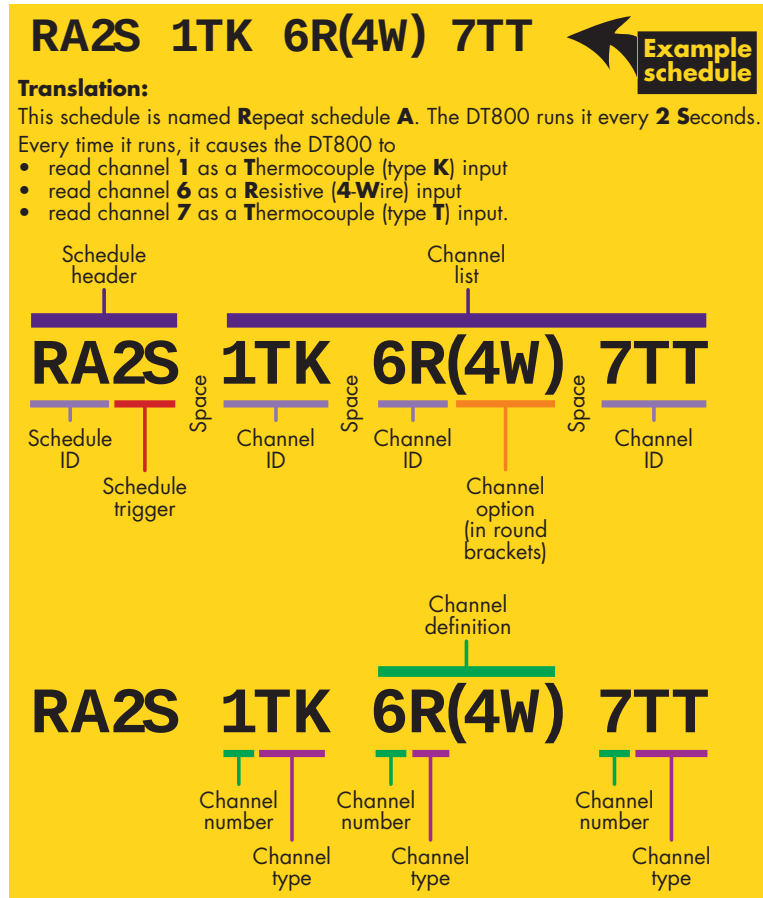


Figure 40 Components of a typical schedule command

UM-0068-A2

Schedule Trigger

All schedules have a **trigger**, which defines when the schedule is to execute the processes assigned to it. Here are the DT800's schedule triggers:

Schedule Trigger		
An interval of **time**	See "Trigger on Time Interval" on page 46.	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" on page 48.
An **external event**	See "Trigger on External Event" on page 46.	
An **internal event** (that is, an event generated within the DT800)	See "Trigger on Internal Event" on page 47.	
A **poll command** from a host computer	See "Trigger on Schedule-Specific Poll Command" on page 47.	

Channel List

Most often you'll create schedules that instruct the DT800 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 you create these schedules, you group the channel details (their IDs and optional instructions) together in a **channel list** within the schedule. Figure 40 (page 44) 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. This is shown in Figure 40 and in "Example — Channel List" below.

The DT800 processes the channels in a channel list from left to right.

"Specifying Channel Details (Channel Lists)" beginning on page 62 covers channel lists in detail.

Example — Channel List

The channel list

```
1V 3R 5..7I 9TK("Boiler Temp") 3DSO=1
```

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..7I` — read the sequence of analog input channels 5 through 7 (inclusive) as currents
- `9TK("Boiler Temp")` — read analog input channel 9 as a type K thermocouple and assign it the unique name **Boiler Temp**
- `3DSO=1` — set digital state output channel 3 ON (low/active — see "Digital State Output Channel Types" on page 154)

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..7I 9TK("Boiler Temp") 3DSO=1
```

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;BURST(100,1000) 1V 3R
```

specifies report schedule A as follows:

- `RA` — schedule ID
- `10M` — trigger (run the schedule every 10 minutes)
- `;BURST(100,1000) 1V 3R` — process list (a burst command and a channel list)
- `1V 3R` — channel list

Groups of Schedules — Jobs

A DT800 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, you can 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 you can choose to assign a schedule to a single channel, such as the DT800's Serial Channel.

See also "Jobs" beginning on page 15.

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

The DT800 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

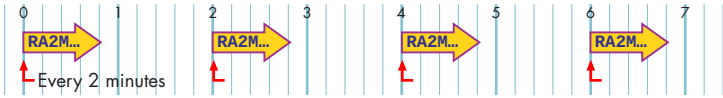


FIGURE 41 Typical interval-triggered schedule

Report schedules can be triggered at regular intervals of time, determined by the DT800's realtime clock. Intervals can be an integer number of seconds, minutes, hours or days:

***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**	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" on page 58), and subsequently runs every multiple of the interval thereafter. If the interval is not an even multiple of 24 hours, the DT800 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 DT800 to run **R**eport schedule **A** every 5 seconds (**5S**).

The schedule header

RG10M

instructs the DT800 to run **R**eport schedule **G** every 10 minutes (**10M**).

Trigger on External Event

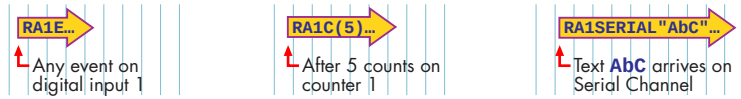


FIGURE 42 Typical externally-triggered schedules

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**:

***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 digital counter channel ***n***
1SERIAL"*text*"	Trigger on the arrival of characters (from an external serial device) at the DT800'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" beginning on page 158.

where

n	is a digital channel number
m..n	is a sequence of digital channel numbers (see "Digital Channels" beginning on page 152)
text	is a string of characters arriving at the DT800's Serial Channel terminals from an external serial device

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 while counter 2 is assigned a value of 15.

Examples — Trigger on Digital Channel Event

The schedule header

`RC1E`

instructs the DT800 to run **R**eport schedule **C** on every transition of digital input 1 (**1E**).

The report schedule trigger

`RA3+E`

instructs the DT800 to run **R**eport schedule **A** whenever digital input channel 3 receives a low to high (positive/rising) transition.

Examples — Trigger on Serial Channel Event

The schedule header

`RB1SERIAL"Pasta8zZ"`

instructs the DT800 to run **R**eport schedule **B** on the arrival of the specific character sequence **Pasta8zZ** at the DT800's Serial Channel terminals (**1SERIAL"Pasta8zZ"**).

The schedule header

`RG1SERIAL""`

instructs the DT800 to run **R**eport schedule **G** on the arrival of any character at the DT800's Serial Channel terminals (**1SERIAL""**).

Trigger on Internal Event

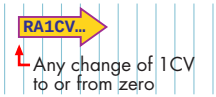


FIGURE 43 Typical internally-triggered (CV-triggered) schedule

Report schedules can also be triggered by internal events, which you must specify to the DT800 as channel variables (CVs) changing value:

***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)" on page 92.

Examples — Trigger on Internal Event

The schedule header

`RK6CV`

instructs the DT800 to run **R**eport schedule **K** upon any change of channel variable 6 to or from zero (**6CV**). 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

`RD3CV`

instructs the DT800 to run **R**eport schedule **D** whenever the value of channel variable 3 changes to 0 or from 0 (**3CV**).

The schedule header

`RK12+CV`

instructs the DT800 to run **R**eport schedule **K** whenever the value of channel variable 12 changes from 0 to any value (**12+CV**).

Trigger on Schedule-Specific Poll Command

FIGURE 44 Typical schedule-specific poll-triggered schedule

Instead of a time or event trigger, you can apply the **poll trigger** (**X**) 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). Schedules that can be polled in this way have the schedule IDs **RAX**, **RBX**,…**RKX**), and the poll commands that trigger them are **XA**, **XB**,…**XK**.

The poll command is issued

- by a host computer, or
- by an alarm action (see "Example — Alarm Action Processes: Using an Alarm to Poll a Schedule" on page 101).

See also "RxX Schedule and Digital State Outputs" on page 155.

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 DT800 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 DT800.

Using Poll Commands with Standard Report Schedules

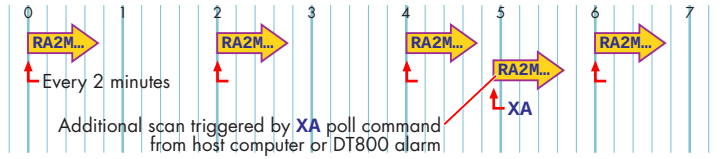


FIGURE 45 A time-triggered or event-triggered schedule can also be triggered by its poll command

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

A report schedule's trigger can be enabled or disabled by an external condition (Figure 46). 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 DT800's digital input channels (*n*DS), or
- internal conditions specified to the DT800 as states of channel variables:

:*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 DT800 to run Report 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 DT800 to run Report schedule **D** every second (**1S**) while digital input 4 is low (**:4~W**).

The schedule header

`RK2H:9W`

instructs the DT800 to run Report schedule **K** every two hours (**2H**) while digital input 9 is high (**:9W**).

The schedule header

`RC5M:12CV`

instructs the DT800 to run Report schedule **C** every 5 minutes (**5M**) while channel variable 12 is not zero (**:12CV**).

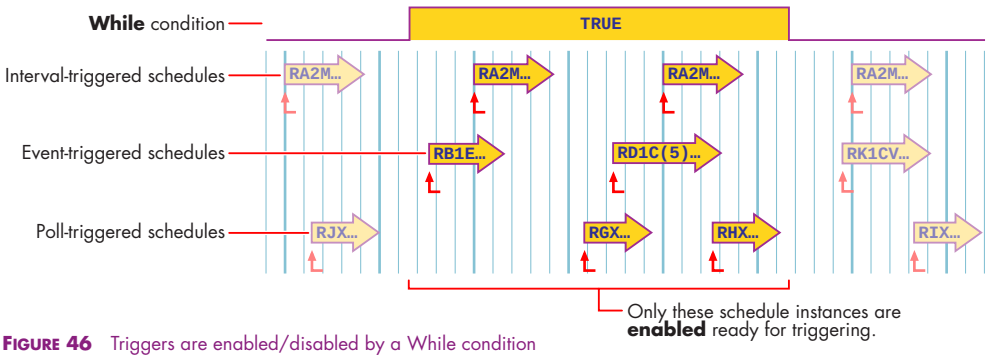


FIGURE 46 Triggers are enabled/disabled by a While condition

The schedule header

`RF6..8E:5W`

instructs the DT800 to run Report 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 47 Continuous schedule

You can create report schedules that run continuously. These schedules start scanning as soon as they are received by the DT800 (they are not activated by a trigger), and run until you stop them (by sending a halt command or resetting the DT800, for example).

The speed of **continuous schedules** depends on the mode of sampling used — see "Normal Mode Sampling" (page 51)and "Fast Mode Sampling" (page 52).

You define a continuous schedule simply by omitting the trigger from a report schedule.

Example — Continuous Schedule

Sending

`RA 1TK 6R(3W) 7TT`

causes the DT800 to scan channels 1, 6 and 7 continuously. Notice that this schedule has no trigger; for example, no **1S** after **RA**.

Schedule Modifiers

You can modify the way basic report schedules function. DT800 **schedule modifiers** do the following:

- Extend the type of data acquired by the report schedule — see "Statistical Report Schedules" on page 49.
- Change the underlying sample rate (data acquisition rate) — see "Fast Mode Sampling" on page 52, and "Burst Mode Sampling" on page 53.

Special-Purpose Report Schedules

Polled Report Schedule (RX)

FIGURE 48 The DT800's solitary polled schedule

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

- by a host computer connected to the DT800 during data acquisition, or
- by an alarm action (see "Example — Alarm Action Processes: Using an Alarm to Poll a Schedule" on page 101).

The DT800 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) sent from the host computer or from an alarm. The polled schedule does not accept any of the report schedule triggers.

Channels, calculations and alarms included in a polled schedule are processed, reported and/or logged once every time the DT800 receives an **X** poll command.

Example — Polled Report Schedule

The schedule

```
RX 1V 2V
```

runs once every time the DT800 receives an **X** character.

Immediate Report Schedules

FIGURE 49 Typical immediate schedule

Instead of scanning according to time or event triggers, **immediate schedules** run immediately — and once only — when they are received by the DT800.

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 DT800 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 6R(3W) 7TT
```

causes the DT800 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 DT800, give an immediate schedule time to execute before issuing a following **BEGIN** command, otherwise the immediate schedule's data may not be returned. You can ensure this in DeTransfer 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 page 108) can overcome this by ensuring a return character is placed after each reading.

Re-Running an Immediate Schedule

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

Statistical Report Schedules

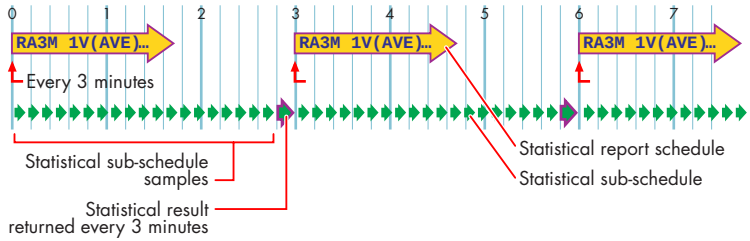


FIGURE 50 Concepts — statistical report schedule and statistical sub-schedule

A report schedule can instruct the DT800 to return statistical information (average, SD, max., min.,...) for one or more channels. The DT800 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" below.

To return statistical data, you include — in any report schedule — a statistical channel option for the specific input channels, calculations, and so on that you want to be scanned statistically. 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.
- You don't 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" below).

For details of the statistical channel options available, see the Statistical category of the DT800 Channel Options table (page 74), or "Channel Options — Statistical" on page 85. "Triggering and Schedule Order" (page 59) is also relevant to the statistical sub-schedule.

Redefining the Statistical Sub-Schedule's Trigger

You can alter the statistical sub-schedule's trigger from its default of one second.

You define the statistical sub-schedule's trigger in the same way as for report schedules (see "Schedule Trigger" on page 45"), by using the **RS** schedule ID and sending an **RS...** schedule command to the DT800. 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 DT800 to collect specified statistical data
RS10S	every 10 seconds
RS30M	every 30 minutes
RS1-E	on each 1 to 0 transition of digital input 1
RS	continuously

Statistical Sub-Schedule Halt/Go

You can halt the statistical sub-schedule by sending the **HS** command, and start it again by sending the **GS** command.

Important Because statistical sampling of channels stops the moment you send the **HS** command, 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" on page 59.

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 (see "Multiple Reports" on page 70). For example, the channel list

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

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

Statistical Sampling Error Message

If no statistical data has been scanned before being reported, then the an error `E53-no statistical samples` is returned (see the "DT800 Error Messages" table beginning on page 197), and data is set to 99999.9.

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**) and includes statistical sampling. It 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 (**RS10S 2TT(AV)(MX)**).

Sampling Modes

The DT800 has three sampling modes — **normal mode**, **fast mode** and **burst mode** (introduced in "Three Modes" on page 10). The following table gives you an idea of the differences between the sampling modes by comparing how quickly the DT800 can take readings for one type of DT800 input, voltage measurement with zero correction (**V** channel type). Resolution and noise rejection are also compared.

Mode	Typical Maximum Sampling Speed in Hz (samples/second) for V Channel Type	Resolution	Noise Rejection
Normal	40Hz for one **V** channel 7Hz for each of 10 **V** channels sampled simultaneously	18 bits (best)	50dB (best)
Fast	200Hz for one **V** channel 72Hz for each of 10 **V** channels sampled simultaneously	15 bits	Reduced
Burst	25kHz for one **V** channel 1.5kHz for each of 10 **V** channels sampled simultaneously	13 bits	Reduced

Table: Sampling Modes Comparison — **V** Channel Type

Notice that as speed increases, resolution and noise rejection decrease. Of course, lower resolution and/or noise rejection is not necessarily detrimental — for example, 13-bit resolution can be more-than-adequate in many applications.

The sampling speeds above are for voltage inputs (**V** channel type). Sampling speeds are faster for voltage-not-corrected inputs (**VNC** channel type), and slower for more-complex inputs such as thermocouples and bridges — see "Number of Fundamental Samples per Reading" on page 63.

Sampling Speeds

The following table compares the sampling speed in samples per second per channel attainable for various channel types and sampling modes using the DT800's default settings. (Higher speeds are possible by fine-tuning the DT800's settings — see "Getting Optimal Speed from Your DT800" on page 188.)

		Samples/Second (Hz) for EACH Channel			
		Number of Channels Sampled Simultaneously			
Channel Type	**Mode**	**1**	**5**	**10**	**20**
Voltage (no correction; **VNC**)	Normal	45	44	23	14
	Fast	210	140	100	63
	Burst	50k	12k	6k	3k
Voltage (with zero correction; **V**), current, strain (voltage excite)	Normal	40	14	7	4
	Fast	200	110	70	80
	Burst	25k	3k	1.5k	750
Thermocouples	Normal	40	14	7	3.8
	Fast	180	90	54	29
	Burst	12k	3k	1.5k	750
Resistance, RTDs, strain (current excite)	Normal	40	7	4	2
	Fast	170	65	34	16
	Burst	12k	1.6k	800	400
AC voltage	Normal	7.5	1.5	0.7	0.38

Table: Sampling Speeds

Normal Mode Sampling

To take a **reading** of one of its analog input channels, the DT800 does much more than just measure the channel once. It actually measures the channel many times (in the background), averages these **background measurements** to produce a **fundamental sample**[4], then carries out computations on the appropriate fundamental samples to arrive at "the reading" — see Figure 51.

In fact, the DT800 takes 80 background samples within each cycle of the local mains electricity supply (20ms at 50Hz, 16.7ms at 60Hz). It does this for a very important reason — to reject hum (unwanted noise) that can be induced into signals by the mains supply. Moreover, the DT800 is capable of doing this for up to five channels at a time, by interleaving them within the mains period.

This method — taking 80 background measurements for each of up to five channels within a mains period and averaging them for the period — is the DT800's default mode of operation and is called **normal mode** sampling. There are no commands required to implement it, and it supports all schedule functions (reading input channels, setting output channels, executing calculations and processing alarms, for example).

This results in an overall sampling speed for, say, voltage channels (**V** channel type) of approximately 29Hz. That is

- one **V** channel can be sampled approximately 29 times each second, or
- 29 **V** channels can be sampled approximately once each second.

The actual sampling speed varies a little for different channel types and other conditions.

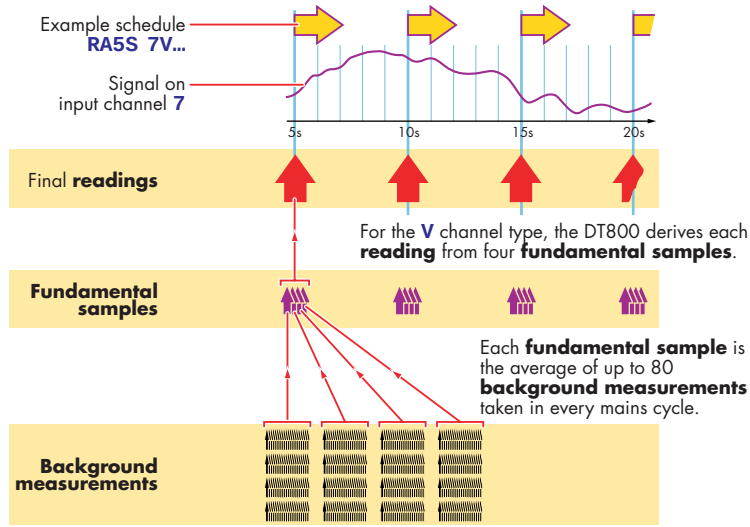


FIGURE 51 Concepts: readings, fundamental samples and background measurements — normal mode and fast mode

[4] Fundamental samples are also covered in
- "Burst Factor 1 — Number of Fundamental Samples" (page 56)
- "Number of Fundamental Samples per Reading" (page 63)
- the Fundamental Samples column of the DT800 Channel Types table (page 63)
- "Speed Factor — Channel Type (Fundamental Samples)" (page 188)
- "reading" (page 209).

Normal mode sampling has greatest accuracy, greatest resolution and greatest noise rejection of the three sampling modes, and is appropriate to most data collection applications where a number of channels are to be sampled from once per second to once per minute to once per hour and longer.

Fast Mode Sampling

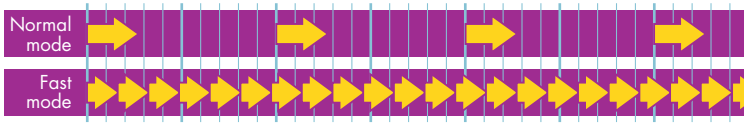


FIGURE 52 Fast mode compared to normal mode

You're able to make the DT800 sample faster by altering some of its settings. For example, you could

- decrease the number of background measurements taken to average for each fundamental sample (set P46 to, say, 5 or 2; normal mode's default is 80)
- set the ADC frequency to maximum (set P60=100)
- increase the setting for local mains frequency (set P11 to, say, 500 or 1000)
- optimize other minor ADC settings (set P58=100, for example).

But making such changes has a global effect — that is, all schedules are sampled according to the new settings. This may be a disadvantage because, if you speed up sampling in this way to accommodate schedules that require fast sampling, the resulting lower performance (reduced resolution, accuracy and noise rejection) is also applied to any slow schedules you have.

Therefore, a better alternative is to set these parameters on a schedule-by-schedule basis. The **fast mode modifier** allows you to do this.

Fast Mode Modifier — General

The general format of the fast mode modifier is

```
,FAST(m,ADC_frequency,mains_frequency)
```

where

		Range
m	is the number of background measurements to average for a fundamental sample (P46)	1 to 150 (default = 2)
ADC_frequency	is the ADC speed in Hz (P60)	1 to 100 (default = 100)
mains_frequency	is the local mains frequency in Hz (P11)	48 to 1000 (default = 1000)

Fast Mode Modifier — Default

You'll mostly use the fast mode modifier in its default form

```
,FAST
```

(all quantifiers omitted) because the default causes the DT800 to scan continuously at the fastest speed possible for this mode (burst mode is faster still — see page 53) by automatically setting

- number of background measurements taken to average for each sample to 2 (P46=2)
- ADC frequency to 100kHz (P60=100)
- local mains frequency to 1000Hz (P11=1000)
- frame capture cycles to 1 (P57=1).

Using Fast Mode

To use the fast mode modifier, you apply it to individual schedules. For example

```
RA,FAST 1V 2V 3V
```

specifies that the **RA** schedule is to run continuously (no trigger) in default **FAST** mode to sample the three analog channels.

No Trigger in Fast Mode Schedules

Although you can include a trigger in a report schedule containing the fast mode modifier, there's no benefit in doing this — fast mode is intended for continuous use (see Figure 52, and "Continuous Report Schedules (No Trigger)" on page 48).

To see why, consider the time-triggered (**3S**) schedule command

```
RA3S,FAST 1V
```

which instructs the DT800 to go into fast mode (alter its background measurement, ADC frequency and mains frequency settings), take a reading, wait three seconds (**3S**) and repeat this process until you halt the schedule. Here the DT800 is simply operating exactly as it would in normal mode, but with the reduced noise rejection, reduced resolution and reduced accuracy of fast mode. For this reason, we recommend that you reserve the fast mode modifier for use when you want the DT800 to take continuous readings at high speed (the speed is determined by the fast mode modifier's settings).

Fast Mode and Burst Mode

The DT800 has another mode of operation called **burst mode** (see page 53, in which it takes readings continuously and even more rapidly than it can in fast mode. But this very high speed operation can only be sustained for short periods of time — that is, in bursts. Therefore:

- Use fast mode when you require rapid measurements to be taken over a long time period.
- Use burst mode when you require extremely rapid measurements to be taken but only for a brief time period.

Fine-Tuning Fast Mode

If you need to match specific requirements, you can set the variables for the fast mode modifier to produce sample rates that are faster than normal mode, but less than the fastest obtainable in fast mode. For example

```
RA1M,FAST(20,50,100) 1V 2V 3V
```

The Real Advantage — Tailored Sampling

The real advantage of the fast mode modifier is that individual report schedules can be tailored to sample faster, and others to sample more slowly but at greater accuracy and resolution. For example, in the program

```
BEGIN
 RA5M 1..4TK
 RB,FAST 8V 9V 10V
END
```

the thermocouples (**RA5M 1..4TK**) are sampled every 5 minutes in normal mode (no FAST modifier) for maximum accuracy and resolution. Meanwhile, the voltages are being measured continuously at the fast mode rate (**RB,FAST 8V 9V 10V**).

Other Influences on Sampling Speed

You can further increase sampling speed by doing one or more of the following:

- Use simple channel types — for example, use uncorrected voltage **VNC** instead of corrected voltage **V**, or measure only voltage in bridge measurements (assume excitation is constant).
- Disable the DT800's auto-range feature by applying a **GL...** channel option (see page 72).
- If data is returned in real time, use a minimal data format and increase the comms baud rate.
- Don't log data (memory access is relatively slow, especially to Flash cards).

See the sections "Getting Optimal Speed from Your DT800" on page 188 (particularly the "Speed Factor — Channel Type (Fundamental Samples)" subtopic), and "Number of Fundamental Samples per Reading" on page 63 for further discussion of sampling speed.

Burst Mode Sampling

The DT800 also supports **burst mode** sampling, which captures analog input data at very high sampling rates for short periods of time. This is useful for capturing waveforms, pulses and other fast analog events.

Although you implement burst mode in the same way as you do fast mode — by applying a schedule modifier to a report schedule — the mechanisms of the two modes are completely different, and burst mode allows much higher sampling speeds than fast mode.

Important You can only sample analog input channel types in burst mode. The DT800 does not support the following in burst mode:

- frequency channel types
- digital and counter channels
- serial sensor channel
- delays
- analog output channels
- system variables, system timers
- calculations, spans, polynomials, attenuation factors
- ALARM, IF or DO commands

Burst Mode Basics

When a burst schedule commences, the DT800 continuously samples the analog channels being bursted at the specified burst clock speed (typically much faster than normal mode or fast mode) and loads them into a circular data buffer, the DT800's **burst memory**, which holds exactly the number of readings you specify — see Figure 53. When the burst memory is full, the next new reading overwrites (replaces) the oldest reading, and so on.

The DT800 continues this process of very fast sampling and circular updating of the oldest readings in burst memory until either the burst is completed or the burst times out.

Bursts and DT800 Resources

Burst mode sampling uses all of the resources of the DT800, and so, during a burst, no other functionality is possible. When the burst is complete, other functionality resumes.

Moreover, you can include more than one burst channel in a schedule and more than one burst schedule in a job, but the DT800 only carries out one burst at a time. (Although, when processing a burst schedule containing several channels, the DT800 interleaves its focus across each of the channels until the bursting of all the channels is complete.)

Burst Pre/Post Trigger — Introduction

In addition, if you want, you can specify an analog or digital trigger for the burst, called the **pre/post trigger**, and data from burst memory can be captured relative to this trigger (that is, before and after this trigger). The position of the pre/post trigger in the burst memory (specified in %) defines the proportion of the memory allocated to pre-trigger data, and the

For a burst, the DT800 temporarily allocates part of its memory to hold exactly the quantity of fundamental samples required to obtain the number of burst ***readings*** you specify. This component of the DT800's memory is called **burst memory**.

Burst memory is **circular**. That is, once the burst memory is full, the next new sample replaces (overwrites) the oldest sample, and so on. In this way, the burst memory always holds the **latest** and the **exact number** of ***readings*** you specify.

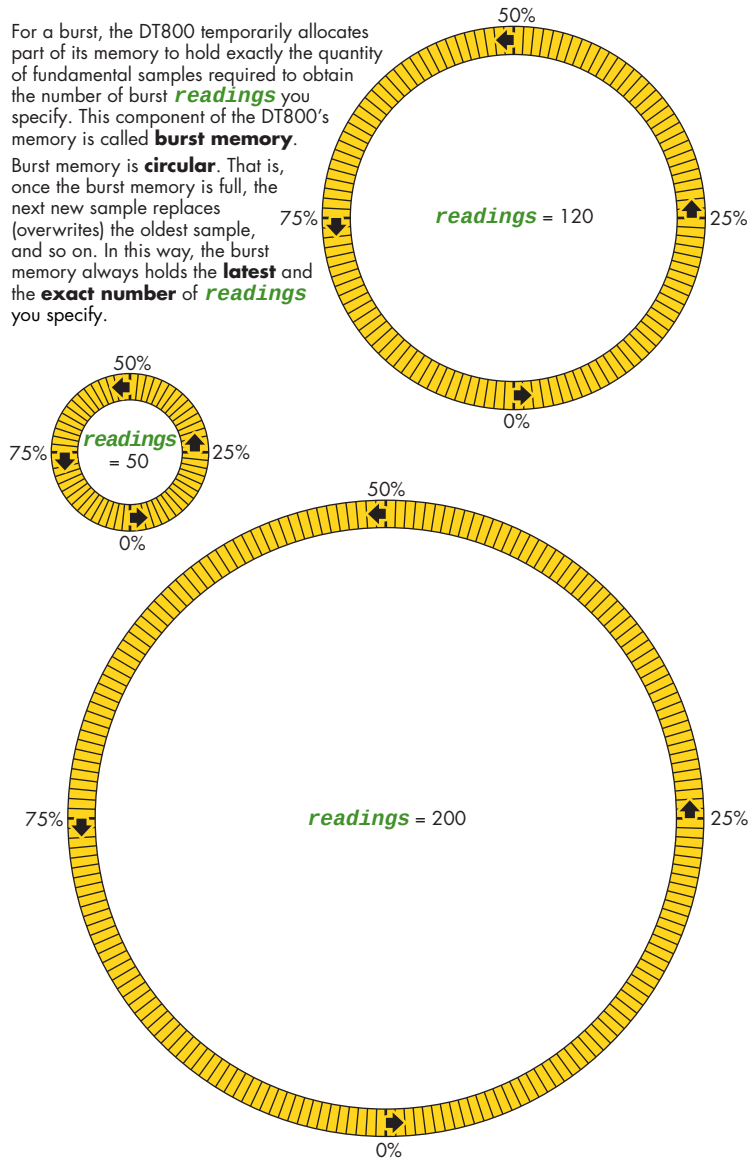


FIGURE 53 Burst memory is circular and adjusts its size to suit the number and type of ***readings*** required.

proportion allocated to post-trigger data (called **pre-trigger memory** and **post-trigger memory**) — see Figure 54.

Warning Don't confuse this pre/post trigger with the trigger of the burst's report schedule:

- The report schedule trigger (**5S** in **RA5S...**, for example) starts the process of loading samples into the circular burst memory at burst clock speed.
- The pre/post trigger controls the "snapshot" of the burst memory that the DT800 captures. That is, which readings — relative to the position of the pre/post trigger — the DT800 returns and/or logs.

See also "Report Schedule Trigger" (page 56) and "Pre/Post Triggers" (page 57).

When the pre/post trigger occurs, the DT800 returns pre-trigger and post-trigger data relative to the trigger position you specify. For example:

- For the 120 readings and **50% pre/post trigger position** shown in Figure 54 (page 55), the 50% trigger instructs the DT800 to return a 120-readings burst consisting of the 60 readings taken immediately prior to the pre/post trigger occurring ("pre-trigger"), and the 60 readings taken immediately after the pre/post trigger occurred ("post-trigger").
- For the 120 readings and **30% pre/post trigger position** shown in Figure 54, the 30% trigger instructs the DT800 to return a 120-readings burst consisting of the 36 readings taken immediately prior to the pre/post trigger occurring ("pre-trigger"), and the 84 readings taken immediately after the pre/post trigger occurred ("post-trigger").
- For 120 readings and **0% pre/post trigger position**, the DT800 returns the 120 readings that immediately follow the trigger (all post-trigger).
- For 120 readings and **100% pre/post trigger position**, the DT800 returns the 120 readings stored prior to the trigger (all pre-trigger).
- For 120 readings and **no pre/post trigger** defined, the DT800 simply returns the first 120 burst readings it collects.

Post-Burst Calculation Period

During a burst, the DT800 rapidly samples the analog inputs being bursted and temporarily stores the resulting primary ADC data. Then when the burst is complete, the DT800 calculates this primary data to engineering units ready for return and/or logging. This post-burst calculation period can take several seconds depending on the number of data points and type of analog data.

The DT800's default is to return burst data to the host computer immediately after it is collected and calculated. You can disable this — see **/B** in the DT800 Switches table (page 111).

Stopping a Burst

Any command arriving at the DT800's **Host RS-232** port or **Ethernet** port cancels a burst.

Maximum Speed of Burst Mode

See "Best Speed in Burst Mode" on page 188.

Burst Mode Modifier

The format of the burst mode modifier is

;BURST(*readings*{,*speed*{,*trigger*{,*position*{,*timeout*}}}})

where

readings	is the number of readings to be taken for each channel in the burst. Figure 53 illustrates this. Range 1 to 65535 fundamental samples — see "Actual Number of Burst Samples" below.
speed	is the burst clock speed (ADC speed) in Hz. ***speed*** is optional. If omitted, speed is determined by P48 (default is 100kHz). Range 1000 to 100000
trigger	is the pre/post trigger quantifier. ***trigger*** is optional. If omitted, the burst returns 100% of readings immediately. If included, it can be either • a digital event ***n*+E**, ***n*–E** or ***n*E** (see "Pre/Post Trigger — Digital Event" on page 57), or • an analog level defined by the **LEVEL** channel option (see "Pre/Post Trigger — Analog Level" on page 57).
position	is the pre/post trigger position in the burst memory, expressed as the proportion of pre-trigger readings in %. See examples in "Burst Pre/Post Trigger — Introduction" (page 53), and Figure 54 (page 55). ***position*** is optional. If omitted, 100% of the readings are taken after the trigger. Range 0 to 100%
timeout	is the pre/post trigger timeout period. If the pre/post trigger does not occur in this period, the DT800 terminates the burst and returns a `Burst Timeout` message. **nS** — timeout is in seconds **nM** — timeout is in minutes **nH** — timeout is in hours ***timeout*** is optional. If omitted, timeout is determined by P54 (default is 10S). Note that a ***timeout*** of **0S**, **0M**, or **0H**, or setting **P54=0**, creates an infinite timeout. (The burst can be cancelled by any command arriving at the DT800.)

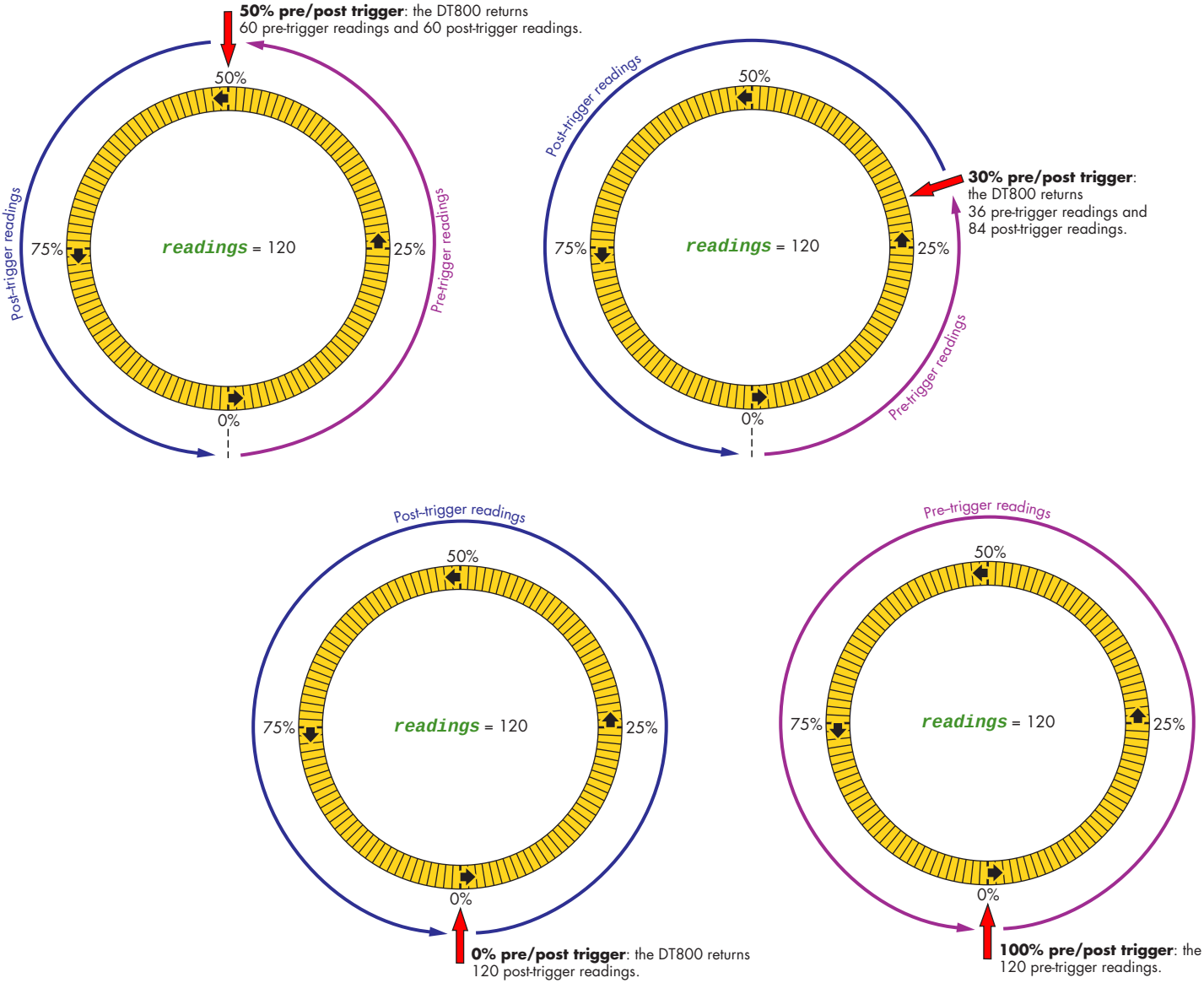


FIGURE 54 Burst pre/post trigger examples (50%, 30%, 0% and 100%)

Actual Number of Burst Samples

There are two important factors that contribute to the actual number of samples that the DT800 takes in a burst. The factors are

- for each channel type, the number of **fundamental samples** required to obtain an individual **reading** — see Figure 55
- the number of channels in the burst.

Burst Factor 1 — Number of Fundamental Samples[5]

For many of its channel types, the DT800 must take more than one sample in order to arrive at a reading. For example:

- a voltage-not-corrected reading (**VNC** channel type) requires the DT800 to take just one sample (which is also the final reading), whereas
- a corrected voltage reading (**V** channel type) actually requires the DT800 to take four samples and perform a calculation to arrive at each final reading.

Such samples are called **fundamental samples**.

Burst Factor 2 — Number of Channels

The number of samples taken by a burst also depends on the number of channels in the burst, because the number of ***readings*** is returned for each channel in the burst. For example, the burst modifier

```
;BURST(100) 1VNC
```

returns 100 readings for the single channel **1VNC**, a total of 100 readings, whereas

```
;BURST(100) 1VNC 2VNC 3VNC
```

returns 100 readings for each of the three channels, a total of 300 readings. Therefore, the circular burst memory size is 100 samples in the first example, and 300 samples in the second example.

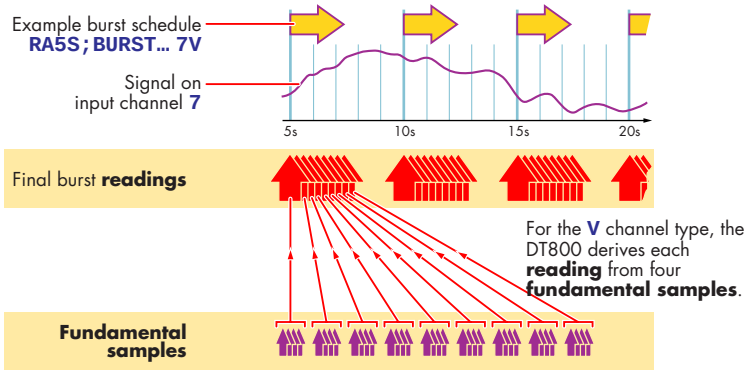


FIGURE 55 Concepts: readings and fundamental samples — burst mode

[5] Fundamental samples are also covered in
- "Normal Mode Sampling" (page 51)
- "Number of Fundamental Samples per Reading" (page 63)
- the Fundamental Samples column of the DT800 Channel Types table (page 63)
- "Speed Factor — Channel Type (Fundamental Samples)" (page 188).

Example — Actual Number of Burst Samples

The following burst modifier demonstrates the combined effect of both of the above factors:

```
;BURST(100) 1V 2V 3V
```

The **V** channel type in the **1V** channel ID requires the DT800 to take four fundamental samples to arrive at each reading (see the Fundamental Samples column of the DT800 Channel Types table on page 63), and 100 readings are wanted in the burst (**(100)**). That is, 4 x 100 = 400 samples are required to obtain the 100 burst readings for the **1V** channel. Similarly, the DT800 must sample the **2V** channel 400 times and the **3V** channel 400 times to obtain their 100 readings each. Therefore, the total number of fundamental samples required to obtain 100 readings of each of the three channels is 3 x 400 = 1200.

Note The DT800's burst memory is limited to 65,535 fundamental samples taken in binary multiples called **frames** (see the footnote on page 188) — not 65,535 readings.

Bursts — The Bottom Line

Because of the two factors above, the amount of burst memory required by a burst schedule can sometimes be complex to determine. Because of this, the DT800 always checks your burst commands prior to executing the first burst and returns the error message `E34 - Too many samples requested for burst` if the number of fundamental samples required exceeds the DT800's burst memory capacity of 65,535 fundamental samples. Therefore, we recommend you simply send a test burst and, if E34 is returned, make changes to your burst schedule such as

- reducing ***readings***
- using alternative channel types that require fewer fundamental samples (for example, use **VNC** instead of **V**) — refer to the Fundamental Samples column of the DT800 Channel Types table (page 63)
- reducing the number of channels included in the burst — use separate bursts of (ideally) just one channel each, rather than one burst modifier containing several channels.

Report Schedule Trigger

You always use the burst mode modifier in conjunction with a report schedule. The report schedule's trigger starts the burst process, which then proceeds according to the burst quantifiers you specify.

Examples — Report Schedule Triggers and Bursts

The report schedule

```
RA5S;BURST(100,1000) 1V 2V 3V
```

instructs the DT800 to carry out a burst sampling of the three voltage channels every 5 seconds.

The report schedule

```
RC1+E;BURST(250,5000) 1..4VNC
```

instructs the DT800 to carry out a burst sampling of the four voltage channels every time a 0 to 1 transition event is received by digital input channel 1.

The report schedule

```
RK;BURST(300,1000) 1V 2V
```

instructs the DT800 to continuously collect bursts of 300 readings from each of the two voltage channels. (When no burst trigger is specified, the DT800 commences the burst within 30ms of the schedule trigger occurring.)

Pre/Post Triggers

See the lead-in to this topic in "Burst Pre/Post Trigger — Introduction" on page 53.

The burst pre/post ***trigger*** specifies the point at which pre-trigger buffering stops and post-trigger buffering starts (buffering = loading into burst memory). Don't confuse it with the trigger of the report schedule that contains the burst modifier.

A burst pre/post trigger can be either a digital event or an analog level.

Pre/Post Trigger — Digital Event

The burst pre/post ***trigger*** can be a digital event that is input to the DT800 by means of any of its digital input channels.

The DT800 recognizes the following transitions:

a **0 to 1** transition	specified by ***n*+E**	where ***n*** is the digital channel number
a **1 to 0** transition	specified by ***n*-E**	
any transition OR	specified by ***n*E**	

The DT800 responds to the pre/post trigger within 10µs of the trigger occurring.

Example — Digital Event Pre/Post Trigger

The schedule command

```
RF60S;BURST(2000,5000,9+E,50) 11V
```

instructs the DT800 to collect a burst of **2000** voltage readings at **5000**Hz (0.2ms intervals) from analog channel **11** every 60 seconds (**RF60S**). When a positive-going transition occurs on digital input channel 9 during the burst (**9+E**), save the most recent **50**% of readings in pre-trigger memory and collect the subsequent 50% of readings in post-trigger memory. When the data is later returned, the trigger will have occurred in the middle of the data set — 1000 readings before the pre/post trigger, and 1000 readings after the pre/post trigger.

Pre/Post Trigger — Analog Level

The burst pre/post trigger can also be a particular value or level reached on a DT800 analog input channel (which can be a channel under test or a separate trigger channel). The general format of this burst pre/post triggering is

```
;BURST(readings{,speed{,LEVEL{,position{,timeout}}}})
```

where

LEVEL	indicates that one of the analog channels in the channel list will provide the pre/post trigger

In addition, the trigger channel must be in the schedule channel list and have the **LEVEL** channel option applied to it:

```
;BURST(readings{,speed{,LEVEL{,position{,timeout}}}})
channel(LEVEL>value{:nCV})
;BURST(readings{,speed{,LEVEL{,position{,timeout}}}})
channel(LEVEL<value{:nCV})
```

where

***channel*(LEVEL>*value*)**	generates a pre/post trigger if the reading for the ***channel*** exceeds ***value***	More than one channel in the burst schedule can have the **LEVEL** channel option, and any one can trigger pre/post buffering.
***channel*(LEVEL<*value*)**	generates a pre/post trigger if the reading for the ***channel*** falls below ***value***	

:*n*CV	is a channel variable While condition (see "Trigger While" on page 48). **:*n*CV** is optional. It is set to 1 if this channel's input level triggered the burst; otherwise, ***n*CV** is set to 0. The CVs can be tested later to identify the trigger that caused the burst.

Note The channels used to trigger must be type VNC. This is because the burst mode does not calculate the actual readings until after the burst has finished reading.

Examples — Analog Level Pre/Post Trigger

The schedule command

```
RA;BURST(500,2000,LEVEL,50,1M) 1VNC(LEVEL>200)
```

instructs the DT800 to run the burst continuously (**RA**, no report schedule trigger), and the burst cycles 500 readings (**500**) in the burst memory looking for the level of the test channel to exceed 200mV (**LEVEL>200**). When the pre/post trigger occurs, save the most recent **50**% of readings in pre-trigger memory and collect the subsequent 50% of readings in post-trigger memory. Timeout if the trigger does not occur in 1 minute (**1M**).

This program is appropriate for "wave catching" or "pulse catching".

The program

```
BEGIN
 RA1+E;BURST(100,1000,LEVEL,25)
  1VNC(LEVEL>1200:1CV)
  2VNC(LEVEL>1000:2CV)
 RB1-E
  1CV("Chn 1 trigger")
  2CV("Chn 2 trigger")
END
```

defines a level-triggered burst that can be triggered either by channel **1VNC** exceeding 1200mV (**LEVEL>1200**), or by channel **2VNC** exceeding 1000mV (**LEVEL>1000**). When the trigger occurs, 25% of readings are kept in pre-trigger memory and the following 75% of readings are collected in post-trigger memory. The pre/post trigger channel is returned by the (**RB1-E**) schedule.

Working with Schedules

Entering Schedules Into the DT800 (BEGIN–END)

Report schedules must be entered into (that is, sent to) the DT800 as a group. And because the schedules and processes that comprise a job or program often extend over more than one line, you embrace them between the keywords **BEGIN** and **END** to designate the beginning and end of the group to the DT800. Here's an example:

```
BEGIN
 RA10S
  4TT("Oven Temp")
  5TT("Flue Temp")
 RB1S
  2C("Water Flow")
END
```

Note You cannot append additional channels to a schedule once it has been sent to the DT800. Instead, you must re-send the full set of schedules, including the additional channels. (But it is possible to alter an individual schedule's trigger — see "Changing a Schedule Trigger" on page 59.)

BEGIN–END Rules

Note the following rules:

- Each line can be up to 254 characters long.
- Processes on lines without a schedule header are included in the schedule immediately above. In the example above, **4TT("Oven Temp")** and **5TT("Flue Temp")** are included in schedule **RA**, and **2C("Water Flow")** is included in schedule **RB**.
- A carriage return must terminate each line.
- Always place the **END** statement on a separate line, by itself. (If a syntax error exists in a line, the DT800 ignores the remainder of that line. Therefore, if such a line contains an **END** statement, the DT800 never sees the **END** statement, which can leave the DT800 in an indeterminate command processing state.)

How BEGIN–END Works

When the DT800 receives the **BEGIN** keyword, the DT800 halts all currently-executing schedules and deletes them, ready to receive the new program… UNLESS

- the schedules are locked (see **/F** in "Switches" on page 111), or
- the current program contains data, or
- the current program contains alarms, or
- the current program is locked.

The BEGIN–END construct can contain blank lines and any other DT800 commands (these are executed on entry). When **END** is received, the original Halt/Go state is restored.

See "Jobs" on page 15 for additional important BEGIN–END information.

Using Immediate Schedules in Programs

Immediate schedules are often included in DT800 programs for tasks such as assigning initial values to channel variables, setting output channels to initial states, taking tare readings for input channels, and so on.

In these cases, you enter the immediate schedule processes after the **BEGIN** keyword and before the first report schedule. For example, here's a program containing three immediate schedules (lines 2, 3 and 4):

```
BEGIN
 10CV(W)=10
 22CV(W)=125
 4DSO=1
 RA1M
  1TK("Oven Temp",=1CV)
  IF(1CV>22CV){[10CV=10CV-1]}
  IF(10CV<1){[4DSO=0]}
END
```

When you send the above program to the DT800, the **CV** and **DSO** assignments (lines 2, 3 and 4) are made exactly as if you'd sent the three lines as separate immediate schedules.

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 page 111).

Synchronize-To-Midnight Switch Enabled

If the synchronize-to-midnight switch is enabled (**/S**, the DT800'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 DT800 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 DT800 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.

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 you send the schedule

```
RA10H
```

to the DT800 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.

Retrieving Entered Schedules and Programs

Use the **SHOWPROG** command to return the current program running in the DT800, or the **SHOWPROG"JobName"** command to return the program for **JobName** in the logger (see the DT800 Job Commands table on page 16). These commands return everything between **BEGIN** and **END**.

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

You can change a schedule's trigger at any time simply by sending a new schedule ID and trigger without any processes, such as

```
RC10M:2W
```

Important If you include any processes, a new schedule is created that replaces all previous schedules UNLESS

- the previous schedules have logged data into memory, or
- logging is enabled by the **LOGON** command (see "LOGON and LOGOFF Commands" on page 76), or
- the schedules are locked by the **/F** switch (page 111), or
- a **LOCKJOB…** command has been applied (see the DT800 Job Commands table on page 16), or
- the **PROFILE"JOB_SETTINGS"…** command has been used to prevent the re-definition of jobs (see "User Startup Profile" on page 113).

Naming Schedules

You can add a name (maximum eight characters, no spaces) to any of the general-purpose report schedules (**RA**, **RB**,…**RK**). For example

```
RA"Boiler_1"10S 1..5TK
```

Schedule names are returned in the **DIRJOBS** report (page 16) and **STATUS14** report (page 122).

Halting & Resuming Schedules

Schedules can be halted individually or as a group:

H	Halt all schedules
HA, HB,…HK	Halt **RA**, **RB**,…**RK** schedule
HS	Halt the statistical sub-schedule (see "Statistical Sub-Schedule Halt/Go" on page 50)

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

G	Resume all schedules
GA, GB,…GK	Resume **RA**, **RB**,…**RK** schedule
GS	Resume the statistical sub-schedule (see "Statistical Sub-Schedule Halt/Go" on page 50)

Locking Schedules

Schedules in the DT800 can be locked by the **/F** switch command (page 111) to prevent them from being accidentally changed or deleted.

You can also lock the entire job that contains the schedules — see the **LOCKJOB"*JobName*"** command in the DT800 Job Commands table (page 16).

Deleting Schedules

You cannot delete individual schedules from the DT800 — you must delete the entire job containing the schedules.

Important A locked job must be unlocked and any stored data and alarms deleted before the job itself can be deleted — see the **UNLOCKJOB**, **DELDATA**, and **DELALARMS** commands in the DT800 Job Commands table on page 16.

Once a job has been unlocked, and stored alarms and data has been deleted, use the following commands to delete the job:

Command	Action	
DELJOB	Deletes the current job from the DT800	See the DT800 Retrieval Commands — Summary table on page 186.
DELJOB"*JobName*"	Deletes only ***JobName*** from the DT800	
DELJOB*	Deletes all jobs from the DT800	

If any schedule has stored alarms or data into memory, or data logging is enabled by **LOGON**, or schedules are locked by **/F**, you cannot erase the job (the DT800 issues error message E4 or E48 — see "Error Messages" beginning on page 197).

Sending a new job to the DT800 automatically erases the current job from the runtime environment. However, the job and its data remains in memory and can be run again at a future time.

Special Commands in Schedules

Here are some special commands that you'll find useful for controlling the flow of data-processing in schedules. There are two tools for conditional program flow control:

- the **IF...** command
- conditional expressions

and one tool for unconditional program flow control:

- the **DO...** command.

Conditional Processing — IF... Command

The DT800's **IF...** command allows processes to be performed conditionally — that is, upon other factors being true. The general format of the command is

```
IF(condition)"actionText"{[actionProcesses]}
```

where

If ***condition*** is true	the processes are carried out	See "Condition Tests" below.
If ***condition*** is false	the processes are not carried out	
actionText	is optional and, if included, is returned to the host computer each time the **IF...** command tests true. The ***actionText*** is not logged.	
actionProcesses	can be reading input channels, setting output channels, performing calculations, setting of any global or system parameters, and so on. If the ***actionProcesses*** produce data, this can be returned to the host computer but is not logged.	

Condition Tests

The ***condition*** works in the same way as the condition for alarms. Normally, channel variables are tested in the condition, and the tests can be as follows:

Operator	Operation	Number of Setpoints
<	Less than setpoint	1
>	Greater than or equal to setpoint	1
<>	Less than first setpoint, OR greater than or equal to second setpoint	2
><	Greater than or equal to first setpoint AND less than second setpoint	2

The setpoints can be a floating-point constant or a channel variable. The number of setpoints depends on the logical operator.

The **IF...** condition is tested each time that the schedule it belongs to is run, and the text and processes are executed every time the condition tests true.

Examples — IF... Command

When run in a schedule, the **IF** command

```
IF(1CV>3.57){[2CV=12.6]}
```

specifies that if the current value of 1CV exceeds 3.57 (**1CV>3.57**), assign the value 12.6 to 2CV (**2CV=12.6**). If 1CV is less than 3.57 when the condition is tested, no assignment is made.

When run in a schedule, the **IF** command

```
IF(10CV><10,100){[5CV(W)=1 RA1M]}
```

specifies that if the value of 10CV is between 10 and 100 (**10CV><10,100**), set a flag (**5CV(W)=1**) and change the trigger for the RA schedule to 1 minute (**RA1M**).

"Working Channels Hide CV Data" (page 92) explains the **W** channel option used in the example above.

IF...THEN...ELSE

The **IF...** command has no ELSE alternative (as in the BASIC language's IF…THEN…ELSE construct). But where an IF…THEN…ELSE test and function is required, you can achieve this using two **IF...** commands.

For example, for any pass of a schedule containing the two IF commands

```
IF(10CV<100){[5CV(W)=0]}
IF(10CV>100){[5CV(W)=1]}
```

only one of the tests will be true, and so the flag will be set appropriately.

See also "Conditional Processing — Boolean Expressions" below.

Conditional Processing — Boolean Expressions

Boolean expressions can also be used in schedules to return a result that is dependent on a condition being true or false.

For example, the expression

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

sets 2CV to a value of **1CV*2** if 1CV is less than 1000, or to a value of **1CV*4** if 1CV is greater than or equal to 1000. (The Boolean expressions **(1CV<1000)** and **(1CV>=1000)** evaluate to a value of **1.0** if true, or **0.0** if false.)

Conditional expressions such as the example above provide an IF…THEN…ELSE function. (See also "IF…THEN…ELSE" above.)

Unconditional Processing — DO... Command

The **DO...** command is similar to the **IF...** command, except that it has no conditional test. This means that its processes are performed each and every time the report schedule containing the **DO...** command is run.

The general format of the **DO...** command is

DO"*actionText*"{[*actionProcesses*]}

where

actionText	is optional and, if included, is returned to the host computer each time the **DO...** command tests true. The ***actionText*** is not logged. See "DO... actionText" below.
actionProcesses	can be commands for reading input channels, setting output channels, performing calculations, setting of any global or system parameters, and so on. ***actionProcesses*** is optional. If the ***actionProcesses*** produce data, this can be returned to the host computer but is not logged. See "DO... actionProcesses" below.

The **DO...** command can include either **"*actionText*"** or **{[*actionProcesses*]}** or both.

The **DO...** command is particularly useful for executing — within schedules — DT800 commands that cannot normally be placed in schedules (parameter, switch, unload, alarm, job and delete commands, for example). In other words, the **DO...** command allows you to place direct commands within runtime commands[6] so that the direct commands are executed during runtime. You can even place **RESET** within a **DO...** command, but we recommend that you NEVER do this.

Warning Use the **DO...** command with great care. Although it's a powerful and flexible command, it does not prevent you from making unreasonable or unrealistic requests of the DT800 during runtime, and some users have found that it can lead to program execution problems or program failure.

DO... actionText

actionText is returned to the host computer whenever the **DO...** command is executed within a report schedule. ***actionText*** can be

- any printable text
- **^A** to **^Z** control characters; **^G** (bell), **^M** (carriage return) and **^J** (line feed) are often useful
- the special substitution characters **#** (replaced by the current time when the ***text*** is returned) and **@** (replaced by the current date when the ***text*** is returned).

DO... actionProcesses

actionProcesses can, in theory, be any DT800 command. But, in practice, some DT800 commands are likely to cause program runtime errors and even program failure. In particular, do not include report schedules or alarms in **DO...** commands.

The most common use of **DO...** ***actionProcesses*** is to change parameter and switch settings to suit requirements of various parts of your program.

actionProcesses are executed where they occur in report schedules. Therefore, be aware that their result is in effect when the DT800 executes the next command of your program.

Examples — DO... Command

The schedule command

```
RA5M DO"Hello World"
```

instructs the DT800 to return the message `Hello World` to the host computer when the schedule runs (every five minutes).

In the program

```
BEGIN
   RB1M
   DO"Boiler 99^M^J"
   1..5TK
END
```

the **DO...** command returns the heading text string `Boiler 99` before each record of data (**^M** = carriage return, **^J** = line feed).

The command

```
DO{[RUNJOB"Phase_2"]}
```

runs the job **Phase_2** each time it executes.

The schedule command

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

instructs the DT800 to output a modem initialization string (**AT&F E0 Q1 S0=2 ^M**) whenever an external event occurs (**2+E**); for example, whenever the modem powers up.

The schedule command

```
RD1M
   DO{[P11=500 P47=10]}
   1..5TK
   DO{[P11=50 P47=5]}
   8V 10V
```

instructs the DT800 to change its ADC setup (**P11** and **P47**) to read channels **1..5TK**, then change it again to read channels **8V** and **10V**.

The schedule command

```
RE1D
   DO{[U"Job1"A]}
   DO{[DELDATA"Job1"]}
```

instructs the DT800 to — every midnight (**1D** trigger) — unload data for schedule A of Job1 (**U"Job1"A**), then delete Job1's data (**DELDATA"Job1"**).

The program

```
BEGIN
   RF5S DO{[/h/r]} 1..3TK
   RG30S DO{[/H/R]} 5PT385 6PT385
END
```

instructs the DT800 to

- every five seconds (**RF5S**) — enter free-format mode (**/h**) and disable the return of real-time data (**/r**) for channels **1..3TK**, and
- every 30 seconds (**RG30S**) — return real-time data (**/R**) in fixed format (**/H**) for the two **PT385** channels.

[6] Direct commands are commands that perform direct tasks within the DT800 the moment they are sent (for example, switch commands and parameter commands). Runtime commands define the runtime tasks (for example, a schedule command).

Specifying Channel Details (Channel Lists)

Define channels using channel number, channel type and channel options

All analog and digital channels on the DT800 are multipurpose. Therefore, to make a channel do what you want it to, you must specify its internal signal and excitation routing, sampling method and data processing. You do this by defining the channel's number, type, and any options — see Figure 40 on page 44 and

- "Channel Numbers" next
- "Channel Types" on page 63
- "Channel Options" on page 70.

Channel Numbers

As Figure 11 on page 24 shows, each input and output channel has a **channel number**, and the terminals of each analog input channel are identified by the implied usage of the channel if no modifiers are applied:

*	Terminal where default excitation is applied, or excite output terminal, or positive shared terminal
+	Positive independent input terminal, or positive shared terminal
–	Negative independent input terminal, or positive shared terminal
#	Return common terminal, or excite return terminal

Thus the channel ID **5V** defines an independent input between the + and – terminals, while **5*V**, **5+V** and **5-V** define shared-terminal inputs between the *, + or – terminals (respectively) and the # terminal.

Figure 56 shows how channel numbers are used in schedules.

Channel Number Sequence

A channel ID that contains two channel numbers separated by two or more 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 example

1..5	is equivalent to	1 2 3 4 5
1+..3–	is equivalent to	1+ 1– 2* 2+ 2– 3• 3+ 3–
1+..3–(#)	is equivalent to	1+ 1– 1# 2* 2+ 2– 3* 3+ 3–
1#..3#(#)	is equivalent to	1# 2* 2+ 2– 3* 3+ 3– 3#

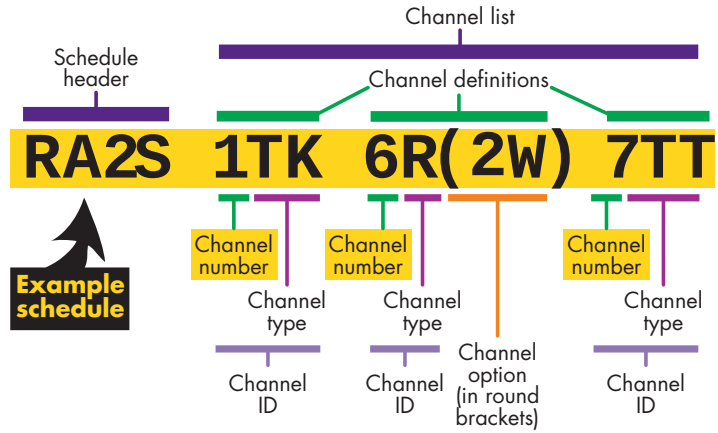


FIGURE 56 Channel number component of typical channel definition

Channel Types

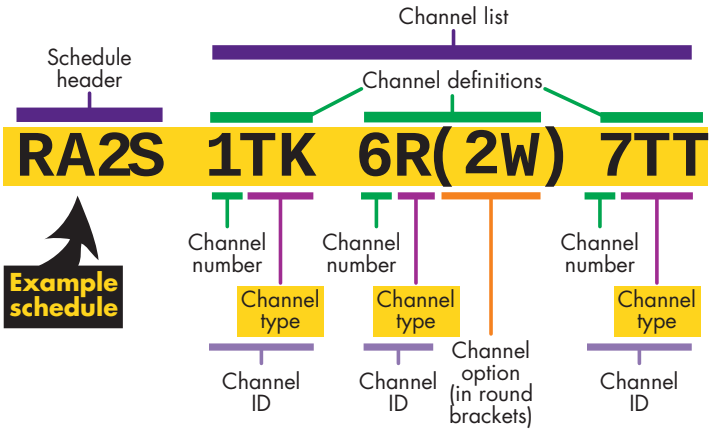


FIGURE 57 Channel type component of typical channel definition

You tell the DT800 what type of sensor you've connected to a channel, and the wiring configuration you've used to connect it, by specifying the channel's **channel type** (Figure 57). In other words, the channel type you include in each channel definition you send to the DT800 is the way you tell the DT800 how you want it to read the channel and then process the reading. For example:

- The channel type **TK** in the channel definition **1TK** tells the DT800 to measure whatever you've connected to input channel **1** as a type K thermocouple.
- The channel type **R** in the channel definition **6R** tells the DT800 to measure whatever you've connected to input channel **6** as a resistance.

This means that it's possible to read one input channel using different channel types. For example, you can read a thermocouple connected to input channel 1 as both a thermocouple and a voltage by sending the channel definitions **1TK** and **1V** to the DT800. These instruct the DT800 to measure channel 1 as a type K thermocouple and then measure it again as a voltage.

The table below lists all channel types available for use with the DT800.

Default Channel Settings

Most channel types have specific settings that the DT800 applies automatically (in the background) to make the channel operate in a standard way. These automatic settings are called **default channel options**. Default channel options for the various channel types are listed in the Default Channel Options column of the DT800 Channel Types table (below). Remember: the DT800 applies these values automatically — you do not have to type them in.

For more information see "Default Channel Options" on page 70.

Default Wiring Configuration

The Wiring Configuration > **Default** column in the Channel Types table below lists the wiring configuration that the DT800 applies if you do not specify a wiring configuration in the channel option.

Number of Fundamental Samples[7] per Reading

The Fundamental Samples column in the table below shows the number of fundamental, low-level DT800 ADC samples required by each channel type to construct a single reading. The fastest rate that the DT800 can take these fundamental samples is 100kHz — that is, 10µs apart.

For example, the **V** channel type requires four fundamental samples: a zero reading taken on the channel's **+** terminal, a reading across the **+** and **–** terminals, another zero reading on the **–** terminal, then another reading across the **+** and **–** terminals (in that order).

Maximum Scan Rate At the DT800's maximum scan rate of 100kHz, four fundamental samples takes 40µs — that is, a 25kHz maximum scan rate for **V**). Similarly, the **R** channel type requires eight fundamental samples (seven rounded up to the nearest power of two) and therefore has a maximum scan rate of 12.5kHz.

[7] Fundamental samples are also covered in
- "Normal Mode Sampling" (page 51)
- "Burst Factor 1 — Number of Fundamental Samples" (page 56)
- the Fundamental Samples column of the DT800 Channel Types table below
- "Speed Factor — Channel Type (Fundamental Samples)" (page 188)
- "reading" (page 209).

See "Default Channel Options" on page 70.
See "A Special Channel Option — Channel Factor" on page 70.

Category	Signal/Sensor Details	Channel Type	Examples	Default Channel Options	Channel Factor	Resolution	Output Units	Wiring Configuration and Default		Fundamental Samples	Comment
Voltage	Voltage — with zero correction ("corrected")	**V**	1V 1+V	(1,T1M)	attenuation factor	1µV	mV	Figures 100 to 104	2W	4	Analog input voltage range is ±10mV to ±13V
	Voltage — no correction ("uncorrected")	**VNC**		(1,T1M)	attenuation factor		mV		2W	1	
	Voltage output on **Ao** terminal	**VO**	VO=100				mV		2W	1	Cannot be used if any other channels use analog triggering in burst mode sampling

Table: DT800 Channel Types (sheet 1 of 5)

See "Default Channel Options" on page 70.

See "A Special Channel Option — Channel Factor" on page 70.

Category	Signal/Sensor Details	Channel Type	Examples	Default Channel Options	Channel Factor	Resolution	Output Units	Wiring Configuration and Default	Fundamental Samples	Comment
Current	Current; use with external shunt (see "Current Inputs" on page 170)	I	2I 3#I	(100)	current shunt Ω	10nA	mA	Figures 105 to 107	2W 4	Requires an external current shunt (typically 100Ω, but higher for small currents)
	4-20mA current loop	L	1#..5#L	(100)	current shunt Ω	0.01%	Percent		2W 4	See "4–20mA Current Loops" on page 140.
Resistance	Resistance by 2, 3 or 4-wire methods, 1MΩ maximum	R	4R(P250)	(0,P7)	Resistance offset adjustment Ω	1mΩ	Ohms	Figures 108 to 115	4W 7	Default excitation: power (7mW) Use 2W or 3W channel options for 2 and 3-wire configurations
Bridge See "Bridges" on page 146	4-wire; quarter, half & full bridge; current excitation	BGI	1BGI(60)	(350,I10)	bridge resistance Ω	1ppm	ppm	Figures 116 to 122	4W 7	Default excitation: current (10mA) External completion required for 1/2 & 1/4 bridges.
	Ratiometric, 4 & 6-wire bridges, voltage excitation	BGV	1BGV	(0,V3500,4W)	offset adjustment ppm	0.3ppm	ppm		4W 4	Default excitation: voltage (3500mV) Use 6W channel option for 6-wire configuration. External completion required for 1/2 & 1/4 bridges.
AC Voltage Measurement	RMS value of the AC component of the signal (by trapezoidal approximation with DC component subtracted).	VAC	3VAC	(100.0,NS5000)	measurement period in ms	10µV	mVac	Figures 100 to 104	2W	Occupies all ADC resources; cannot be used in burst schedule See "AC Voltage (RMS)" on page 140.
Frequency	Frequency measurement (squared waveform preferred)	F	9F	(0.0,GL1V)	threshold in mV	0.01Hz	Hz	Figures 100 to 104	2W —	0.5Hz to 10kHz Occupies all ADC resources; cannot be used in burst schedule See "Frequency" on page 142.
Time, Date and System Timers	**Time** of day	T	T			1S	Time		0	See "Internal Channel Types (in Detail)" on page 68.
	Day or **date**	D	D			1D	Day		0	
	System timers (for program control, for example) See "System Timers" on page 68.	ST	1ST	(60), (60), (24), (7)	range	1	Counts	Internal	0	Increment every sec (1ST), min (2ST), hr (3ST), day (4ST)
Delay	Delays schedule execution for nominated milliseconds (ms)	DELAY	DELAY=12			1ms	ms		0	**Note** Use carefully — prevents execution of any other schedules
System Data	System variable	*n*SV	3..5SV			1	None		0	See "System Variables" on page 69.

Table: DT800 Channel Types (sheet 2 of 5)

See "Default Channel Options" on page 70.
See "A Special Channel Option — Channel Factor" on page 70.

Category	Signal/Sensor Details	Channel Type	Examples	Default Channel Options	Channel Factor	Resolution	Output Units	Wiring Configuration and Default		Fundamental Samples	Comment
Variables	Channel variables: general purpose holders for data, calculation results,…	nCV	5CV			6 digits	None			0	Use channel options (page 75) to assign data to a CV. Read the CV as for a normal channel.
	Integer variables	nIV	3IV				None				See "Rainflow Cycle Counting" on page 87.
Text	General purpose text for headings, etc. (ten 80-character channels)	$ (1$ to 10$)	7$							0	Assign by sending n$="my text" See "Text" on page 68.
Serial Channel	See "Serial Channel — Channel Types" on page 164.										
Temperature	Thermocouples B, C, D, E, G, J, K, N, R, S and T	TB, TC,…	3TJ	(1)		0.1°C	degC	Figures 100 to 102	2W	6	See "Thermocouples" on page 143.
	Platinum RTDs (α = 0.00385, 0.00392)	PT385 PT392	5PT392	(100,P7)	0°C resistance Ω	0.1°C	degC	Figures 108 to 115	4W	7	Default excitation: power (7mW)
	Nickel RTD (α = 0.005001)	NI	1NI(50)	(1000,P15)	0°C resistance Ω	0.1°C	degC		4W	7	Default excitation: power (15mW)
	Copper RTD (α = 0.0039)	CU	7CU(135)	(100,P15)	0°C resistance Ω	0.1°C	degC		4W	7	Default excitation: power (15mW)
	Thermistors: Yellow Springs 400XX series	YS01 to YS07 YS16 YS17	2YS04	(1e10,P15)	parallel resistor Ω	0.1°C	degC		4W	7	Default excitation: power (15mW) — see "Thermistors" on page 144.
	Semiconductor current source types (Analog Devices)	AD590 AD592 TMP17	6AD590	(100,V5000)	shunt resistor Ω	0.1°C	degC	Figures 126 to 128	2W	4	Default excitation: voltage (5000mV) Temperature is determined by current measurement with voltage excitation. Calibrate by variation of shunt value channel factor.
	Semiconductor voltage output types (National Semiconductor Corp.)	LM135 LM235 LM335	3LM335	(1,I1)	Slope correction	0.1°C	degC	Figures 132 to 134	2W	4	Default excitation: current (1mA) Temperature is determined by voltage measurement with 5 volt excitation. Slope correction by attenuation factor relative to 0°K.
	Semiconductor voltage output types (National Semiconductor Corp., Analog Devices)	LM34 LM35 LM45 LM50 LM60 TMP35 TMP36 TMP37	5LM35	(0,V5000)	Temperature offset adjustment	0.1°C	degC	Figures 129 to 131	3W	4	Default excitation: voltage (5000mV) Calibration by temperature offset in °C

See "RTDs" (page 145)

Table: DT800 Channel Types (sheet 3 of 5)

See "Default Channel Options" on page 70.

See "A Special Channel Option — Channel Factor" on page 70.

Category	Signal/Sensor Details	Channel Type	Examples	Default Channel Options	Channel Factor	Resolution	Output Units	Wiring Configuration and Default	Fundamental Samples	Comment
Digital See "Digital Channels" beginning on page 152.	Digital state input (1 bit/input)	**DS**	**4DS**			1	State	Figure 138	0	If averaging, increase precision with **FF*n*** option
	Digital nybble (4 bits/input)	**DN**		**(15)**	bit mask (decimal)		Nybble		0	Result is 0 to 15. Channel number = LSB of nybble. Maximum channel number = 13.
	Byte input on a group of digital channels (8 bits/input)	**DB**	**1DB(7)**	**(255)**	bit mask (decimal)	1	Byte		0	Result is 0 to 255. Channel number = LSB of byte. Maximum channel number = 9.
	Digital word (16 bits/input)	**DW**		**(65535)**	bit mask (decimal)		Word		0	
	Digital state input on an analog channel	**AS**	**5-AS(3300)**	**(2500)**	threshold (mV)	1	State		4	If averaging, increase precision with the FFn option
	Output on a single digital channel. **1** = ON (high/active, 5V) **0** = OFF (low, 0.6V)	**DSO**	**3DSO=1**	**(0)**	delay or width (ms)				0	Delay<65,535ms. ***n*DSO(*delay*,R)** generates pulses. **Note** Use with care (delays other channels)
	Nybble output on a group of digital channels	**DNO**	**1DNO=0**	**(15)**	bit mask (decimal)				0	Zeros in mask not modified. Channel number = LSB of byte. Maximum channel number = 5.
	Byte output on a group of digital channels	**DBO**	**1DBO=0**	**(255)**	bit mask (decimal)				0	Zeros in mask not modified. Channel number = LSB of byte. Maximum channel number = 1. Can only output 1 byte from **1DBO**.
Counter See "Counter Inputs C1 to C8" on page 154.	Up counter	**C**	**1..4C**		range	1	Counts	Figure 138	0	Count range 0 to 2^{32}–1. For example, **1C(3)** counts 0,1,2,0,1,2,... Presetting a counter outside of the maximum count range (for example, **1C(5)=8**) causes an error (99999.9).
Buzzer and Attention LED	Activate DT800's internal buzzer (see "Buzzer" on page 29)	**1WARN**	**1WARN**						0	**1WARN=1** activates buzzer **1WARN=0** deactivates buzzer
	Activate Attention LED (see "Attention LED Commands" on page 29)	**2WARN**	**2WARN**							**2WARN=1** activates Attention LED **2WARN=0** deactivates Attention LED

Table: DT800 Channel Types (sheet 4 of 5)

See "Default Channel Options" on page 70.

See "A Special Channel Option — Channel Factor" on page 70.

Category	Signal/Sensor Details	Channel Type	Examples	Default Channel Options	Channel Factor	Resolution	Output Units	Wiring Configuration and Default	Fundamental Samples	Comment
Internal Maintenance See also "Internal Maintenance" on page 68.	Reference temperature of terminal block (the DT800's body temperature)	**REFT**	**REFT**				degC		2	Used for thermocouple reference junction compensation
	Main internal battery voltage	**VBAT**	**VBAT**				V		2	Terminal voltage of internal 12V lead acid battery. Battery flat if below 11.5V.
	Internal (Lithium) memory-backup battery voltage	**VLITH**	**VLITH**				V		2	Internal memory-backup battery voltage. Replace battery if below 2.8V.
	Internal main battery current	**IBAT**	**IBAT**				mA		2	Positive if charging, negative if discharging
	Input voltage to charger	**VCHG**	**VCHG**				V		2	External charger voltage 0 to +30V
	Analog voltage source reference	**VREF**	**VREF**				V		2	2.45 to 2.55V
	Analog zero voltage reference	**VZERO**	**VZERO**				mV		2	±1 mV
	Host RS-232 port –12V source	**HOVN**	**HOVN**				V		2	
	Kernel 1.182V reference	**VREFK**	**VREFK**				V		2	1.0 to 1.4V
	Kernel zero voltage reference	**VZEROK**	**VZEROK**				mV		2	±50.0mV
	Multiplexer +5V source	**MP5**	**MP5**				V		2	4.5 to 5.5V
	Multiplexer negative voltage source	**MVN**	**MVN**				V		2	–14 to –17V
	Multiplexer positive voltage source	**MVP**	**MVP**				V		2	14 to 17V
	PC Card 12V source	**PCVPP**	**PCVPP**				V		2	12±1V
	PC Card 3V/5V source	**PCVCC**	**PCVCC**				V		2	3.3±0.5V or 5±0.5V
	Raw voltage onto kernel from external supply	**VEXT**	**VEXT**				V		2	
	Serial Channel negative source voltage	**SSVN**	See **SSVN** in the DT800 Serial Channel — Channel Types table (page 164)							
	Threshold DAC output voltage	**TRGLEV**	**TRGLEV**				V		2	–4 to +4V
	Common-mode rejection ratio at maximum gain	**CMRR**	**CMRR**				dB		2	

Table: DT800 Channel Types (sheet 5 of 5)

Internal Channel Types (in Detail)

The DT800 has its own internal channels, which you can read in exactly the same way as the obvious "external" channels. Use the channel types below.

Time

The DT800'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
```

Time can be in several formats, selected by 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 base	705.1667
3	h.h (decimal hours) since base	11.7528
4	D.D (decimal days) since base	724.674
5	s.s (decimal seconds) since base	62611833.6

System variable 12SV returns day.time as decimal days. P40 defines the separator in the hms.s format, which defaults to ASCII 58 (colon).

See also "Setting the DT800's Clock/Calendar" (page 117), and "Efficient Storage of Time and Date" (page 78).

Date

The DT800's real-time clock/calendar also maintains the date, which is read in the same way as a channel but without a channel number. That is, sending

```
D
```

returns

```
Date 26/03/2000
```

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

P31=	Format		Example
0	Day number	DDDDD	724
1	European	DD/MM/YYYY	25/12/2000
2	North America	MM/DD/YYYY	12/25/2000
3	ISO	YYYY/MM/DD	2000/12/25
4	Decimal days since base	D.D	724.674
5	Decimal seconds since base	s.s	62611833.6

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

See also "Setting the DT800's Clock/Calendar" (page 117), and "Efficient Storage of Time and Date" (page 78).

Text

Ten 80-character text channels are available for labelling, data headings, site identification, DT800 identification, and so on. See the Text channel type on page 65.

You define the string by sending

```
n$="my text string"
```

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

Include control characters in the text string as follows: ^M for carriage return, ^J for line feed. (See the Control column in the ASCII Characters table starting on page 189.)

Internal Maintenance

There are several internal maintenance channels, which you read in the same way as normal channels. See the "Internal Maintenance" section of the DT800 Channel Types table (page 67).

System Timers

There are four internal reloading system timers, which you 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 Maximum Value	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 synchronized to the previous midnight or Sunday, and increment at the beginning of each second, minute, hour or day.

They are initialized to new values corresponding to the new time and date if the DT800 clock/calendar time or date is changed. For example, if the time and date are set to 13:45:53 and 25/12/92, the system timers are set to 1ST=53, 2ST=45, 3ST=13 and 4ST=5 (Friday).

System timers have channel types (see ST on page 64):

```
nST(range,R)
```

Set the channel factor *range* between 1 and 65535. If you include the R channel option, the system timer resets to zero after it is scanned (read) in a schedule. If a *range* other than

the default maximum value is specified, the timer is initialized to a value calculated from the previous midnight or Sunday.

System timers can also be assigned an initial value or expression:

n*ST(*range*,R)=*initial

For example, **2ST=3**.

If the ***initial*** value is greater than the ***range***, the timer is set to zero when the next increment is due.

System Variables

System variables provide various system values. A system variable (SV) can be included in a schedule's channel list, or used in the same way as an immediate scan. All system variables are read-only except where indicated as **writable** in the table below.

System Variable	Function		Writable
1SV	Returns kB free in internal memory		
2SV	Returns kB stored in internal memory		
3SV	Returns kB free in memory card (0 if no card)		
4SV	Returns kB stored in memory card (0 if no card)		
5SV	Returns number of statistical scans in last schedule		
6SV	Returns build number of the DT800's firmware (see also **14SV** below and "version number" on page 210)		
7SV	Returns job presence	= 1.0 if a job is loaded = 0.0 if no current job	
8SV	Returns mains frequency in Hz (P11) Defaults to 50/60Hz		✓
9SV	Returns memory card presence	= 0 if no memory card = 1 if memory card inserted	
10SV	Returns ID of the owning schedule	= 1 for RA schedule = 2 for RB schedule ↓ = 11 for RK schedule = 12 for immediate schedule	
12SV	Returns decimal day.time — for example, 56.5 is midday of day 56	Use formatting for more precision — for example, **12SV(FF4)**	
13SV	Returns DT800's serial number		
14SV	Returns version number (major and minor number) of the DT800's firmware (see also **6SV** above and "version number" on page 210) 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	Bits, 0 to 15 8 = RI, 4 = DCD, 2 = DSR, 1 = CTS	
17SV	Returns Host RS-232 port output handshake line states	Bits, 0 to 3 2 = DTR, 1 = RTS	✓

Table: DT800 System Variables (sheet 1 of 2)

System Variable	Function		Writable
18SV	Returns Serial Channel input handshake line states	Bits, 0 to 1 1 = CTS	
19SV	Returns Serial Channel output handshake line states	Bits, 0 to 1 1 = RTS	✓
20SV	Returns alarm presence in current job	1 = alarm(s) specified 0 = no alarms	
23SV	Minimum temperature exposure	°C	
24SV	Maximum temperature exposure	°C	
25SV	Returns current status of a modem connected to the DT800's Host RS-232 port — see "DT800 Modem (Remote) RS-232 Connection" on page 129	0 = no modem connected (direct connection assumed) 1 = modem connected and no call in progress 2 = modem connected and call in progress	
26SV	Burst level trigger span correction factor		✓
27SV	Burst level trigger offset correction factor		✓

Table: DT800 System Variables (sheet 2 of 2)

Note System variables normally have no decimal places. Use the output format channel option **FF*n*** to increase the resolution.

Channel Options

In brackets, separated by commas, no spaces

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, prefixing returned channel IDs with DDE or OLE tags, statistical operations on the channel data, and so on.

Figure 58 shows how a channel option is applied to a channel in a schedule's channel list, by enclosing the channel option(s) in round brackets after the channel ID.

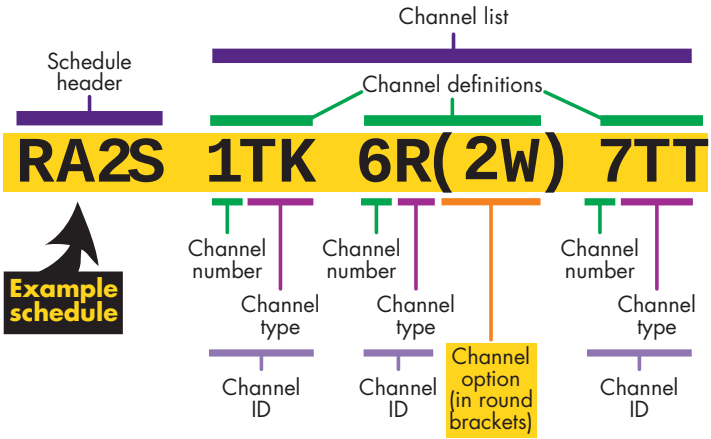


FIGURE 58 Channel option component of typical channel definition

You can apply more than one channel option to a channel ID. If you do this, the channel options need not be in any order, but they must be separated by a comma (no space).

Only certain channel options can be applied to each channel type. If you apply an inappropriate channel option, the DT800 notifies you by returning an `E3 - Channel option error` message.

You can put the same channel in the list more than once, each with different channel options. Then the DT800 treats each occurrence as a separate entity.

Example — Channel Options

The channel options in the channel definition

```
5PT385(4W,200.0,"Steam Temp",FF0)
```

configure the DT800 for 4-wire (**4W**) resistance measurement of a platinum RTD temperature sensor (**PT385**) that has a resistance of **200.0** ohms at 0°C. The channel is labelled **Steam Temp** (appears in the returned data), and **FF0** sets the output resolution to 1°C. In this example, the data is returned as

```
Steam Temp 266 DegC
```

instead of the default

```
5PT385 265.7 DegC
```

Multiple Reports

Multiple reports are possible from each channel by adding additional sets of channel options. The DT800 samples a channel only once every scan. However, second and subsequent sets of channel options generate additional reports. This is particularly useful for statistical reports (see "Statistical Report Schedules" on page 49) and output formatting (see the "Output Data Format" channel options on page 75).

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 options table below (page 74). If statistical options are included, then each option list in the multiple report MUST hold a statistical option.

Mutual Exclusion

Options grouped by a shaded rectangle in the Mutual Exclusions column of the table below are mutually exclusive. That is, if more than one channel option from a mutual exclusion group is placed in a channel list, an error is generated upon program entry.

Order of Application

The Order of Application column in the table below shows the order in which the DT800 applies each channel's channel options. This is necessary to ensure that, for example, the sensor is excited before a polynomial is applied, and that the polynomial is applied before the data is averaged.

The DT800 applies channel options in this order regardless of the order in which you type them in the channel definition.

Default Channel Options

Many channel types have one or more default channel options, which the DT800 applies automatically whenever you specify the channel type. These are listed in the Default Channel Options column of the DT800 Channel Types table (page 63).

For example, whenever you include a resistance channel type **R** in a channel list, the DT800 automatically applies its default resistance channel options (**(0,P7)** — see page 64) unless you specify other values. That is, the DT800 applies

- a resistance offset of zero ohms (**0**), and
- a power excitation of 7mW (**P7** — see the P channel option on page 72)

to the input channel.

See also "Default Channel Settings" on page 63.

Remember: the DT800 applies these default values automatically — you do not have to type them in. But, of course, you can override a default channel option simply by including the corresponding channel option (with a different value) in the channel definition.

A Special Channel Option — Channel Factor

The DT800's **channel factor** channel option **f.f** is the basic tool you use for simple linear scaling of data from an input channel. It's always a number in floating-point or exponential format. See "Channel Options — Scaling" beginning on page 90.

Channel factor is special because, for some channel types, it has a dedicated purpose and therefore cannot be used for simple scaling of these channel types For example:

Channel Type	Channel Factor's Dedicated Purpose
Voltage, thermocouple	Attenuation factor (for example, **1V(5.5)** instructs the DT800 to measure input channel **1** as a **V**oltage and multiply the reading by **5.5**)
LM335	Slope correction factor
Current, 4–20mA current loop	Indicates to the DT800 the resistance of the external current shunt (the DT800 uses this value and the voltage it measures across the shunt to calculate current flow)
Current-excited bridge (BGI)	Indicates to the DT800 the arm resistance used in the bridge
RTD	Indicates to the DT800 the resistance of the RTD element at 0°C (convert RTDs with different 0°C resistances to common units of temperature by specifying each RTD's channel factor appropriately)
System timers, counters	Indicates to the DT800 the maximum or terminal count
Digital…	Various functions — see the Digital category in the DT800 Channel Types table, page 66 (for example, defines the threshold for the analog state input channel type **AS**)

Dedicated channel factors have default values. For example, the current channel type **I** has a default channel factor of 100 ohms (shunt resistance), and the analog state input channel type **AS** has a default channel factor of 2500mV (state recognition threshold).

For channel types where the channel factor has no dedicated purpose, use it for simple linear scaling of the measured data.

For details of the channel factor channel option, see

- the Channel Factor column of the DT800 Channel Types table — page 63
- the Scaling category in the DT800 Channel Options table — page 73
- "Channel Factor (f.f)" — page 90.

Example — Channel Factor

The three channel definitions in the schedule command

```
RA30S 1V(10.1) 4PT385(200.0) 3CV(2.2)
```

contain channel factor channel options that instruct the DT800 to do the following:

- **1V(10.1)** — scale (multiply) the **V**oltage measured on input channel **1** by **10.1**
- **4PT385(200.0)** — use **200.0**Ω (instead of the default 100.0Ω at 0°C) when calculating the temperature represented by the signal from the RTD
- **3CV(2.2)** — scale (multiply) the value in channel variable 3 by **2.2**

Category	Channel Option	Mutual Exclusions Function	Range of Option (n)	Order of Application	Comment
Input Termination	**T100K**		Terminates +, – inputs with 100kΩ to DT800 guard terminal **Gu**	1	Provides input bias current path. Defaults ON for most independent inputs and OFF for shared types. **T1M** is default channel option for **V** and **VNC** channel types.
	T1M		Terminates +, – inputs with 1MΩ to DT800 guard terminal **Gu**		
	T10M		Terminates +, – inputs with 10MΩ to DT800 analog common terminal **Ac**		
	U		Unterminates +, – inputs		
Shared-Terminal Return	**#**		Specifies the location of the negative terminal	1	
Resistance and Bridge	**3W**		Configures channel for a 3-wire measurement	1	For situations where economy in wiring cost is important. The 4-wire method is more accurate.
	4W		Configures channel for a 4-wire measurement	1	4-wire is the DT800's default method of resistance and current-excited bridge (BGI) measurement.
	4WC		Configures channel for a 4-wire bridge with shared completion resistors	1	Declares the **–** terminal of the even channel of the channel pair as the common input
	6W		Configures channel for a 6-wire voltage-excited bridge measurement	1	Used mostly by the **BGV** channel type
	6WC		Configures channel for a 6-wire bridge with shared completion resistors	1	Declares the **–** terminal of the even channel of the channel pair as the common input

Table: DT800 Channel Options (sheet 1 of 5)

Category	Channel Option	Mutual Exclusions Function	Range of Option (n)	Order of Application	Comment	
Gain (manual gain selection)	GL	Locks channel gain to default for channel type		1	Allows the auto-gain system to only decrease the gain	
	GL20V	Locks channel gain for 20V input signal range		1	**Note** Maximum analog input voltage must not exceed ±13V.	Applying a GL*x* channel option disables autoranging for that channel. See "Autoranging" on page 13.
	GL10V	Locks channel gain for 10V input signal range		1		
	GL5V	Locks channel gain for 5V input signal range		1		
	GL2V	Locks channel gain for 2V input signal range		1		
	GL1V	Locks channel gain for 1V input signal range		1	Default for frequency measurement (see "Fixed Gain" on page 142)	
	GL500MV	Locks channel gain for 500mV input signal range		1		
	GL200MV	Locks channel gain for 200mV input signal range		1		
	GL100MV	Locks channel gain for 100mV input signal range		1		
	GL50MV	Locks channel gain for 50mV input signal range		1		
	GL20MV	Locks channel gain for 20mV input signal range		1		
	GL10MV	Locks channel gain for 10mV input signal range		1		
Excitation (output current or voltage)	V or V*n*	Voltage excitation *n* = excitation voltage in mV	0 to 10000mV	1	For sensors requiring voltage excitation. The voltage is measured to full precision when needed (as with bridge channel types).	The excitation system is fully programmable. The DT800 actively measures the excitation voltage and current, and makes adjustments when it is off-target by a percentage specified by P58. **Note** Do not exceed 20mA or 20V. Examples: (V2500) provides 2500mV excitation, (P50000) provides 50mW excitation
	I or I*n*	Current Excitation *n* = excitation current in mA	0 to 20mA	1	For sensors requiring current excitation. The current is measured to full precision when needed (as with resistance channel types).	
	P or P*n*	Power excitation *n* = excitation power in µW	0 to 200000µW	1	For sensors that may require a fixed or maximum power dissipation such as temperature sensors	
	N	No excitation by DT800 (assumes external excitation)		1	Excite terminal may be used as a shared-terminal input channel	
Low Threshold Digital State Input	LT	Configures DT800 digital channel 7 or 8 for low threshold (10mV) detection		1	See "Digital State Inputs D7 and D8 (Low-Level)" on page 153.	
Burst trigger level	LEVEL<*t*:*n*CV or LEVEL>*t*:*n*CV	Sets burst trigger level *t* on analog input		1	See "Pre/Post Trigger — Analog Level" on page 57.	

Table: DT800 Channel Options (sheet 2 of 5)

Category	Channel Option	Mutual Exclusions Function	Range of Option (n)	Order of Application	Comment
Resetting (to zero)	**R**	Reset counter, timer, variable after reading		2	Valid only for counters, system timers, variables (for example, **nCV(R)**), and for pulsing digital state outputs (for example, **1DSO(1000,R)=1** pulses output on for 1000ms).
Wrap range	***wrap***	Number of counts after which a counter automatically resets	1 to 65535	2	See "Counter Channel Options" on page 157
AC Voltage	**NS*n***	***n*** = number of samples; default = 5000	10 to 30000		See "AC Voltage (RMS)" on page 140, especially "Number of Samples".
Scaling See "Channel Options — Scaling" on page 90	***f.f***	Channel factor	±1e18	2	Generally a scale factor specific to channel type (see the Channel Factor column in the table beginning page 63)
	S*n*	Apply span ***n***	1 to 50 (polynomial & span index is shared ⇒ a total of 50 is allowed)	3	Applies a previously-defined span of form **S*n*=**physical low, physical upper, signal lower, signal upper "text" See "Spans (Sn)" on page 90)
	Y*n*	Apply polynomial ***n***		3	Applies a previously-defined polynomial of form Yn=a,b,c,d,f,g"text" (see "Polynomials (Yn)" on page 91)
	F*n*	Apply intrinsic function ***n***.	1 to 7	3	***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 (32-bit) See "Intrinsic Functions (Fn)" on page 90.
	T*n*	Apply thermistor scaling (correction) ***n***	1 to 20		Enables the storing of constants a, b and c in the thermistors formula (see page 144).
Data Manipulation (cannot be used in alarms)	**DF**	Difference Δx = (current reading – previous reading)		4	Returns the difference between the latest reading and the previous reading These options cannot be used directly in alarms. The channel must be included in a scan schedule where the channel value is assigned to a variable that can then be tested in alarm statements. For example **RA2S 1V(RC,=1CV) RZ2S ALARM1(1CV>0.45)1DSO**
	RC	Rate of change (per second) $\frac{\Delta x}{\Delta t}$		4	Rate of change based on latest and previous readings and their respective times
	RS	Reading / time difference (in seconds) $x/\Delta t$		4	Useful when the sensor reading is already a difference (e.g. resetting counters)
	IB	"Integrate" (x_units.seconds) $\left(x - \frac{\Delta x}{2}\right)\Delta t$		4	"Integration" with respect to time between the latest and the previous readings (area under curve)
	DT	Time difference between readings in seconds		4	

Table: DT800 Channel Options (sheet 3 of 5)

Category	Channel Option	Mutual Exclusions Function	Range of Option (n)	Order of Application	Comment
Digital Manipulation (cannot be used in alarms)	TRR	Time from rising edge to rising edge		5	Can be applied to any type of channel (even to non-digital types with the use of channel factor as a threshold)
	TRF	Time from rising edge to falling edge		5	
	TFR	Time from falling edge to rising edge		5	
	TFF	Time from falling edge to falling edge		5	
	TOR	Time of rising edge		5	
	TOF	Time of falling edge		5	
Reference Channel	TR	Cold junction reference temperature		6	Any non-thermocouple temperature sensor measuring isothermal block temperature. If already compensated use 11SV(TR) as reference channel.
Serial Channel	See "Serial Channel — Channel Options" on page 164.			6	
Rainflow Cycle Analysis	See "Rainflow Cycle Counting" on page 87.			6	

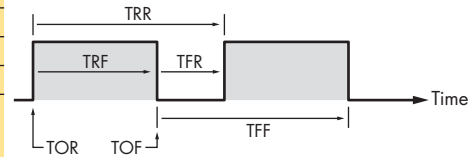


FIGURE 59 Digital channel options

See **P50** and **P51** on page 109.

CONFIGURATION LINE (see "Multiple Reports" on page 70)

Dynamic Data Exchange	DDE	Prefix returned channel ID with DDE tag (&!)		Don't care	Only for use with DeTransfer version 3.00 or later. You can also add DDE or OLE tags to a schedule ID, date or time — see P45 in the DT800 Parameters table (page 109).
Object Linking and Embedding	OLE	Prefix returned channel ID with OLE tag ($!)			
Statistical (cannot be used in alarms) See "Channel Options — Statistical" on page 85.	AV	Average of channel readings		7	(Ave) is appended to returned value
	SD	Standard deviation of channel readings		7	(SD) is appended to returned value
	MX	Maximum channel reading		7	(Max) is appended to returned value
	MN	Minimum channel reading		7	(Min) is appended to returned value
	TMX	Time of maximum channel reading		7	(Tmx) is appended to returned value
	TMN	Time of minimum channel reading		7	(Tmn) is appended to returned value
	DMX	Date of maximum channel reading		7	(Dmx) is appended to returned value
	DMN	Date of minimum channel reading		7	(Dmn) is appended to returned value
	IMX	Instant (time and date) of maximum		7	
	IMN	Instant (time and date) of minimum		7	
	INT	Integral for channel		7	The time integral's time base is seconds. For other time bases apply a span or polynomial (for example, Y1=0,2.778e-4"AHrs" for hours).
	NUM	Number of samples in statistical calculation			
	Hx:y:m..nCV	Histogram x = lower limit y = upper limit	x, y ±1e18	7	Report time sampling. See "Histogram (Hx:y:m..nCV)" on page 86.

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 continuous rapid scanning).

At the report time as determined by the report schedules, the statistical summary is reported. If no sample has been taken before the reporting time, an error is reported (9999.9).

Note Statistical options are not valid in alarms. If you want to alarm on a statistical value then use a channel variable (nCV) to pass the statistical value to the alarm.

Table: DT800 Channel Options (sheet 4 of 5)

Category	Channel Option	Mutual Exclusions	Function	Range of Option (n)	Order of Application	Comment	
Variables See "Channel Variables (nCV)" on page 92.	**=*n*CV**		Assign channel reading to channel variable. That is, ***n*CV** = reading	1 to 500	8	Channel variables are like memory registers in a calculator. You can assign them directly (for example, **1CV=2.5**), or assign a channel reading to the variable at scan time (for example,**1V(=7CV**). You can also read the contents of a variable, modify it and then replace it with the modified value. 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. That is, ***n*CV** = ***n*CV** + reading	1 to 500	8		
	-=*n*CV		Subtract channel reading from channel variable. That is, ***n*CV** = ***n*CV** – reading	1 to 500	8		
	***=*n*CV**		Multiply channel variable by channel reading. That is, ***n*CV** = ***n*CV** * reading	1 to 500	8		
	/=*n*CV		Divide channel variable by channel reading. That is, ***n*CV** = ***n*CV** ÷ reading	1 to 500	8		
Destination	**NR**		No return		9	Channels tagged with **NR** are not returned to the host computer. Useful for channels that are to be logged but not forwarded in real time to a computer.	
	NL		No log (cannot be used in alarms)		9	Channels tagged with **NL** are not logged, but they are returned to the host computer.	
	W		Causes DT800 to treat the channel as a **working channel** (also known as an **intermediate channel**)		9	A working channel's data is not returned or logged (unless the working switch **/W** is on). See examples on page 95.	
Output Data Format	**FF*n***		fixed-point n=decimal places	0 to 7	9	For example, **FF2** returns 71.46 mV	Note that the number of significant digits displayed is set by P32 (see page 108).
	FE*n*		Exponential, n=significant digits	0 to 7	9	For example, **FE2** returns 7.14e1 mV	
	FM*n*		Mixed FF and FE, n=decimal places	0 to 7	9	Uses exponential format if exponent is less than –4 or greater than n	
	"*UserName*"		User-defined returned channel name text ***UserName*** — maximum 16 characters ***UserUnits*** — maximum 8 characters	ASCII text	9	***UserName*** overrides default channel name, default units are unchanged	Specifies a unique channel name that replaces the channel type text returned to host (when enabled by **/C /U /N**)
	"*UserName*~*UserUnits*"				9	***UserName*** overrides default channel name, ***UserUnits*** overrides default units	
	"*UserName*~"				9	***UserName*** overrides default channel name, no units	
	"~*UserUnits*"				9	No channel name, ***UserUnits*** overrides default units	
	"~"				9	No channel name, no units	

Table: DT800 Channel Options (sheet 5 of 5)

Part E — Logging and Retrieving Data

Logging Data

Go for quality, not quantity

The DT800 stores data acquired from input channels and calculations into its internal lithium-backed SRAM data memory, or into an inserted FLASH memory card (see Figure 18 on page 30). Data is automatically stored in a memory card if one is inserted and it has free space; otherwise data is automatically stored in the DT800's internal memory. This stored data is called **logged data**.

Data is logged in files within the DT800's file system (see "The DT800 File System" on page 79). The data from each report schedule is logged into separate files. When the logged data is later unloaded, the data is unloaded for each report schedule in turn.

The DT800 stores approximately 65,000 data values per megabyte of memory (based on 5 data items per report schedule; slightly fewer if schedules have less than 5 channels, slightly more if schedules have more than 5 channels). Therefore the DT800's 2MB internal memory can hold approximately 130,000 data values, and a 64MB memory card can hold approximately 4,160,000 data values. See also "Data Storage Capacity — Readings/MB" on page 78.

Logged data is retained in the internal memory of the DT800 when the DT800 is reset and if its main power supply is removed.

LOGON and LOGOFF Commands

You globally enable data logging by sending the **LOGON** command, and globally disable data logging by sending the **LOGOFF** command. Data logging can also be individually controlled for each of the report schedules by the **LOGON*x*** and **LOGOFF*x*** commands.

Remember The DT800's default is data logging DISabled.

LOGON	Enables logging (data and alarms) for all schedules except those whose logging state is set by **LOGOFF*x***
LOGOFF	Disables logging (data and alarms) for all schedules except those whose logging state is set by **LOGON*x***
LOGON*x*	Enables logging for schedule ***x*** (data and alarms)
LOGOFF*x*	Disables logging for schedule ***x*** (data and alarms)

Disabling Data Logging for Specific Channels

Data from all of the input channels and calculations from all of the report schedules RA, RB, RC,…RK schedules is logged after the general **LOGON** command is sent to the DT800.

Data from all of the input channels and calculations from a particular report schedule is logged if the schedule-specific **LOGON*x*** command is sent to the DT800.

You can also disable data logging for specific channels and calculations within report schedules by including the **NL** (No Log) channel option in their channel lists — see page 75. Data is still returned to the host computer for channels with the **NL** channel option.

See also the **W** (Working) channel option on page 75.

Logging to Memory Card

As Figure 60 shows, the DT800 always gives priority to an inserted memory card as the logging destination (if the card has free space). In other words:

- Without a memory card, the DT800 logs data into its internal memory.
- With a memory card inserted, the DT800 automatically logs data into the card first, then into internal memory if the card fills (see "What Happens When Memory is Full?" on page 77).

It's not possible to direct data to particular memory card.

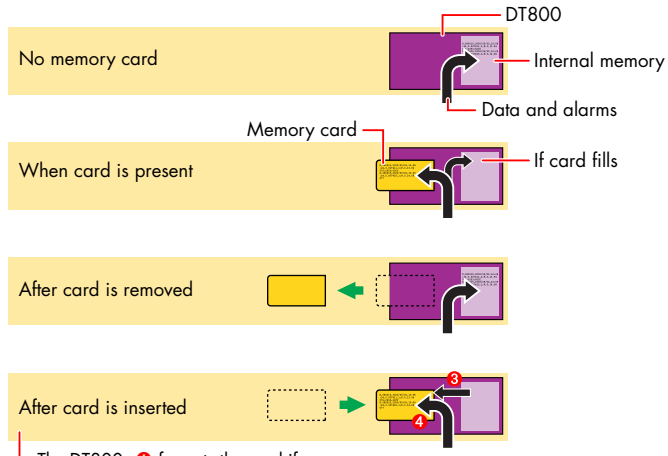


Figure 60 Destinations — logged data and alarms

Automatic Card Formatting

Whenever a memory card is inserted, the DT800 first checks that the card is formatted. If not, it automatically carries out the formatting operation.

Inserting and Removing a Memory Card

Insert a memory card into the DT800, or remove it, only when the Card Busy LED is not illuminated (see Figure 17 on page 28).

Warning Removing the memory card while the DT800 is writing to the card can cause the data to be corrupted or lost.

Note the following:

- If data is being logged into internal memory when a memory card is inserted, all data is transferred to the memory card (then internal memory is cleared) and logging continues into the memory card.
- If data is being logged into an inserted memory card when the memory card is removed, data logging continues into internal memory.

You can use this functionality to collect data from one or more DT800s:

- A memory card can be temporarily inserted into a DT800 to collect the data logged in internal memory, then removed to carry the data to a computer. (The data is automatically deleted from the DT800's internal memory after it has been transferred to the card.)
- Similarly, a memory card can be inserted into several DT800s in turn to collect the data logged in each DT800's internal memory (automatically deletes the data from the individual DT800s). Each DT800 creates a directory named **SN*xxxxxx*** (where ***xxxxxx*** is the DT800's serial number) in the root directory of the memory card, and transfers its logged data into that directory.

This is covered in more detail in "Retrieving Logged Data — Memory Card Transfer" (page 81).

Appending a Job's Data

As stated above, if you remove an inserted memory card during data logging, the DT800 simply redirects logging to its internal memory. Then if you re-insert the same card, the data logged into internal memory while the card was removed is transferred to the card (and cleared from internal memory) and appended (added) to the earlier data.

In general terms, if a card contains data from a job with the same name as the DT800's current job, when you insert the card the data from internal memory is appended to that job's existing data in the card.

Never in Both Places

From the above, you'll see that a job's logged data and alarms can only be in one place: either in internal memory, or on the card — never in both storage locations.

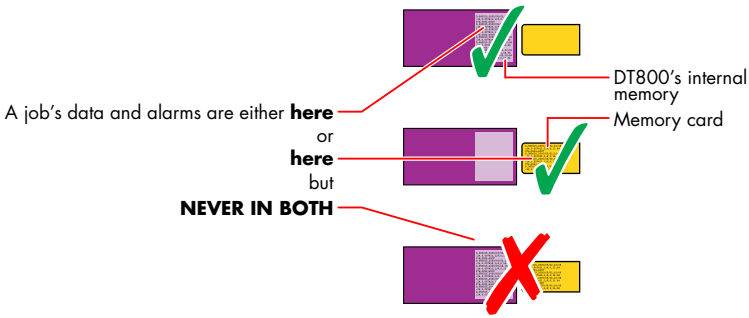


FIGURE 62 The DT800 organizes your data and alarms

Data Storage Issues

What Happens When Memory is Full?

If the DT800's data memory fills during logging, it has two ways of handling newly-acquired or calculated data:

- The DT800 can stop logging new data — see "Stop-When-Full Mode (/o)" below.
- The DT800 can progressively replace old logged data with new data — see"Overwrite Mode (/O)" below.

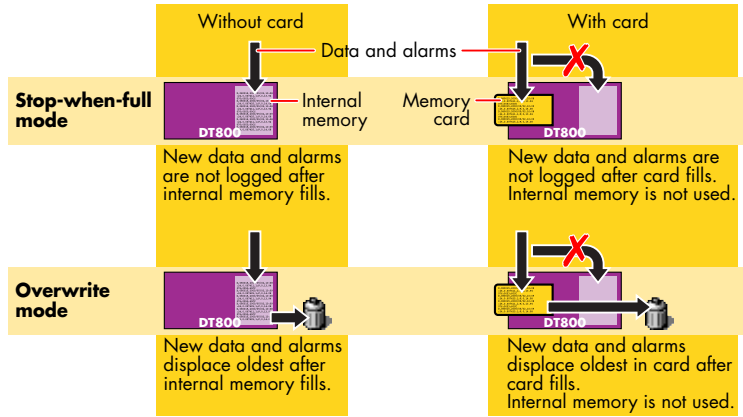


FIGURE 62 Data storage modes

Stop-When-Full Mode (/o)

In stop-when-full mode (the DT800's default), logging simply stops (but remains enabled) when the DT800's data memory becomes full. Existing data is retained and new data is discarded. This mode is enabled by the **/o** switch.

Although data logging stops, the return of latest/new data to the host computer continues if this is enabled (**/R** switch**).**

If a memory card is present, the DT800 fills the card then stops logging.

Overwrite Mode (/O)

In overwrite mode, the oldest data in the DT800's internal memory is overwritten (replaced) by the latest data when the memory becomes full. This circular memory mode is enabled by the **/O** switch.

If a memory card is present, all data logging and overwriting occurs on the card only — data is never logged to the DT800's internal memory when a memory card is present and the DT800 is in overwrite mode.

Changing Storage Modes

You can alternate between the stop-when-full and overwrite storage modes during a logging session (by sending the appropriate switch command — **/O** or **/o**). If you do this, be aware of the following:

Before Memory is Full

If you swap the DT800's storage mode before memory fills, logging continues in the new mode without data loss.

After Memory is Full

If you change the DT800 from stop-when-full mode to overwrite mode (send **/O**) after memory (internal, or card if present) has filled, data logging simply proceeds in overwrite mode. Of course, this causes a gap in the logged data (the period between memory becoming full and overwrite mode being enabled).

If you change the DT800 from overwrite mode to stop-when-full mode (send **/O**) after the memory has filled, data logging stops immediately because memory is full. The internal memory does not become available again until you reset the DT800, or you clear all data from the memory card.

Efficient Storage of Time and Date

The DT800 automatically includes a timestamp and datestamp with every data record. But it's also possible to include the time and date channel types (internal channels **T** and **D** — see page 68) in a schedule. If you do this, the time and date are included in each record just like other data, and so each record contains the time and date twice.

Therefore it's more memory-efficient not to include the **T** and **D** channels in report schedules. Simply rely on the timestamps and datestamps that the DT800 automatically generates.

Storage Status

You can check the amount of data stored and the amount of free space in the DT800's internal memory and memory card with the following commands and system variables:

STATUS	Line 6 — internal memory kB free/stored data Line 7 — memory card kB free/stored data (or **STATUS6** and **STATUS7**)		See "STATUS Commands" on page 122.
1SV	Internal memory	kB free	See "System Variables" on page 69.
2SV		kB stored data	
3SV	Memory card	kB free	
4SV		kB stored data	

Data Storage Capacity — Readings/MB

Use the following table to estimate how many readings per megabyte of data storage you can expect the DT800 to log for each schedule it is running:

Channels per Schedule	Readings per MB of DT800 Storage (approximate)
1	14,500
2	25,800
5	48,500
10	68,500
20	86,300
40	99,100

When you send a job that contains more than one schedule, use the table to estimate the number of readings/MB for each individual schedule, then add these together for an overall approximation.

See also "Storage Capacity" on page 30.

Halt and Go During Data Logging

While data logging is in progress, data from each report schedule is progressively stored into its respective data log file (DATA_A.DXD, for example — see Figures 63 and 64) in the DT800's internal memory (or memory card if one is inserted). Whenever a halt command (**H**alt all report schedules) is issued during a data logging session, a **discontinuity record** is written into all currently-open data log files. Similarly, whenever a **H*x*** command (halt Report Schedule ***x***) is issued during a data logging session, a discontinuity record is written into the currently open file for that report schedule.

In unloaded data, a discontinuity record looks like a normal data record except that all data items are set to zero and its record index (see Figure 10, page 22) is set to 4:

```
D,080435,"JOB1",2001/05/17,10:32:23,0.118530,4;A,0,0.00000,
0.00000,0.00000,0.00000
```

The discontinuity record indicates that a break occurred in the acquisition of data and is included in data subsequently unloaded from memory.

Whenever a **G** command (**G**o all report schedules), or a **G*x*** command (**G**o report schedule ***x***) is issued after a halt command, data logging resumes in the currently-open data log file(s).

Deleting Logged Data

You can delete logged data from memory at any time with the following commands:

DELDATA	Deletes the current job's data from internal memory and memory card	Job names, directory structures, programs and alarms are not erased.
DELDATA"*JobName*"	Deletes only ***JobName***'s data from internal memory and memory card	
DELDATA*	Deletes data for all jobs from internal memory and memory card	
FORMAT"B:"	Reformats the internal memory, removing all logged data and alarms. See also the DT800 Resets table (page 118)	
FORMAT"A:"	Reformats an inserted memory card, removing all logged data and alarms. See also the DT800 Resets table (page 118).	

See also "Summary — Delete Commands" on page 185 for a complete list of DT800 delete commands.

When you use any of the **DELDATA** commands, the appropriate data log files are deleted and logging continues into new files.

Resetting the DT800 by a soft reset or a firm reset (see page 118) does not delete logged data.

The DT800 File System

The DT800 uses a DOS-style file system for storing logged information (data and alarms) and system information. The file system is transparent to the user, and it's not necessary to know about it for general use of the DT800, including when logging data to memory cards and unloading data using the DT800.

But if you intend retrieving data from memory cards using a PCMCIA slot in your computer (see "Retrieving Logged Data — Memory Card Transfer" on page 81), you'll benefit from some knowledge of the DT800 file system.

The internal memory of the DT800 and the memory in memory cards is completely managed by the file system. The internal memory is configured as **drive B**, and the memory card is configured as **drive A** (Figure 18 on page 30).

You can see the directory structure of the DT800's internal memory by executing a **`DIRTREE"B:"`** command from either DeLogger's text window or DeTransfer running on a connected computer. Similarly, you can see the directory structure of an inserted memory card by executing a **`DIRTREE"A:"`** command.

Directory Structure of Internal Memory

Figure 63 shows the directory structure of a typical DT800's internal memory.

The data log files are stored in an HFS (hierarchical file structure) as follows:

- within the root directory there is a JOBS subdirectory — ❶ in Figure 63
- within the JOBS subdirectory there is a subdirectory for each of the jobnames entered into the logger (JOB1, JOB2,…) — ❷ in Figure 63
- within each job subdirectory there is a subdirectory for each of the report schedules in the job (A, B, C,…) — ❸ in Figure 63
- within each schedule subdirectory there is
  - a logged data file **DATA_*S*.DXD**, where ***S*** is the schedule letter (A, B,…K) and **.DXD** is the data file extension
  - a logged alarms file **ALARMS_*S*.DXA**, where ***S*** is the schedule letter (A, B,…K) and **.DXA** is the alarms file extension.

Other directories and files are explained in Figures 63 and 64.

Directory Structure of Memory Cards

The same directory structure is used in DT800 memory cards, except that it's all contained in a higher order directory (folder) named **SN*xxxxxx***, where ***xxxxxx*** is the serial number of the DT800 — see Figure 64. If the memory card has been inserted into more than one DT800, the card contains a serial number directory for each DT800.

To retrieve data, memory cards can be read directly by a Windows 95, 98 or 2000 computer with a PCMCIA slot/drive[8]. If the computer has the appropriate PCMCIA drivers loaded and enabled (consult your computer manuals or supplier), the slot functions as a disk drive in Windows Explorer where you can see the memory card as if it were a floppy disk in a floppy drive. Alternatively, your *dataTaker* supplier can provide a USB FLASH **Memory Card Reader** to install onto your computer. The card reader also appears as a disk drive in Windows Explorer.

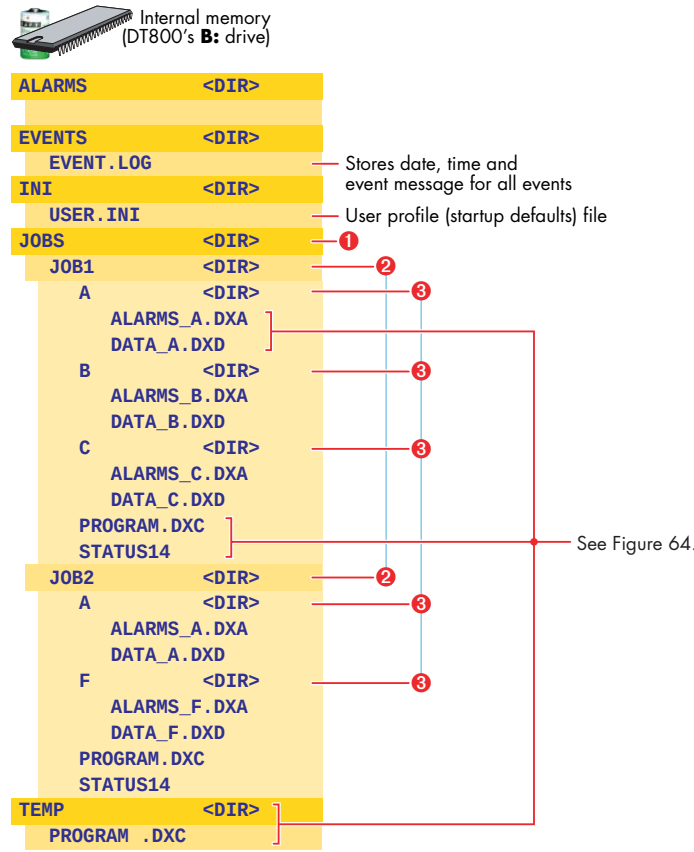


FIGURE 63 Typical structure — DT800 internal data memory (send **`DIRTREE"B:"`**)

[8] Provided that the computer manufacturer has supplied an ATA Flash driver for the PCMCIA slot. Refer to the computer's documentation.

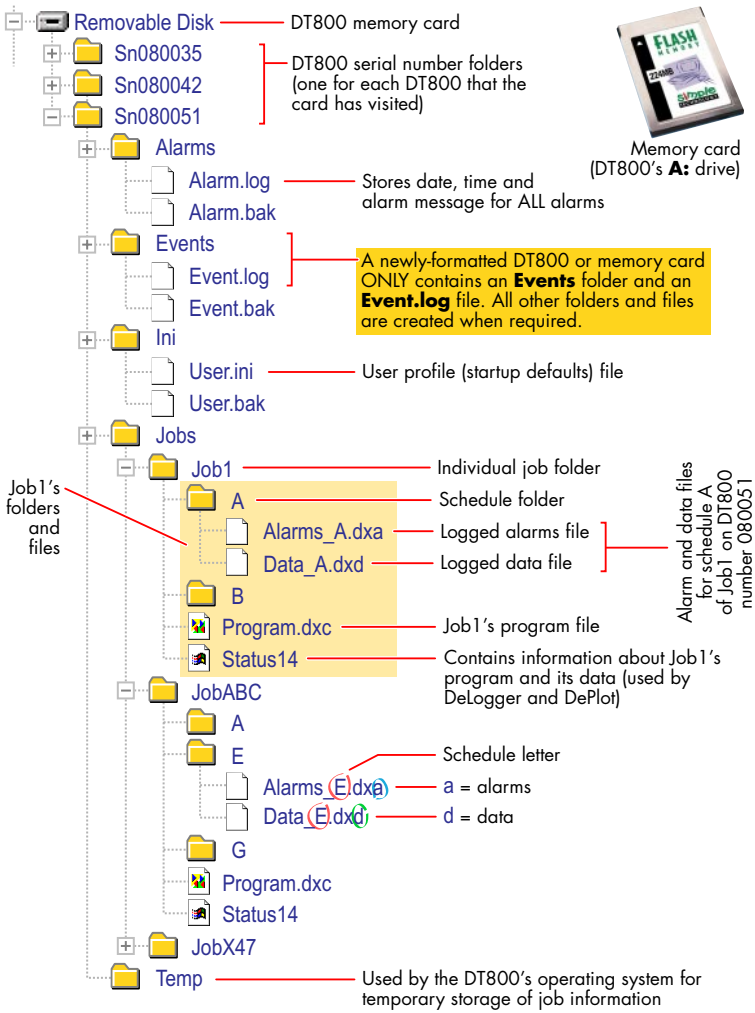


FIGURE 64 Typical structure — DT800 memory card (Explorer-style view)

Data Files are Text Files

Each schedule folder contains a logged data file and a logged alarms file (unless the schedule has not yet logged a data record or an alarm record). These are text files, which you can open in most text editors and spreadsheets (Microsoft Notepad, WordPad, Excel, Word,...).

Logged data files and alarm files contain the individual records that were logged during the data logging session. For example:

```
D,080217,2001/05/14,00:36:44,0.006995,1;A,0,0.00434,0.00161,
                                                  270.0992,387.7028
D,080217,2001/05/14,00:36:45,0.029578,1;A,0,0.00194,0.00345,
                                                  270.0883,387.7035
D,080217,2001/05/14,00:36:46,0.007483,1;A,0,0.00108,0.00295,
                                                  270.0870,387.7004
D,080217,2001/05/14,00:36:47,0.007361,1;A,0,0.00315,0.00255,
                                                  270.1103,387.7155
D,080217,2001/05/14,00:36:48,0.030432,1;A,0,0.00212,0.00251,
                                                  270.1032,387.6875
D,080217,2001/05/14,00:36:49,0.007483,1;A,0,0.00233,0.00147,
                                                  270.0926,387.6985
D,080217,2001/05/14,00:36:50,0.007605,1;A,0,0.00296,0.00230,
                                                  270.0982,387.6938
```

⇓

Size of Data Files

You can see the size of logged data and logged alarms files by sending the **DIRTREE"B:"** command (internal memory) or **DIRTREE"A:"** command (memory card) to the DT800, or by inserting a DT800 memory card into a computer's PCMCIA slot or Memory Card Reader and looking at the file size in Windows Explorer.

Alternatively, "Storage Status" on page 78 explains how you can get similar information.

Retrieving Logged Data

You can retrieve logged data from the DT800 by

- unloading the data to a host computer through one of the DT800's communications interfaces (this does not remove the data from the internal memory or memory card)
- removing an inserted memory card from the logger and reading this in a computer with a PCMCIA slot or a *dataTaker* Memory Card Reader
- temporarily inserting a memory card into the DT800 to extract data stored in internal memory, then removing the card and reading this in a computer (this does remove the data from internal memory).

See also "Logging and Retrieving Alarms" beginning on page 103.

Retrieving Logged Data — Memory Card Transfer

When a memory card is inserted into the DT800 to extract the data logged in internal memory, the following happens automatically in this order (see Figure 60 on page 76):

1. If the card is unformatted, the DT800 formats it.
2. All data in the DT800's internal memory is copied to the memory card, into a directory named with the DT800's serial number (see Figure 64 on page 80). The data is then automatically deleted from the DT800's internal memory. In other words, inserting a memory card into a DT800 removes all data from the DT800.

   You can use this automatic facility as follows:
   - If you're carrying a card onto which you want to put all of the DT800's logged data, just insert the card, wait for the Card Busy LED to stop flashing, then remove the card again. You can also insert the same memory card into a number of DT800s to remove their data. Each DT800 creates a directory (named with its serial number) on the card to store the removed data.
   - Once the data has been transferred to the memory card, you can insert the card directly into a computer's PCMCIA/PC Card socket. The card then appears as a disk drive — see "Directory Structure of Memory Cards" (page 79) — and you can simply open data files on the card directly into most text editors and spreadsheets (Microsoft Notepad, WordPad, Excel, Word,…).
3. If the memory card contains a suitable ONINSERT.DXC file (see "ONINSERT.DXC" on page 115), the ONINSERT.DXC startup job is transferred from the card to the DT800 and run (becomes the DT800's current job).

Retrieving Logged Data — Comms Unload

Commands for retrieving data from a DT800's internal memory or card to a host computer using one of the DT800's communications interfaces are presented in "Unload Commands" next, and summarized in the table on page 186.

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 DT800 automatically sets these switches to their previous state on completion of the unload.

Note Logged data is not cleared from the DT800 when you use this "comms interface / unload commands" method of retrieving data. You can unload the same logged data using one of the DT800's communications interfaces as many times as you want (until you purposefully delete it, of course — see "Deleting Logged Data" on page 78).

Conversely, logged data is cleared from the DT800 when you use the "temporarily insert a memory card" method of retrieving data (discussed earlier).

Always Fixed-Format Mode

When logged data is unloaded, it's always returned to the host computer in fixed-format mode — see page 21, and Figure 65 (page 84). (Real-time data can be returned in either free- or fixed-format mode.)

Order of Unload

Logged data is always unloaded schedule-by-schedule in the order A, B, C,…K unless you specify a particular schedule — see "Unload Commands" below).

Unload Commands

To retrieve logged data from the DT800 by means of one of its comms interfaces you use the **unload commands** below. There are three formats of unload commands:

U commands	Unload **all** data from beginning to end of data store	See "The U Unload Commands" below.
U( ) commands (round brackets / parentheses)	Unload data for a **period** defined by a beginning and end date and time in the **format** of the DT800's current date format (P31) and time format (P39) settings	See "The U( ) Unload Commands" below.
U[ ] commands (square brackets)	Unload data for a **period** defined by a beginning and end date and time in the **strict format** of YYYY/MM/DD,hh:mm:ss,0.ssssss, which overrides the current date format (P31) and time format (P39) settings	See "The U[ ] Unload Commands" on page 83.

The U Unload Commands

The **U** unload commands return all logged data — that is, from the first record to the last record — for a specified job and report schedule.

Here are the DT800's **U** commands:

U	Returns all of the current job's logged data in the order of report schedule A–K
Ux	Returns all of the current job's logged data for report schedule ***x***
U"*JobName*"	Returns all logged data for ***JobName*** in the order of report schedule A–K
U"*JobName*"*x*	Returns all logged data for ***JobName*** for report schedule ***x***

Examples — U Commands

The command

U

instructs the DT800 to return all of the current job's data, schedule-by-schedule.

The command

UC

instructs the DT800 to return all of the current job's data for report schedule C only.

The U( ) Unload Commands

The **U( )** unload commands return logged data for a specified job and report schedule, for a period of time designated by a beginning and end ***time*** and ***date*** that must be specified in the DT800's current date format (Parameter 31) and time format (Parameter 39) settings.

Here are the DT800's **U( )** commands:

U	Returns the current job's logged data in the order of report schedule A–K
U(*from*)	Returns the current job's logged data starting from **BEGIN**, ***time*** or ***time,date***
U(*from*)(*to*)	Returns the current job's logged data starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***
Ux	Returns the current job's logged data for report schedule ***x***
Ux(*from*)	Returns the current job's logged data for report schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date***
Ux(*from*)(*to*)	Returns the current job's logged data for report schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***
U"*JobName*"	Returns ***JobName***'s logged data in the order of report schedule A–K
U"*JobName*"(*from*)	Returns ***JobName***'s logged data starting from **BEGIN**, ***time*** or ***time,date***
U"*JobName*"(*from*)(*to*)	Returns ***JobName***'s logged data starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***
U"*JobName*"*x*	Returns ***JobName***'s logged data for report schedule ***x***
U"*JobName*"*x*(*from*)	Returns ***JobName***'s logged data for report schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date***
U"*JobName*"*x*(*from*)(*to*)	Returns ***JobName***'s logged data for report schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***

where

from	can be • **BEGIN** — start from first data point logged • ***time*** — start from first data point logged at or after this time today • ***time,date*** — start from first data point logged at or after this time and date	Both the ***time*** and ***date*** must be specified in the currently-defined format for date (P31) and time (P39).
to	can be • **END** — finish with last data point logged • ***time*** — end with last data point logged prior to this time today • ***time,date*** — end with last data point logged prior to this time and date	

Note **BEGIN** and **END** used here are not the same as the **BEGIN** and **END** keywords used to indicate the start and end of a DT800 job.

Examples — U( ) Commands

The command

UA(BEGIN)(11:15)

instructs the DT800 to unload schedule **A**'s logged data from the first data point stored (**BEGIN**) until **11:15**.

The command

UB(12:00,19/1/2000)(12:05,20/1/2000)

instructs the DT800 to unload schedule **B**'s logged data starting at **12:00** on the **19/1/2000** and ending at **12:05** on the **20/1/2000**.

Examples — U( ) Shortcuts

The command

UC(13:21)(13:21)

instructs the DT800 to return only the current job's data for schedule **C** that was logged during the minute **13:21** (that is, from 13:21:00 to 13:21:59).

The command

UD(13)(13)

instructs the DT800 to return only the current job's data for schedule **D** that was logged during the hour **13** (that is, from 13:00:00 to 13:59:59).

The U[ ] Unload Commands

The **U[ ]** unload commands return logged data for a specified job and report schedule, for a period of time designated by a beginning and end ***time*** and ***date*** that must be specified in the strict format **YYYY/MM/DD,hh:mm:ss,0.ssssss** (ISO date format), which overrides the current date format (Parameter 31) and time format (Parameter 39) settings.

Here are the DT800's **U[ ]** commands:

U	Returns the current job's logged data in the order of report schedule A–K
U[*from*]	Returns the current job's logged data starting from ***time*** or ***time,date***
U[*from*][*to*]	Returns the current job's logged data starting from ***time*** or ***time,date*** and ending with ***time*** or ***time,date***
U*x*	Returns the current job's logged data for report schedule ***x***
U*x*[*from*]	Returns the current job's logged data for report schedule ***x*** starting from ***time*** or ***time,date***
U*x*[*from*][*to*]	Returns the current job's data for schedule ***x*** starting from ***time*** or ***time,date*** and ending with ***time*** or ***time,date***
U"*JobName*"	Returns ***JobName***'s logged data in the order of report schedule A–K
U"*JobName*"[*from*]	Returns ***JobName***'s logged data starting from ***time*** or ***time,date***
U"*JobName*"[*from*][*to*]	Returns ***JobName***'s logged data starting from ***time*** or ***time,date*** and ending with ***time*** or ***time,date***
U"*JobName*"*x*	Returns ***JobName***'s logged data for report schedule ***x***
U"*JobName*"*x*[*from*]	Returns ***JobName***'s logged data for report schedule ***x*** starting from ***time*** or ***time,date***
U"*JobName*"*x*[*from*][*to*]	Returns ***JobName***'s logged data for report schedule ***x*** starting from ***time*** or ***time,date*** and ending with ***time*** or ***time,date***

where

from	can be • ***time*** — start from first data point logged at or after this time today • ***time,date*** — start from first data point logged at or after this time and date	Type ***time*** and ***date*** in fixed-format ISO- style (see examples below).
to	can be • ***time*** — end with last data point logged prior to this time today • ***time,date*** — end with last data point logged prior to this time and date	

Examples — U[ ] Commands

Here's the **U[*from*]** command showing valid forms of fixed-format mode date and time:

```
U[2000/07/26,00:00:01,0.250366]
U[2000/07/26,00:00:01]
U[2000/07/26]
```

Here's the **U[*from*][*to*]** command showing valid forms of fixed-format mode date and time:

```
U[2000/07/26,00:00:01,0.250366][2000/07/26,00:00:01,0.750244]
U[2000/07/26,00:00:01][2000/07/26,00:00:01,0.750244]
U[2000/07/26][2000/07/26,00:00:01,0.750244]

U[2000/07/26,00:00:01,0.250366][2000/07/26,00:00:01]
U[2000/07/26,00:00:01][2000/07/26,00:00:01]
U[2000/07/26][2000/07/26,00:00:01]

U[2000/07/26,00:00:01,0.250366][2000/07/26]
U[2000/07/26,00:00:01][2000/07/26]
U[2000/07/26][2000/07/26]
```

Labelling the End of Unloaded Data

As Figure 65 shows, the DT800 automatically includes special records when it returns logged data to the host computer. These records signify

- the end of the unload of an individual schedule's data — **end-of-schedule record**
- the end of a complete unload (may contain one or more schedules) — **end-of-unload record**.

The records are added even if the unload process is aborted before completion (by sending the **Q** command — see next topic).

See also Figure 10 (page 22), which describes the components of typical fixed-format records in more detail.

Quitting an Unload

Stop an unload operation by sending the Quit Unload command

```
Q
```

to the DT800.

There may be a short delay between sending the command and the return of data actually stopping. This is because the output buffer of the DT800 is generally full during an unload and these records have still to be returned to the host. (Sending **Q** actually stops the copying of stored data into the DT800's output buffer.)

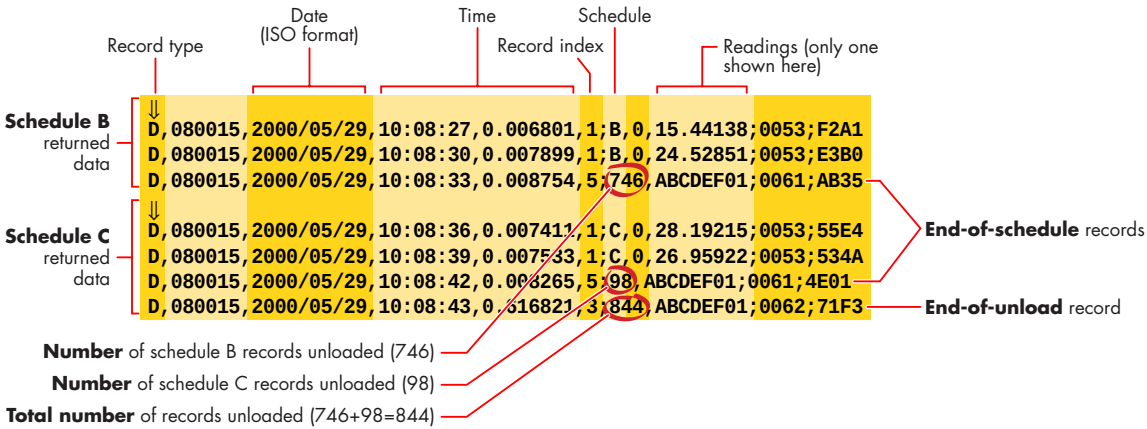


FIGURE 65 Typical unloaded data (always returned in fixed-format mode; see also Figure 10 on page 22)

Part F — Manipulating Data

Channel Options — Statistical

Useful for reducing data

Channels can be sampled frequently and a statistical summary returned at longer intervals (see ""Statistical Report Schedules" on page 49"). Statistical channels are sampled for the period between report times, and the statistical summary is generated and returned at report time.

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 category in the DT800 Channel Options table (page 74):

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)	
IMN	Instant of minimum (combines DMN and TMN)	
INT	Integral	(Int)
H*x*:*y*:*m*..*n*CV	Histogram	
NUM	Number of samples	

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

```
RA1M 3TT(AV)
```

returns

```
3TT 103.7 degC (Ave)
```

This is the average (**(AV)**) temperature returned every one minute (**RA1M**) for the type T thermocouple connected to channel 3 (**3TT**). The text (Ave) is appended to the units to indicate that the data is an average.

If statistical channels have not been sampled before they are reported, these channels report error E53 (see the DT800 Error Messages table beginning on page 197) and data is returned as 99999.9. This condition is likely to occur when the RS trigger is an event, the statistical sub-schedule has been halted, or a statistical scan interval (**RS**) is longer than the reporting time interval.

Statistical Channel Options and Multiple Reports

If statistical options are part of a multiple report schedule (see"Multiple Reports" on page 70), each option list must contain a statistical option — for example

```
4PT385(I,500,AV)(MX)(TMX)(MN)(TMN)
```

Note that the first channel option list (**(I,500,AV)**) must include all of the options required for managing and sampling the channel. This rule applies to any options above the configuration line in the DT800 Channel Options table (page 74), because the channel is sampled and scaled according to the first option list.

Statistical Channel Options and Alarms

Statistical results are tested in alarms by first assigning them to channel variables (see "Channel Variables (nCV)" on page 92).

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. See the "Statistical" category on page 74.

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. See the "Statistical" category on page 74.

Maximum and Minimum

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

The time and date of these can be reported with the **TMX**, **TMN**, **DMX**, **DMN**, **IMX** and **IMN** channel options. See the "Statistical" category on page 74.

Integration (INT)

The integration channel option (see the "Statistical" category on page 74) 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.

Example — Integration

When integration is applied to a flow rate sensor as follows

```
S5=0,0.1,0,1000"litres"
3C("Fuel Consumption",S5,INT,R)
```

the volume of the flow is returned:

```
Fuel Consumption 34.54 litres (Int)
```

The flow rate sensor with a counter output (**3C**) is scaled by a span (**S5** — see "Spans (Sn)" on page 90) and then integrated. Note that the span units have been declared as litres, which is the result after integration, although the span calibration is actually to litres per second.

Histogram (H*x*:*y*:*m*..*n*CV)

You can use the DT800 to generate a histogram (frequency distribution) of channel samples by applying the histogram channel option, which instructs the DT800 to

- divide the measured data range into a number of intervals called **classes** (see Figure 66)
- count the number of readings that occur in each class during the histogram period
- load each class count into a separate channel variable.

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

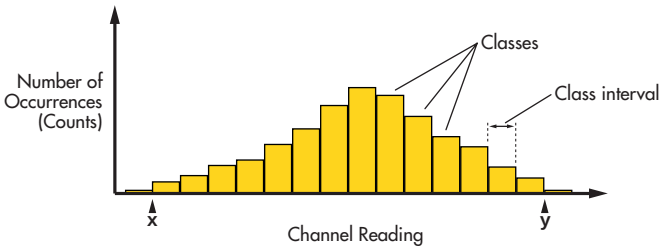


FIGURE 66 Histogram

In addition, the DT800 automatically counts the number of under-range, over-range and total readings, and stores these in three separate channel variables.

See also the Statistical category in the DT800 Channel Options table (page 71).

The format of the histogram channel option is

```
Hx:y:m..nCV
```

where

x	is the lowest channel reading of interest
y	is the highest channel reading of interest (***y*** > ***x***)
m	is the first channel variable (***m*****CV**) to store counts
n	is the last channel variable (***n*****CV**) to store counts

Three other counts are also stored by the DT800:

(*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 histogram channel option does not affect the usual reporting or logging of the channel's readings.

The number of channels that can be histogrammed is limited by the number of channel variables available (the DT800 supports a maximum of 500 CVs).

Example — Histogram

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

```
RA1S 1TT(H25.0:35.0:1..8CV)
```

This schedule generates a histogram with five temperature classes and intervals of 2°C:

1CV counts for first class (25.0 to 26.999°C interval)
2CV counts for second class (27.0 to 28.999°C interval)
3CV counts for third class (29.0 to 30.999°C interval)
4CV counts for fourth class (31.0 to 32.999°C interval)
5CV counts for fifth class (33.0 to 34.999°C interval)

It also loads cumulative data into the following three CVs:

6CV number of under-range samples (<25°C)
7CV number of over-range samples (>35°C)
8CV total counts (sum of 1..7CV)

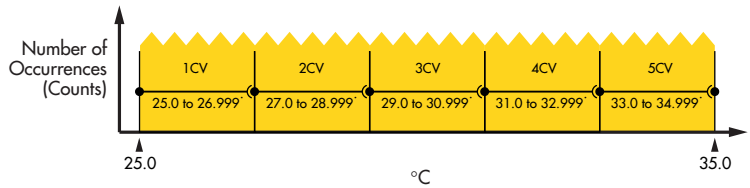


FIGURE 67 Histogram example

Then you can return and log the channel variables using another schedule such as

```
RB1H 1..8CV(R)
```

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 DT800 implements the ASTM E 1049-85 standard: Standard Practices for Cycle Counting in Fatigue Analysis.

You can carry out real-time rainflow analysis using the DT800's **RAINFLOW...** channel option, which instructs the DT800 to monitor attached strain gauges at regular intervals and reduce the resulting large quantity of data into simple cycle histograms.

The DT800 can also produce a formatted report of the accumulated cycle histograms — see "Reporting Rainflow Data — The Rainflow Report" on page 88.

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 — The Rainflow Channel Option

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 DT800's 32-bit signed Integer Variables (channel type ***n*IV**).

The **RAINFLOW...** channel option requires you to specify a maximum cycle size, 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 in terms of a percentage of ***a***
c	is the first IV in the sequence to be used for accumulating data
d	is the last IV in the sequence to be used for accumulating data

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 **5**% of 1000 (= 50) units are rejected as noise.

The number of variables allocated for the rainflow analysis must be set to the number of cycle size classes you require 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 you'll need 17 variables. 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**
	⇓	⇓	⇓
Summary data	***d*–7**	Contains the count of cycles for the last cycle size class	**33IV**
	***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.

Collecting Rainflow Data — The Rainflow Sample Rate

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 50mSec (**50T**; 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 — see page 75).

The fastest sampling rate is obtained by the schedule trigger

```
RA,FAST(options)
  3BGI(RAINFLOW:a:b:c..dIV,W)
```

See "Fast Mode Sampling" on page 52.

Reporting Rainfall Data — The Rainflow Report

You collect rainflow data 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
```

to the DT800. The DT800 returns a tabular report as illustrated below, which is normally viewed and saved in the Receive (upper) window of DeTransfer. (DeLogger software does not support the Rainflow Report.)

```
Rainflow (5% rejection)01/01/2000 00:03:43
n     IV       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 DT800, 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..28IV,W)
  RB7D 1..28IV
  LOGONB
END
```

logs the histogram data every 7 days. You can create reports manually after you download the primary cycle count data.

Example 1— 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 DT800 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
```

returns rainflow histogram data such as the following:

```
Rainflow (5% rejection)15/03/2000 00:05:22
n     IV       Range    Mean     Cycles
====================================
1     101      0        0.0      0
2     102      53       11.7     187805
3     103      105      11.5     153802
4     104      158      11.7     128263
5     105      211      11.6     65312
6     106      263      11.7     39342
7     107      316      12.3     25449
8     108      368      12.5     9612
9     109      421      13.0     3950
10    110      474      13.4     1919
11    111      526      13.6     760
12    112      579      13.8     354
13    113      632      14.2     49
14    114      684      14.9     20
15    115      737      15.1     3
16    116      789      0.0      0
17    117      842      0.0      0
18    118      895      0.0      0
19    119      947      0.0      0
20    120 >=   1000     0.0      0
====================================
Total cycles            616640
Peak/Valley mean        11.7
Max Peak                42
Min Valley              -22
Max buffered cycles     38
Valid input points %    100.00
```

Best Speed for Rainflow.

For best rainflow speed from the DT800:

- turn off house keeping (**/k**)
- select a channel type with fewer fundamental samples
- use the fast modifier
- set the gain lock to just above the maximum expected voltage (**GL20MV**)
- make the rainflow channels working channels (**W**)

For example:

```
BEGIN
  /k
  RA,FAST
    1BGI(RAINFLOW:1000:5:1..20IV,GL20MV,W)
END
```

Here are typical rainflow sample speeds:

1BGI	300 Hz	**1V**	480 Hz
2BGI	180 Hz	**2V**	350 Hz
4BGI	100 Hz	**4V**	220 Hz
8BGI	56 Hz	**8V**	120 Hz
12BGI	33 Hz	**12V**	76 Hz

Channel Options — Scaling

Getting sophisticated

The DT800 provides many different tools for scaling channel readings:

- automatic scaling
- channel factor
- intrinsic functions
- spans
- polynomials
- channel variables

It can also carry out calculations on channel data — see "Calculations (Expressions)" on page 94. In addition, you can combine scaling and calculation methods — see "Combining Methods" on page 95.

Automatic Scaling

The DT800 automatically scales measurements according to the channel types you specify. In other words, whatever the output of a particular sensor, its measurements are always converted (scaled) to engineering units (volts, amps, ohms, °C,...) according to the channel type you specify in the channel ID in the channel list — this is why you specify a channel's channel type.

Channel Factor (f.f)

Many input channel types support a channel factor (a floating-point number) as a channel option — see the Channel Factor column of the DT800 Channel Types table beginning on page 63.

The channel factor channel option usually provides linear scaling. However, for some channel types, the channel factor has a dedicated purpose and therefore cannot be used for scaling. See the table in "A Special Channel Option — Channel Factor" on page 70.

Example — Channel Factor

■ In the channel list

```
1V 1V(101.0)
```

the first **1V** returns true millivolts, and **1V(101.0)** includes the channel factor **(101.0)**, which returns the reading multiplied by **101.0** in units of millivolts as follows:

```
1V 2.543 mV
1V 256.84 mV
```

Here, the channel factor could be the attenuation of an input voltage attenuator network.

Intrinsic Functions (F*n*)

The DT800 has seven inbuilt and mutually exclusive intrinsic functions that are applied as channel options (see page 73). 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)

Channels with an intrinsic function applied return data followed by the text modifier listed above. If you place more than one intrinsic function in a channel's channel option list, only the last is applied.

Example — Intrinsic Function

■ The channel list

```
1V(F2)
```

returns the square root of the reading as follows:

```
1V 455.6 mV (Sqrt)
```

Spans (S*n*)

Spans are used to define calibrations for linear sensors. They are particularly suited to 4–20mA current loop inputs. A total of 50 spans and polynomials can be defined. Spans are applied to a channel as a channel option (see page 73).

A span is defined as follows:

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

where

n	is the span number (**1** to **50**) — spans and polynomials must not both use the same number.
a and ***b***	are the physical coordinates of two points on the calibration line — see Figure 68.
c and ***d***	are the signal coordinates of the two points on the calibration line — see Figure 68. If not specified, ***c*** and ***d*** default to **0** and **100** respectively, which is useful for 4–20mA current-loop channels (channel type **L**).
Text	replaces the channel's default units text.

A single span definition may be applied to any number of channels in any schedules or alarms.

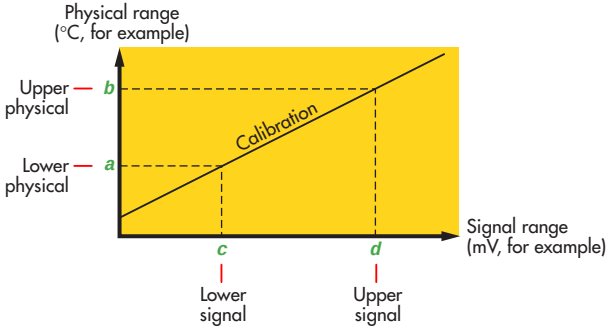


FIGURE 68 Span coordinates

Using Spans — Guidelines

- When creating a DT800 program, define any spans (and polynomials) ahead of the schedules and alarms.
- A span's number is not related to any channel number — they do not have to match. Use any span for any channel.
- Use one span for each type of sensor/transducer

Example — Span

The commands

```
S17=0,300,100,1000"kPa"
RA5M 1V(S17,"Boiler pressure")
```

define the span **S17** then use it in a schedule, which instructs the *dataTaker* to return data in the form

```
Boiler pressure 239.12 kPa
Boiler pressure 247.33 kPa
⇓
```

Polynomials (Y*n*)

Polynomials are used to define calibrations for non-linear sensors. A total of 50 spans and polynomials can be defined. Polynomials are applied to channels as a channel option (see page 73).

The DT800 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 channel reading, and the k's are coefficient terms.

A polynomial is defined as follows:

```
Yn=k0,k1,k2,k3,k4,k5"Text"
```

where

n	is the polynomial number (**1** to **50**) — polynomials and spans must not both use the same number.
k0, k1,...	are the coefficient terms — only the coefficient terms up to the required order need to be entered.
Text	replaces the channel's default units text.

Simple scale and offset corrections are also possible (internally, the DT800 treats spans as a first order polynomial). The coefficient terms of a polynomial are evaluated by a least square regression. Various statistical programs are available for this purpose. Some nonlinear sensors are supplied with their calibration polynomial. A single polynomial may be applied to any number of channels in any schedules or alarms.

Example — Polynomial

The commands

```
Y18=25.5,0.345,0.0452"Deg C"
RA5M 1V(Y18)
```

define the polynomial **Y18** then use it in a schedule, which instructs the DT800 to return data in the form

```
1V 44.35 degC
1V 43.89 degC
⇓
```

Thermistor Scaling (T*n*)

The DT800 has channel types for many 2-wire YSI thermistors (Yellow Springs Instruments www.ysi.com). For other thermistor types, the DT800 supports thermistor scaling — the conversion of a resistance reading to a temperature. The DT800 does the conversion from resistance to temperature using

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

To apply thermistor scaling, firstly obtain the constant terms a, b and c from the thermistor manufacturer, then define a thermistor conversion by sending a thermistor command

```
Tn=a,b,c"Temperature units"
```

(***n*** is the thermistor conversion number — 1 to 20).

Finally, apply the conversion by including a resistance channel ID with the thermistor channel option

```
mR(Tn)
```

in a schedule (where ***m*** is the number of the channel to which the thermistor is attached).

Example — Thermistor Scaling

The commands

```
T1=26.5,1.034,-0.0085"K"
RA5M 3R(T1,"Solvent temp.")
```

define the thermistor conversion **T1** and its use in a schedule, which instructs the DT800 to return data in the form

```
Solvent temp. 356.21 K
Solvent temp. 356.35 K
⇓
```

See also "Thermistors" on page 144.

Channel Variables (*n*CV)

Channel variables are memory locations (registers) for holding floating-point data such as channel readings and the result of expressions. The DT800 has 500 channel variables, identified as **1CV** to **500CV**. They can be used

- to receive
  - the data from input channels at the time of scanning
  - the results of calculations
- to pass input channel data and results of calculations to
  - the host computer
  - the DT800's internal memory and memory cards
- to pass input channel data to expressions (see "Calculations (Expressions)" on page 94)
- as the test data or as the setpoints in alarms
- to transfer statistical data to alarms
- for temporary storage of date and time, and for time-based calculations such as elapsed time, down time and rates
- to trigger report schedules.

Assigning Readings to Channel Variables

A channel variable receives (is assigned) the current value of any input channel by including the channel variable in a channel option list — see page 75 (DT800 Channel Options table). For example

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

returns the voltage for channel 1 and assigns (overwrites) the voltage value to channel variable **2CV**.

Channel variable assignments are made at the report time of the embracing report schedule. They are not made at the statistical sub-schedule scan time.
Where statistical results are to be tested, then channel variables provide the only means of using statistical results in alarms. For example, the program

```
BEGIN
   RS1S
   RA1M
   3TT(SD,=1CV,W)
   ALARM1(1CV>0.1)"Excessive variability"
END
```

tests the standard deviation of the temperatures read over each minute.

A schedule or an immediate scan containing a CV channel option can

- initialize a channel variable automatically
- be sent from the host computer to assign a value to a channel variable.

Reading a Channel Variable

The DT800 treats channel variables in the same way as normal input channels — you return and log the current value of a channel variable using a normal schedule command.

Arithmetic Operations

You can use one of four basic arithmetic operations (**+=**, **-=**, ***=** and **/=**) when storing input channel data into channel variables. For example, the immediate scan command

```
5V(+=1CV)
```

scans channel **5V**, sets **1CV** equal to **1CV+5V** (acts as an accumulator), and reports the value of **5V**.

Similarly, the immediate scan command

```
5V(S1,/=1CV)
```

scans channel **5V**, applies span 1 (**S1**), sets **1CV** equal to **1CV/5V(S1)** and reports the value of **5V(S1)**.

Statistical Channel Variables

When a channel variable is included as a channel option for a statistically-scanned channel, the statistical result is stored in the channel variable and not the individual readings. For example, the schedule command

```
RS5S RA10M 3V(AV,=1CV)(MX,=2CV)(MN,=3CV)
```

stores the 10-minute average, maximum and minimum into channel variables **1CV**, **2CV** and **3CV** respectively.

Results of Expressions

Channel variables can also be assigned the results of expressions (see "Calculations (Expressions)" on page 94). For example, the command

```
3CV=(1+COS(2CV))*1.141
```

evaluates the expression and assigns the result to **3CV**.

Channel Variables as Triggers

You can use channel variables as report schedule triggers. See "Trigger on Internal Event" (page 47).

Using Channel Variables

Channel variables are used in the same way as channels within schedules and alarms. Channel options can be used to modify the function and data format of channel variables. For example, the commands

```
RA1M 3CV(H:1:10:100..122CV)=SQRT(6CV+7CV)
RB1H 100..119CV
```

instruct the DT800 to evaluate the **SQRT** expression and apply the result to the histogram (**H**...) every minute (**RA1M**), and return the accumulated histogram data to the host every hour (**RB1H**)

Channel variables are not normally returned with units text, however you can define units using polynomials (page 91) or **"*UserName*~*UserUnits*"** channel options (page 75). For example

```
Y20=0,1.0"kPa"
11CV(Y20)=SQRT(4CV/6CV)
```

Channel variables can be used in alarms, both as the test value and as the setpoint(s). For example

```
ALARM1(4CV<>2CV,3CV)"[5CV=20]"
```

Channel variables are useful when comparing an input channel against several thresholds. For example

```
IF(1V(=1CV)>0.5)"Over 0.5 Volts"
IF(1CV>0.6)"Over 0.6 Volts"
IF(1CV>0.7)"Over 0.7 Volts"
```

where channel **1V** is sampled once (rather than risking different values) and tested against a number of setpoints.

Working Channels Hide CV Data

When input channels or channel variables are used in intermediate steps of a program, then the **W** channel option (see page 75) can declare these as **working channels** and prevent data being returned or logged. See examples on page 95.

During program debugging, the **W** option can be overridden by the **/W** switch to return intermediate data.

Naming Channel Variables

You can use **("*CVname~Units*")** to name a channel variable. For example

2CV("WindSpeed~km/h")

CVname may have a maximum of 16 characters; ***Units*** may have a maximum of 8 characters.

Channel Variables Report

Send the command

NAMEDCVS

to instruct the DT800 to return a summary of all named channel variables in use, for example

CV	S	CV Name	Value	Units
14	A	Wind Speed 01	14.4	km/h
15	B	Wind Speed 05	19.6	km/h
98	F	Wind Run	14400	m/h
3	F	Temp 01	145.7	deg C

where

CV	is the channel variable number ***n*** (***n*CV**)
S	is the schedule identifier
CV Name	is the channel variable's name, if any (see "Naming Channel Variables" on page 93)
Value	is the value stored in the channel variable when the **NAMEDCVS** command was sent
Units	is the units defined for the channel variable

Calculations (Expressions)

At report time only

The DT800 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 operators:

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

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

The operator precedence is

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

Expressions evaluate left to right, but parentheses can be used to define a particular order of evaluation. Parentheses can be nested.

Expressions are evaluated at the report time of the embracing schedule, and in the order in which they occur within the schedule.

Conditional Calculations

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

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

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

END

LOGON G
```

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
```

Part G — Alarms

Alarm Concepts

Limits, tests and actions

DT800 **alarms** allow you to make decisions based on the magnitude of DT800 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**.

You instruct the DT800 to carry out actions when an alarm tests true. These actions can be setting the DT800's digital state outputs, issuing messages, or executing commands to change the DT800's operation.

There are two types of alarm commands (Figure 69):

- the **ALARM** command — acts once on the transition of the alarm test from false to true ("single-shot" alarm)
- the **ALARMR** command — acts **R**epeatedly at the schedule interval while the alarm tests true ("repeating" alarm)

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. If an alarm tests true, the alarm's actions are executed before control passes to the next process in the schedule.

The general format of the single-shot alarm command is

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

and of the repeating alarm command is

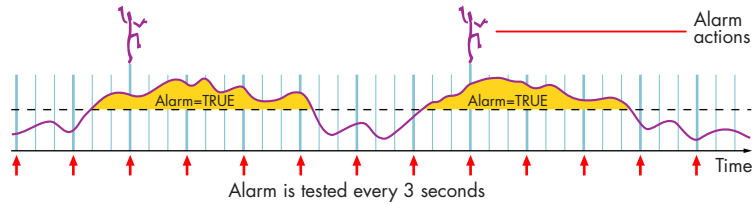
ALARMR*n*`(`*test*`)`*digitalAction*`"`*actionText*`"{[`*actionProcesses*`]}`

where

	n	is the alarm number. ***n*** is optional. See "Alarm Number" on page 97.
	test	is the alarm's true/false test. It is the DT800 input to be tested (see "Alarm Input" on page 97), followed by the test condition (see "Alarm Condition" on page 98). You can include an optional delay period in ***test*** (see "Alarm Delay Period" on page 98).
ACTIONS	***digitalAction***	is one or two digital state output channels whose state mimic the alarm state. See "Alarm Digital Action Channels" on page 98. ***digitalAction*** is optional.
ACTIONS	***actionText***	is a text message that is returned to the host and/or logged when alarm tests true. See "Alarm Action Text" on page 99. ***actionText*** is optional.
ACTIONS	***actionProcesses***	is a list of DT800 commands that are executed when alarm tests true. See "Alarm Action Processes" on page 100. ***actionProcesses*** is optional.

These actions can be replaced by a logical operator (AND, OR or XOR) to combine more than one alarm. See "Combining Alarms" on page 101.

Single-shot alarm **RA3S ALARM...**
Alarm actions occur **once** when the alarm becomes true.



Repeating alarm **RA3S ALARMR...**
Alarm actions occur **every 3 seconds** while the alarm is true.

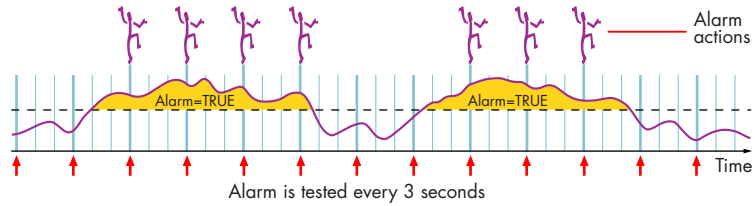


Figure 69 Comparing single-shot and repeating alarms (3-second schedule example)

Example — Typical Alarm Command

The alarm command

`ALARM1(2V>1000)1DSO"HighVoltage"{[5CV=1 HA RB1M]}`

(which you include in any report schedule) contains the following:

`ALARM1`	is the single-shot alarm command (`ALARM`) and an alarm number (`1`).
`(2V>1000)`	is the alarm true/false *`test`* of the input (`2V`) against the condition (`>1000`mV). If, at the moment of testing, the voltage on channel 2 is greater than 1000mV (`2V>1000`), the alarm is true. Otherwise, the alarm is false.
`1DSO`	is the *`digitalAction`*. It instructs the DT800 to set its digital state output channel 1 ON if the alarm is true and OFF if the alarm is false.
`"HighVoltage"`	is *`actionText`*. The DT800 returns and/or logs this message when the alarm becomes true.
`{[5CV=1 HA RB1M]}`	is the *`actionProcesses`*, a list of commands that the DT800 executes when the alarm becomes true.

Alarm Number

Shown as ***n*** in the alarm command (above); optional.

Each of the two alarm types can be given either

- a number in the range 1 to 32767 — that is, `ALARM`***n*** or `ALARMR`***n*** where ***n*** is the alarm number (for example, `ALARM37`), or
- no number — that is, `ALARM` or `ALARMR`

The following table compares the functionality of alarms with and without numbers:

Alarms with Alarm Number Examples: `ALARM10`, `ALARMR99`	**Alarms without Alarm Number** `ALARM` or `ALARMR`
Do not have to be entered in numeric sequence	Order doesn't matter
The alarm number, entry time and exit time are logged in the DT800's general data store.	No alarm information is logged
`?`***n*** (see "Polling Alarm Data" on page 102) returns the current value of the tested channel.	`?`***n*** returns an undefined alarm error.
`?`***x*** and `?ALL` (see "Polling Alarm Data" on page 102) return all current values.	`?`***x*** and `?ALL` have no effect.

Alarm Input

Part of *`test`* in the alarm command (page 96); not optional.

The **alarm input** is the data item or value that is to be tested by the alarm command. Alarm inputs for alarm commands can be any of the following:

- any analog, digital, counter or serial input channel definition (channel number + channel type + channel options)
- any internal channel
- any channel variable
- time and date
- any system timer
- any system variable

Examples — Alarm Input

The alarm command

`ALARM`***n***`(2R>50)`*`actions…`*

tests the `2R` analog input channel.

The alarm command

`ALARM`***n***`(REFT<100)`*`actions…`*

tests `REFT`, the DT800's body temperature.

The alarm command

`ALARM`***n***`(T>10:30:00)`*`actions…`*

tests the DT800's internal time channel (`T` channel type — see "Time" on page 68).

The alarm command

`ALARM`***n***`(2SV>2000)`*`actions…`*

tests `2SV`, the free space in the DT800's internal data store (see "System Variables" on page 69).

Alarm Condition

Part of ***test*** in the alarm command (page 96); not optional.

The alarm input is compared with the **alarm condition**, which is either

- one of the logical operators **<** or **>** followed by a single setpoint, or
- one of the logical operators **<>** or **><** followed by a pair of comma-separated setpoints.

Operator	Number of Setpoints	Operation
<	1	Less than the setpoint
>	1	Greater than or equal to the set-point
<>	2	Less than the first set-point, OR greater than or equal to the second set-point
><	2	Greater than or equal to the first set-point AND less than the second set-point (that is, between the two setpoints)

The setpoints can be floating-point constants or channel variables.

Recommendation When testing digital state inputs for state 0 or 1, it's best to test for state 0 using **(nDS<0.5)** and to test for state 1 using **(nDS>0.5)**. This avoids any hunting around the 0 and 1 values.

Examples — Alarm Condition

The alarm command

ALARM*n*(5TK<100)*actions…*

tests if temperature measured on channel 5 (**5TK**) is less than 100°C (**<100**).

The alarm condition

ALARM*n*(2R>50)*actions…*

tests if resistance on channel 2 (**2R**) is greater than or equal to 50 ohms (**>50**).

The alarm command

ALARM*n*(10CV<>20CV,30CV)*actions…*

tests if the value of 10CV is less than the value of 20CV, or greater than or equal to the value of 30CV — that is, if the value of 10CV is outside the range of 20CV to 30CV.

The alarm command

ALARM*n*(3CV><5.5,8.5)*actions…*

tests if the value of 3CV is greater than or equal to 5.5 and less than 8.5 — that is, if the value of 3CV is between 5.5 and 8.5.

The alarm command

ALARM*n*(T><06:30,18:30)*actions…*

tests if the DT800's time is between 06:30:00 and 18:30:00.

Alarm Delay Period

An addition to ***test*** in the alarm command (page 96); optional.

You can extend the test component of the alarm command by adding a **delay period**. If you do this, the alarm's test must remain true for the delay period before any of the actions can be performed. The format is

/*N*S	Seconds
/*N*M	Minutes
/*N*H	Hours
/*N*D	Days

where ***N*** is an integer in the range 1 to 255.

If the alarm tests false again during the delay period, the delay counter is reset and does not begin counting again until the next true test. The result is a filtering action that ensures input noise does not cause unwanted or rapid output actions.

Example — Delay Period

The alarm command

ALARM*n*(3V>100.0/30S)*actions…*

specifies that the voltage on channel 3 (**3V**) must equal or exceed 100.0mV (**>100.0**) for 30 seconds before the actions are performed (**/30S**).

Alarm Digital Action Channels

Shown as ***digitalAction*** in the alarm command (page 96); optional.

One — or two (comma-separated) — digital state output channels (***n*DSO**) can be declared for each alarm to mimic the state of the alarm. That is, these outputs are set OFF if the alarm tests false, and are set ON of the alarm tests true:

Alarm	DSO	
False	OFF	DSO = 0 (DSO is set high)
True	ON	DSO = 1 (DSO is set low/active)

Typically, you use the digital state outputs to annunciate the DT800 alarm by switching devices such as relays, sirens and lights, or to directly control actuators and similar equipment.

In the **ALARM…** command, you list the single digital state output, or the pair of comma-separated digital state outputs, immediately after the alarm ***test***.

See also "Digital State Outputs" beginning on page 154.

Examples — Digital Action

The command

ALARM*n*(2V>660.0)4DSO

tests the voltage on channel 2 (**2V**) and

- turns digital state output 4 ON when the voltage equals or exceeds 660.0mV
- turns digital state output 4 OFF when the voltage drops below 660.0mV.

The command

ALARM*n*(2V>660.0)7DSO,8DSO

tests the voltage on channel 2 (**2V**) and

- turns the two digital state output channels ON when the voltage equals or exceeds 660.0mV
- turns the two digital state output channels OFF when the voltage drops below 660.0mV.

Alarm Action Text

Shown as ***actionText*** in the alarm command (page 96); optional.

Action text is automatically returned to the host computer and/or logged to alarm memory

- once whenever a single-shot alarm (**ALARM** or **ALARM*n***) tests true, or
- repeatedly at the controlling schedule's rate while a repeating alarm (**ALARMR** or **ALARMR*n***) remains true.

You can include up to 200 characters of action text in each alarm. A total text space of 16384 characters is reserved for all alarms (shared with expressions text).

Note There is no garbage collection in this text space. That is, new text is appended to existing text in the text space, and superseded text is only removed by a system reset.

The action text is listed in quotes **" "** as follows:

ALARM*n*(*test*)"*actionText*"

Setting the alarm message switch to **/z** stops the return of the action text to the host — see the DT800 Switches table (page 112). This is useful when the action text is only required for a display (if fitted).

The format of the action text returned by an alarm differs for the default free-format mode (**/h**) and fixed-format mode (**/H**). See the examples below.

Substitution Characters

You can place special substitution characters into ***actionText***. These instruct the DT800 to dynamically insert the following information when the alarm returns and/or logs its action text:

Characters		Example
!	Substitutes DT800 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	**Deg C**
?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	**#**
?ncv	Substitutes the current value of the specified channel variable, where ***ncv*** is the number of the channel variable (**1** to **500**). For example, **?3** instructs the DT800 to substitute the contents of 3CV into the alarm action text. You can also specify the format and number of decimal places (see "Output Data Format" on page 75). For example: • **?3F** inserts the value of 3CV in fixed-point format • **?3E** inserts the value of 3CV in exponential format • **?3M1** inserts the value of 3CV in mixed format with **1** decimal place	

Some of these characters are especially useful with SMS messaging.

Any of the ASCII control characters (**^A** to **^Z**) can also be included in action text. Some useful control characters are

^G	Bell
^M	Carriage return
^J	Line feed
^b	Quotation mark (**"**)

Example — Action Text in Free-Format Mode (/h)

When the DT800 is in free-format mode (see page 21), the action text in the alarm command

ALARM8(*test*)"Boiler Pressure is ?V?U at @^M^J"

instructs the DT800 to return an alarm record of the form

```
Boiler Pressure is 1.563MPa at 14:32:01.23964
```

on each false-to-true transition of the alarm (includes current value, units text and time that alarm occurred). No action text is issued on the true-to-false transition.

Example — Action Text in Fixed-Format Mode (/H)

When the DT800 is in fixed-format mode (see page 21), the action text in the same alarm command

ALARM8(*test*)"Boiler Pressure is ?V?U at @^M^J"

instructs the DT800 to return an alarm record[9] of the form

```
A,080035,"Boiler_1",2001/04/16,14:32:01.25487,8;B,1,"Boiler
Pressure is 1.563MPa at 14:32:01.23964^M^J";0102;3D95
```

on each false-to-true transition of the alarm (includes current value, units text and time that alarm occurred)

where

A	signifies that this is an Alarm record
080035	is the DT800's serial number
"Boiler_1"	is the name of the job containing the alarm
2000/04/16,14:32:01.25487	is the date and time of the alarm record
8	is the alarm number — see Figure 10
B	is the name of the schedule containing the alarm
1	is the alarm state: 1=false to true, 2=continuing true, 3=true to false — see "Alarm States and Tags" on page 103
"Boiler Pressure is 1.563KPa at 14:32:01.23964^M^J"	is the action text
0102;3D95	are communications error checks (record character count;checksum)

No action text is issued on the true-to-false transition.

[9] See also Figure 10 (page 22).

Alarm Action Processes

Shown as **`actionProcesses`** in the alarm command (page 96); optional.

Action processes can be any DT800 functions that you want 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 DT800. You can 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.

You can include immediate schedules in action processes.

Note Action processes cannot include alarm commands or new report schedules.

You place action processes within the bracket sequence **`{[ ]}`** of an alarm command. Action processes are executed

- once when an **ALARM** or **ALARM*n*** tests true, or
- repeatedly at the controlling schedule's rate while an **ALARMR** or **ALARMR*n*** remains true.

Action processes have many uses. Some common ones are demonstrated in the following examples.

Example — Alarm Action Processes: Switching an Actuator

Probably the most traditional use of action processes is to manage the DT800's digital state output channels, which in turn manage devices connected to them (relays for switching power to lights or sirens to annunciate an alarm condition, to turn pump motors on or off,...).

Every second (**1S**), the schedule

```
RA1S
  ALARM(2I<12.54)"Pump ON"{[3DSO(W)=0]}
```

tests the alarm, whose input is an external-shunt current-type level sensor mounted in a tank and connected to the DT800's analog channel 1 (**2I** — see **I** channel type on page 64). If the tank level falls below the setpoint (**2I<12.54**) the alarm tests TRUE, which causes the DT800 to

- return and/or log the alarm message `Pump ON`
- set its digital state output 3 to **0** (OFF = high/active) to turn a pump on (**{[3DSO(W)=0]}** — see "Digital State Outputs" on page 154).

You could also achieve this using an alarm digital action (page 98) instead of an alarm action process:

```
ALARM(1I<12.54)1DSO
```

Example — Alarm Action Processes: Controlling a System

Alarm action processes can also be used to control a system or process. Although this method uses simple "bang-bang" actuator control, it is often adequate.

The schedule

```
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. It assumes that the thermal inertia of the system is sufficient to prevent hunting about the control point. The two alarms work to hold the temperature at 75°C ± 0.25°C.

Examples — Alarm Action Processes: 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)
  ALARM1(1CV>5.0){[RA2M]}
  ALARM2(1CV<4.5){[RA15M]}
```

measures wind speed

- every 2 minutes if wind speed is greater than 5m/s, or
- every 15 minutes if windspeed is less that 5m/s

as follows:

`1V("Wind speed",S1,=1CV)`	is the instruction to record the wind speed and assign it to channel variable 1 for testing
`ALARM1(1CV>5.0){[RA2M]}`	changes the schedule's trigger to every 2 minutes (**{[RA2M]}**) if windspeed exceeds 5.0m/s (**(1CV>5.0)**)
`ALARM2(1CV<4.5){[RA15M]}`	changes the schedule's trigger back to every 15 minutes (**{[RA15M]}**) if windspeed drops below 4.5m/s (**(1CV<4.5)**)

Note the deliberate 0.5m/s hysteresis to prevent oscillation around the switchover point.

The program

```
RC30M LOGONC HC
  5TK("Oven Temp")
RD1M
  ALARM(5TK>120){[GA]}
  ALARM(5TK<120){[HA]}
```

continuously monitors the temperature of an oven and logs the temperature whenever it exceeds 120°C:

`RC30M LOGONC HC` `  5TK("Oven Temp")`	is a report schedule that logs (**LOGONC**) oven temperature (**5TK**) when active (initially halted by **HC**)
`RD1M`	initiates the alarm tests every minute (**1M**)
`  ALARM(5TK>120){[GC]}`	sets schedule C to Going (**{[GC]}**) if oven temperature exceeds 120°C (**(5TK>120)**)
`  ALARM(5TK<120){[HC]}`	sets schedule C to Halted (**{[HC]}**) if oven temperature drops below 120°C (**(5TK<120)**)

Example — Alarm Action Processes: Using an Alarm to Poll a Schedule

The program

```
BEGIN
  1CV(W)=0
  5CV(W)=1
  6CV(W)=2^5CV
  RA1S
    1CV(W)=1CV+1
    ALARM(1CV>6CV){[XB 5CV(W)=5CV+1 6CV(W)=2^5CV]}
  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 1, 2, 4, 8, 16, 32, 64, 128, 256,... seconds as follows:

`RA1S` `1CV(W)=1CV+1`	is a simple one-second accumulator
`ALARM(1CV>6CV)`	tests if the next log point has been reached
`{[`	indicates start of ***actionProcesses***
`XB`	polls report schedule B
`5CV(W)=5CV+1`	increments the exponent
`6CV(W)=2^5CV`	calculates the next log point
`]}`	indicates end of ***actionProcesses***
`RBX LOGONB` `1..5TK`	is a report schedule that collects data when polled

Combining Alarms

Alarms can be combined together to yield a single result from several alarm tests. They are combined using logical operators, 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.

The logical operators are **AND**, **OR** and **XOR**.

The DT800 evaluates a combined alarm in the order in which its component alarms appear in the schedule command. This means that the alarm numbers do not have to be sequential.

Examples — Combining Alarms

The combined alarm

```
ALARM4(3TK>100)OR
ALARM2(2TK>100)OR
ALARM7(5TK>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 **2TK**, **3TK** or **5TK** exceeds 100°C after 10:00:00 am.

The combined alarm

```
ALARM(T><06:00,18:00)AND
ALARM(1TK>35)"Vents open"{[1DSO(W)=1]}
```

opens the vents in a glasshouse if the air temperature exceeds a threshold during daylight hours. The combined alarm becomes true if the temperature exceeds 35°C between the hours of 06:00 and 18:00, switching ON (low/active) the DT800's digital state output 1 (`{[1DSO(W)=1]}`) to activate the vent mechanism, and issuing a `Vents open` message.

Polling Alarm Data

Alarm data (that is, the current value of alarm inputs) can be polled (requested) by the host computer at any time.

There are three commands for polling alarm data:

?*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**	For numbered and un-numbered alarms
?ALL	returns the current input values of all alarms in all schedules	

When un-numbered alarms are polled by **?*x*** and **?ALL**, the alarm number is returned as A0.

The format of data returned by an alarm poll command differs for the default free-format mode (**/h**) and fixed-format mode (**/H**). See the following examples.

Examples — Polling Alarm Data

When the DT800 is in **free-format mode** (see page 21) and the alarm command

```
ALARM5(2R>50)"Warning^M^J"
```

is included in schedule **F**, the alarm poll command

```
?5
```

instructs the DT800 to return an alarm **data** record of the form

```
A5 2R>50 2R 123.45 Ohm
```

where

A5	is the alarm ID
2R>50	is the alarm ***test***
2R	is the alarm input
123.45	is the current value of the alarm input
Ohm	is the units text for the alarm input

When the DT800 is in **fixed-format mode** (see page 21) and the same alarm command

```
ALARM5(2R>50)"Warning^M^J"
```

is included in schedule **F**, the alarm poll command

```
?5
```

instructs the DT800 to return an **alarm data** record[10] of the form

```
D,080416,2001/04/12,07:20:40,0.08932,2;F,5,"2R>50",123.45;0069;2614
```

where

D	signifies that this is a Data record
080416	is the DT800's serial number
2001/04/12,07:20:40,0.08932	is the date and time of the alarm record
2	signifies "not used" (see Figure 10)
F	is the name of the schedule containing the alarm command
5	is the alarm number
2R>50	is the alarm test
123.45	is the current value of the alarm input
0069;2614	are communications error checks (record character count;checksum)

[10] See also Figure 10 (page 22).

Logging and Retrieving Alarms

The DT800 stores **alarm state records** for later retrieval and analysis. These records comprise the alarm state, a timestamp, the alarm number and any action text. They can be logged into the DT800's general data store whenever an alarm is tested.

See "Logging Alarm States" below and "Retrieving Logged Alarm States" beginning on page 105.

Logging Alarm States

Alarm States and Tags

The DT800 recognizes these four alarm states/transitions:

- continuing false
- false-to-true
- continuing true
- true-to-false

When the DT800 tests an alarm and logs the alarm record, it includes the state/transition of the alarm in the form of a numeric tag:

State/Transition	Tag	Comment
Continuing false	**0**	Not used
False-to-true	**1**	Universal
Continuing true	**2**	Only used by **ALARMR*n*** (repeating numbered alarms)
True-to-false	**3**	Universal

For examples of alarm records containing these tags, see the What's Logged column in the table on page 104.

Which States are Logged?

You can choose which alarm states are logged. This is controlled by P9:

P9=	State/Transition Logged
0	None
1	False-to-true only — the DT800's default
2	True-to-false only
3	Both

The default setting is **P9=1**, which causes the DT800 to only log the false-to-true transitions.

For a repeating alarm, the alarm state is also logged each time the alarm is tested and is true (that is, at the alarm's report schedule interval).

Numbered Alarms Only

Alarm states can only be logged for numbered alarms. The alarm number identifies these alarms in retrieved alarm data. See "Alarm Number" on page 97.

Where are Alarm States Stored?

Alarm states are logged in the DT800's general data storage memory along with data from other sources such as input channels, calculations, and processes. Although data from these sources each has a separate set of log files, the general data logging commands apply to all.

Enabling Logging of Alarm States

The actual logging of alarm state is enabled by the general **LOGON** command, or by the **LOGON*x*** command for the report schedules to which the alarms belong, where ***x*** = **A**, **B**,…**K**. Alarm states are logged by the schedule to which they belong, in the same way that data is logged by the schedule to which it belongs. See "LOGON and LOGOFF Commands" on page 76.

Overwrite Mode

Normally alarm state is logged until the DT800's data storage memory is filled, after which further logging ceases (although the alarms continue to be tested and other actions performed). However, as with the logging of data (see "What Happens When Memory is Full?" on page 77), the logging of alarm states can also be done in overwrite mode, in which the newest data progressively overwrites (replaces) the oldest data after the data memory is filled. This is enabled by the **/O** switch.

Action Text is Included

If alarms that are being logged have action text, the action text is logged along with the alarm state (in the DT800's general data store). And if the action text includes substitution characters, the substitute text is included in the logged action text. See "Alarm Action Text" on page 99.

Example — Logging Alarm State and Action Text

The program

```
BEGIN
  P9=3 'Log both transitions
  RC1S
    ALARM5(3TK>50)"Warning ?v?u"
  LOGONC
END
```

instructs the DT800 to log an alarm record when the alarm occurs, and another alarm record when the alarm recovers (because P9 is set for both transitions). The action text **"Warning ?v?u"** is logged for the false-to-true transition, and **"ALARM*n* FALSE"** is logged for the true-to-false transition. The two records have the form

```
A,80416,2001/04/12,07:20:40,0.489312,5;C,1,"Warning 53 degC";
0066;15C4
```

and

```
A,80416,2001/04/12,07:23:53,0.216923,5;C,3,"ALARM5 FALSE";
0067;B5DC
```

Logging Alarm States — What's Logged, What's Returned

For the four types of alarm commands, the following table summarises the alarm information that is included in the logged alarm state records and returned in realtime to the host computer:

Alarm Command	State/Transition	Parameter 9	What's Logged	What's Returned
ALARM(*test*)"*actionText*"	Continuing low	—	—	—
	False to true	—	—	*actionText*
	Continuing high	—	—	—
	True to false	—	—	—
ALARM*n*(*test*)"*actionText*"	Continuing low	—	—	—
	False to true	P9=1 or 3	Timestamp,*n*,1,"*actionText*"	*actionText*
	Continuing high	—	—	—
	True to false	P9=2 or 3	Timestamp,*n*,3,"ALARM*n* FALSE"	—
ALARMR(*test*)"*actionText*"	Continuing low	—	—	—
	False to true	—	—	*actionText*
	Continuing high	—	—	*actionText*
	True to false	—	—	—
ALARMR*n*(*test*)"*actionText*"	Continuing low	—	—	—
	False to true	P9=1 or 3	Timestamp,*n*,1,"*actionText*"	*actionText*
	Continuing high	—	Timestamp,*n*,2,"*actionText*"	*actionText*
	True to false	P9=2 or 3	Timestamp,*n*,3,"*actionText* FALSE"	—

See "Example — Action Text in Free-Format Mode (/h)" and "Example — Action Text in Fixed-Format Mode (/H)" on page 99 for a description of the alarm action text returned in real time.

See Figure 10 (page 22) for a description of alarm records logged and retrieved from memory (always in fixed-format mode).

n is the alarm number.

Retrieving Logged Alarm States

You retrieve (unload) logged alarm state records from the DT800 to the host computer using alarm unload commands:

- **A** commands (no brackets) — unload all alarm records (from the beginning to the end of the alarm store).
- **A( )** commands (round brackets / parentheses) — unload alarm records for a user-defined period in the format specified by the DT800's current P31 and P39 settings (date and time formats).
- **A[ ]** commands (square brackets) — unload alarm records for a user-defined period in the DT800's fixed-format style **YYYY/MM/DD,hh:mm:ss,0.ssssss**, which overrides the DT800's current P31 and P39 settings.

Note Logged alarm state records are not cleared/deleted from the DT800 by unload operations. You can unload the same logged records as many times as you want (until you purposefully delete it, of course — see "Deleting Logged Alarm Records" on page 106).

The A Unload Commands

The **A** unload commands return all alarm state records (that is, from the beginning to the end of the alarm store) for user-defined jobs and/or report schedules.

A	Returns the current job's alarms in the order of report schedule A to K
A*x*	Returns the current job's alarms for report schedule ***x***
A"*JobName*"	Returns alarms for ***JobName*** in the order of report schedule A to K
A"*JobName*"*x*	Returns alarms for ***JobName*** report schedule ***x***

Table: Alarm Unload Commands — A

These commands are also listed in "Summary — Retrieval Commands" beginning on page 186.

Examples — A Unload Command

The command

A

instructs the DT800 to unload all of the current job's alarm records from beginning to end for all report schedules A to K.

The command

A"Boiler"G

instructs the DT800 to unload alarm information for job **Boiler**, report schedule **G**.

The A( ) Unload Commands

The **A( )** unload commands return subsets of the alarm state records logged in the DT800. You define the subset to be returned by specifying combinations of

- jobs and/or report schedules, and
- periods designated by a beginning and end ***date*** and ***time***, which you must specify in the DT800's current date and time formats (as set by P31 and P39).

A	Returns the current job's alarms in the order of report schedule A to K
A(*from*)	Returns the current job's alarms • starting from **BEGIN**, ***time*** or ***time,date***
A(*from*)(*to*)	Returns the current job's alarms • starting from **BEGIN**, ***time*** or ***time,date*** and • ending with **END**, ***time*** or ***time,date***
A*x*	Returns the current job's alarms for report schedule ***x***
A*x*(*from*)	Returns the current job's alarms for schedule ***x*** • starting from **BEGIN**, ***time*** or ***time,date***
A*x*(*from*)(*to*)	Returns the current job's alarms for schedule ***x*** • starting from **BEGIN**, ***time*** or ***time,date*** and • ending with **END**, ***time*** or ***time,date***
A"*JobName*"	Returns alarms for ***JobName*** in the order of report schedule A to K
A"*JobName*"(*from*)	Returns alarms for ***JobName*** • starting from **BEGIN**, ***time*** or ***time,date***
A"*JobName*"(*from*)(*to*)	Returns alarms for ***JobName*** • starting from **BEGIN**, ***time*** or ***time,date*** and • ending with **END**, ***time*** or ***time,date***
A"*JobName*"*x*	Returns alarms for ***JobName*** report schedule ***x***
A"*JobName*"*x*(*from*)	Returns alarms for ***JobName*** report schedule ***x*** • starting from **BEGIN**, ***time*** or ***time,date***
A"*JobName*"*x*(*from*)(*to*)	Returns alarms for ***JobName*** report schedule ***x*** • starting from **BEGIN**, ***time*** or ***time,date*** and • ending with **END**, ***time*** or ***time,date***

Table: Alarm Unload Commands — A( )

where

from	can be • **BEGIN** — start from first alarm logged • ***time*** — start from first alarm logged at or after this time today • ***time,date*** — start from first alarm logged at or after this time and date	Both the ***date*** and ***time*** must be specified in the currently-defined format for date (P31) and time (P39). **Note** **BEGIN** and **END** used here are not the same as the **BEGIN** and **END** keywords used to indicate the start and end of a DT800 job.
to	can be • **END** — end with last alarm logged • ***time*** — end with last alarm logged prior to this time today • ***time,date*** — end with last alarm logged prior to this time and date	

These commands are also listed in "Summary — Retrieval Commands" beginning on page 186.

Examples — A( ) Unload Command

■ The command

```
A(12:00,19/1/2000)(12:05,20/1/2000)
```

instructs the DT800 to unload the current job's alarm information for all report schedules in the order A to K, and starting at **12:00** on **19/1/2000** and ending at **12:05** on **20/1/2000**.

■ The command

```
A"Job1"B(BEGIN)(11:15)
```

instructs the DT800 to unload the alarms information for report schedule **B** of **Job1**, from **BEGIN** (the first alarm logged) until **11:15** today.

The A[ ] Unload Commands

The **A[ ]** unload commands return subsets of the alarm state records logged in the DT800. You define the subset to be returned by specifying combinations of

- jobs and/or report schedules, and
- periods designated by a beginning and end ***date*** and ***time***, which you must specify in the DT800's fixed-format style **YYYY/MM/DD,hh:mm:ss,0.ssssss** (ISO date format), which overrides the DT800's current P31 and P39 settings.

A	Returns the current job's alarms in the order of report schedule A to K
A[*from*]	Returns the current job's alarms • starting from ***time*** or ***time,date***
A[*from*][*to*]	Returns the current job's alarms • starting from ***time*** or ***time,date*** and • ending with ***time*** or ***time,date***
A*x*	Returns the current job's alarms for report schedule ***x***
A*x*[*from*]	Returns the current job's alarms for schedule ***x*** • starting from ***date*** or ***date,time***
A*x*[*from*][*to*]	Returns the current job's alarms for schedule ***x*** • starting from ***date*** or ***date,time*** and • ending with ***date*** or ***date,time***
A"*JobName*"	Returns alarms for ***JobName*** in the order of report schedule A to K
A"*JobName*"[*from*]	Returns alarms for ***JobName*** • starting from ***date*** or ***date,time***
A"*JobName*"[*from*][*to*]	Returns alarms for ***JobName*** • starting from ***date*** or ***date,time*** and • ending with ***date*** or ***date,time***
A"*JobName*"*x*	Returns alarms for ***JobName*** report schedule ***x***
A"*JobName*"*x*[*from*]	Returns alarms for ***JobName*** report schedule ***x*** • starting from ***date*** or ***date,time***
A"*JobName*"*x*[*from*][*to*]	Returns alarms for ***JobName*** report schedule ***x*** • starting from ***date*** or ***date,time*** and • ending with ***date*** or ***date,time***

Table: Alarm Unload Commands — A[ ]

where

from	can be • ***date*** — start from first alarm logged at or after this date • ***date,time*** — start from first alarm logged at or after this date and time	**Note** Type ***time*** and ***date*** in fixed-format ISO-style (see examples below).
to	can be • ***date*** — end with last alarm logged on this date • ***date,time*** — end with last alarm logged on this date prior to this time	

These commands are also listed in "Summary — Retrieval Commands" beginning on page 186.

Examples — A[ ] Unload Command

■ Here's the **A[*from*]** command showing valid forms of the fixed-format style date and time:

```
A[2000/07/26,12:30:00,0.250366]
A[2000/07/26,12:30:00]
A[2000/07/26]
```

■ Here's the **A[*from*][*to*]** command showing valid forms of the fixed-format style date and time:

```
A[2000/07/26,12:30:00,0.250366][2000/07/28,18:30:00,0.750244]
A[2000/07/26,12:30:00][2000/07/28,18:30:00,0.750244]
A[2000/07/26][2000/07/28,18:30:00,0.750244]
```

```
A[2000/07/26,12:30:00,0.250366][2000/07/28,18:30:00]
A[2000/07/26,12:30:00][2000/07/28,18:30:00]
A[2000/07/26][2000/07/28,18:30:00]
```

```
A[2000/07/26,12:30:00,0.250366][2000/07/28]
A[2000/07/26,12:30:00][2000/07/28]
A[2000/07/26][2000/07/28]
```

Deleting Logged Alarm Records

Delete logged alarm records from the data memory of the DT800 using these commands:

DELALARMS	Deletes the current job's alarms	Job names, directory structures, programs and data are not erased.
DELALARMS"*JobName*"	Deletes only ***JobName***'s alarms	
DELALARMS*	Deletes all jobs' alarms	

These commands are also listed in "Summary — Delete Commands" on page 185.

Part H — Housekeeping
Configuring the DT800

Parameters

Internal settings

DT800 **parameters** are internal system settings. They are global in their effect, and let you set a variety of options. As a general rule, set the parameters that require changing before you program schedules and alarms.

You can

- set or read parameters from the host computer, from a memory card program or from alarm actions
- pre-set parameters using a **PROFILE...** command — see the PARAMETERS row in the DT800 PROFILE Details table (page 113).

Reading Parameters

To read the current setting of a parameter, simply send the parameter's ID. For example, send

```
P22
```

to read the current setting of parameter 22.

Setting Parameters

Parameters can be set at any time, and new settings generally take effect immediately. For example, send

```
P22=44
```

to set parameter 22 to the value **44**.

Parameters are not the same as channels or variables. If you include a parameter in a schedule, it does not become part of the schedule. Instead, it is processed immediately.

In fixed-format mode (**/H**, see page 111), three parameters are forced: **P22=44**, **P24=13** and **P38=46**. The previous values for these are restored on leaving fixed-format mode (**/h**).

You can also use the **DO** command for executing parameter commands from within schedules. See "Unconditional Processing — DO... Command" on page 61.

Parameter	Specifies	Units	Default Value	Range of Values	Comment
P3	Minimum sleep period	Seconds	4	1 to 30000	Sleep only if sleep duration can be for at least this period of time (DT800 assesses when next schedule is due to expire)
P4	Sleep-to-wake settling latency	Seconds	3	1 to 30000	Time required by DT800 to resume normal operation after leaving sleep mode
P5	Maximum sleep period	Minutes	1	1 to 30000	
P6	Inclusion of job name in fixed-format mode messages	Mode	1	0 to 1	Send **P6=0** for **omit**, **P6=1** for **include**
P9	Logging of alarm state	Mode	1	0 to 3	**P9= / Transition Logged**: 0 None; 1 False-to-true only (default); 2 True-to-false only; 3 Both
P10	Logging of **TEST** results to event log	Mode	0	0 to 1	Send **P10=0** for **disable**, **P10=1** for **enable**
P11	Mains frequency (Set P11 to local mains frequency for best noise rejection.)	Hz	50	25 to 30000	Sets ADC sample duration to 1/Hz seconds. Kept during hardware and software resets.
P12	Minimum measurement period for frequency measurement	ms	20	1 to 30000	See "Frequency" on page 142.

Table: DT800 Parameters (sheet 1 of 4)

Parameter	Specifies	Units	Default Value	Range of Values	Comment
P14	Comms ports password protection timeout	Seconds	600	1 to 30000	When a password is defined, the DT800 automatically signs off after this period of inactivity (see "Password Protection — Comms Ports" on page 124).
P15	Low-power operation	Mode	0	0 to 2	**P15= Mode** 0 Auto 1 Force sleep 2 Force normal operation See "Controlling Sleep" on page 43).
P17	Delay to low-power mode	Seconds	30	1 to 255	Sets how long the DT800 waits to enter low-power mode after communication or peripheral device activity and so on (see "Controlling Sleep" on page 43)
P22	Data delimiter character	ASCII	32 (space)	1 to 255	ASCII character (as decimal number) between data points in free-format mode (see page 21). Forced to 44 (comma) when in fixed-format mode. If using successive immediate schedules, see "Cautions for Using Immediate Schedules" on page 49.
P24	Scan delimiter character	ASCII	13 (**CR**, which the DT800 automatically follows with **LF**)	1 to 255	ASCII character (as decimal number) between groups of data points in a scan in free-format mode (see page 21). **Note** When P24=13 (the default): the DT800 always automatically follows this carriage return (ASCII 13) with a line feed character (ASCII 10). See "Format of Returned Data" on page 21.
P26	XOFF timeout before XON	Seconds	30	0 to 255	Timeout before incoming XOFF state is automatically switched to XON state. **P26=0** disables timeout.
P31	Date format	Mode	1	0 to 5	**P31= Date Format** 0 day number 1 DD/MM/YY European 2 MM/DD/YYYY North American 3 YYYY/MM/DD ISO 4 D.D decimal days since base date 5 s.s decimal seconds since base date See "Date" on page 68 for country default. Kept during software reset.
P32	Number of significant digits	# digits	7	1 to 9	Sets significant digits of output data. Note: logged data is always stored to 5 digits, so P32>5 is only useful for real-time data.
P33	Field width	# characters	0 (variable)	0 to 80	If P33>0 this defines fixed field width for all output data (right justified, space padded or least significant digits truncated). See "Numeric Format" on page 22.
P36	Temperature units	Mode	0	0 to 3	**P36= Units** 0 °C 1 °F 2 K 3 °R Data is converted before being placed into store and cannot be converted at unload time.
P38	Decimal point character	ASCII	46 (.)	0 to 255	The character used as a decimal point in floating-point numbers (see "Output Format" on page 10)

Table: DT800 Parameters (sheet 2 of 4)

Parameter	Specifies	Units	Default Value	Range of Values	Comment		
P39	Time format	Mode	0	0 to 5	**P39= Time Format** 0 hours:minutes:seconds.seconds 1 s.s (decimal seconds) since midnight 2 m.m (decimal minutes) since base 3 h.h (decimal hours) since base 4 D.D (decimal days) since base 5 s.s (decimal seconds) since base	Kept during reset if valid	See "Time" on page 68.
P40	Time separator character	ASCII	58 (:)	1 to 255	ASCII character (as decimal number) separator character for hh:mm:ss time format		
P41	Time sub-second digits	Digits	3	0 to 6	Sets number of digits in time outputs		
P45	DDE/OLE tag control	Mode	0	0 to 2	**P45= Mode** 0 off 1 OLE (**$!**) 2 DDE (**&!**)		Whitespace is replaced with underscore
P46	Number of background measurements of each channel per integral number of mains cycles (normal mode and fast mode)	Background measurement	80	1 to 200 (50Hz), 1 to 167 (60Hz)	Sets the number of background measurements that the DT800 takes to average for a fundamental sample in normal mode (default = 80) and fast mode (default = 2). See "Normal Mode Sampling" (page 51) and "Fast Mode Sampling" (page 52).		
P47	Sensor power supply voltage	Volts	0	0, 5 or 10	Sets voltage at **Sp** (sensor power out terminals) relative to **Sr** — off, 5V or 10V. See Figure 92 (page 148).		
P48	Burst sampling speed	kHz	100	1 to 100	Sets the maximum ADC burst speed. See "Burst Mode Sampling" on page 53.		
P49	Common-mode range select	Mode	0	0 or 1	Send **P49=0** for **–10V to +10V**, **P49=1** for **0V to +20V**. Note that this parameter does not affect the DT800's maximum analog input voltage (still ±13V).		
P50	Time instant format (format of instants in time)	Mode	0	0 to 5	**P50= Time Format** 0 P39P22P31 (time, delimiter, date) 1 P31P22P39 (date, delimiter, time) 2 s.s (decimal seconds) from base 3 m.m (decimal minutes) from base 4 h.h (decimal hours) from base 5 D.D (decimal days) from base	Kept during reset if valid	See **TOR** and **TOF** in the Digital Manipulation category in the DT800 Channel Options table (page 74).
P51	Time interval format	Mode	0	0 to 5	**P51= Time Format** 0 P39P22d (time, delimiter, day) 1 dP22P39 (day, delimiter, time) 2 s.s (decimal seconds) elapsed 3 m.m (decimal minutes) elapsed 4 h.h (decimal hours) elapsed 5 D.D (decimal days) elapsed	Kept during reset if valid	See **TRR**, **TRF**, **TFR** and **TFF** in the Digital Manipulation category in the DT800 Channel Options table (page 74).

Table: DT800 Parameters (sheet 3 of 4)

Parameter	Specifies	Units	Default Value	Range of Values	Comment
P53	Serial sensor timeout	Seconds	10	0 to 60	Send **P53=0** for **no timeout**
P54	Burst timeout	Seconds	10	0 to 30000	Send **P54=0** for **no timeout**
P55	Enable schedule wakeup	Bits	16383	0 to 16383	Bitmap entered as decimal value **S K J I H G F E D C B A X *** (**S** = bit 13, **X** = bit 1, ***** = 0)
P56	Serial Channel debugging — see "Serial Channel Debugging Tools" on page 165				
P57	Number of frame capture cycles	Count	5	1 to 100	Attempts left to capture complete frame set
P58	Excitation tolerance	%	10	0 to 100	Sets tolerance of excitation adjustment relative to specified value. Send **P58=100** for **don't adjust excitation**.
P59	Maximum gain for automatic gain-ranging	Mode	10	0 to 10	See "Autoranging" on page 13.
P60	Maximum sampling frequency	kHz	50	1 to 100	ADC speed
P61	Internal measurement check interval	Sec	3	0 to 30000	Send **P61=0** to **disable**
P63	Gain-set to use		1	0 to 1	Send **P63=0** for **default gain-set**, **P63=1** for **characterized gain-set**

P59=	Upper Autorange Limit
0	20V (Note: maximum analog input voltage must not exceed ±13V)
1	10V
2	5V
3	2V
4	1V
5	500mV
6	200mV
7	100mV
8	50mV
9	20mV
10	10mV

Table: DT800 Parameters (sheet 4 of 4)

Switches

UPPERCASE = ON, lowercase = off

DT800 **switches** are analogous to electrical switches, and are turned on by uppercase and off by lowercase. Like parameters, switches are internal system settings and generally global in effect; unlike parameters, switches can only have two values — on or off. Switch commands can be issued at any time, and most take effect immediately. Delay in effect may occur if data is buffered in the DT800 or in the host computer.

You can use a **PROFILE…** command to pre-set switches — see the SWITCHES row in the DT800 PROFILE Details table (page 113).

You can also use the **DO** command for executing switch commands from within schedules. See "Unconditional Processing — DO… Command" on page 61.

Viewing Switch Settings

The STATUS9 command returns the current switch settings to the host. For example

```
/a/B/C/d/E/f/g/h/i/J/k/l/M/N/o/r/S/t/U/v/w/x/y/Z
```

Switch Enabled	Switch Disabled	Function	Default	Comment
/A	/a	Display alarms	/a	Enables the display of displayable alarms if your *dataTaker* is fitted with a display
/B	/b	Report burst data immediately	/B	Causes the *dataTaker* to unload burst data to the host computer immediately after storing it
/C	/c	Include Channel type	/C	Channel type is included with channel number in returned data — for example 5PT392 instead of 5. See "Format of Returned Data" on page 21.
/D	/d	Add date to returned data	/d	Equivalent to a **D** at beginning of a schedule's channel list
/E	/e	Echo	/E	Enables echo of commands to host (if not unloading data and not in fixed-format mode). Useful in terminal mode communications with the DT800.
/F	/f	Fix (lock) schedules	/f	Prevents a DT800's scan schedules (trigger or channel list) being modified. Note that a restart still erases the schedules.
/G	/g	Disables DT800 startup program	/G	See "Startup Job" on page 115
/H	/h	Fixed-format (Host) mode	/h	Fixed-format mode of data returned from DT800 — see "Fixed-Format Mode /H" on page 21
/I	/i	Schedule ID	/i	Returns schedule identifier
/J	/j		/J	Reserved for future use
/K	/k	Enable default internal housekeeping measurements	/K	Allows DT800 to measure battery, charger and other internal quantities — checks these every P61 seconds (see page 110) if awake
/L	/l	*dataTaker* serial number prefix	/l	Prefixes the DT800's serial number to a schedule's returned data — for example `dataTaker DT800 080015 5PT385 232.5` indicating the data is from `dataTaker DT800` number `080015`.
/M	/m	Messages	/M	Enables error and warning messages to be returned to host (see "Error Messages" on page 197)
/N	/n	Channel numbers	/N	Includes channel number (and type if **/C** switch is on) with returned data. See "Format of Returned Data" on page 21.
/O	/o	Overwrite memory	/o	Oldest data is over-written (**/O**), otherwise logging stops when memory is full. See "Overwrite Mode (/O)" on page 77.
/P	/p			Reserved for future use
/Q	/q			Reserved for future use
/R	/r	Return data	/R	Allows real-time data to be returned to the host computer. **/r** can reduce power consumption.
/S	/s	Synchronize to midnight	/S	Synchronizes all schedules' time intervals to midnight — for example, **RA1M** scans on the minute. Otherwise schedules run from entry time. See ""Time Triggers — Synchronizing to Midnight" on page 58".
/T	/t	Add time to returned data	/t	Equivalent to a **T** at beginning of a schedule's channel list
/U	/u	Units text	/U	Appends measurement units to returned data (see "Format of Returned Data" on page 21) and makes error messages verbose (see "Error Messages" on page 197)

Table: DT800 Switches (sheet 1 of 2)

Switch Enabled	Switch Disabled	Function	Default	Comment
/V	/v	Event log dump	/v	Enables automatic event log dump upon internal error reset
/W	/w	Intermediate (working) channels	/w	Allows working channels (see W channel option on page 75) to be reported but not logged. See also "Calculations (Expressions)" on page 94.
/X	/x	Progressive maxima and minima	/x	If display is fitted, allows the display of progressive maximum and minimum values for statistical channels
/Y	/y	Priority to return data	/y	If real-time data has not been returned before the next scan becomes due, data return is given priority (the scan may be omitted).
/Z	/z	Stops alarm messages	/Z	Enables alarms to issue action text to host computer or printer. See "Alarm Action Text" on page 99.
//	–	Default switches	–	Sets all switches to their default state

Table: DT800 Switches (sheet 2 of 2)

User Startup Defaults

The DT800 can start up after a firm reset (a **SINGLEPUSH** command, a hardware reset or a power-up reset) pre-configured with your preferred settings and running your preferred job:

- To make the DT800 start with your **preferred configuration settings**, see "User Startup Profile" below. You use this method to set startup defaults for time format, date format, mains frequency and other characteristics listed in the DT800 PROFILE Details table below.
- To make the DT800 start running your **preferred job**, see "Startup Job" on page 115.

You can also protect your defaults against possible corruption — see "Protecting Startup Files" on page 116.

User Startup Profile

The DT800 can store your preferred settings and automatically apply them after it is restarted by a firm reset (see page 118). In this way, the DT800 is automatically returned to your chosen configuration every time it (or, more accurately, its relevant sub-system) is re-started. This saves you from having to repeatedly specify the same settings in every program you send to the DT800.

Configuration settings for DT800 sub-systems such as the Host RS-232 port, host modem, Ethernet interface, FTP server, jobs, switches and parameters can all be restored in this way. In addition, you can use this facility to store your own information in the DT800.

USER.INI (User Initialization File)

Your default settings are stored in the DT800's user initialization file USER.INI, which is automatically read and applied upon DT800 startup and sub-system initialization. If USER.INI contains no user default for a particular setting, the DT800 uses its factory default (listed in the table below).

You use **PROFILE...** commands to build the USER.INI file. These commands are described on page 115 and use information from the table below. The table lists the pre-defined DT800 sub-systems — called **sections**— for which you can specify startup defaults.

You can also use USER.INI as a repository for additional information — see "Adding Your Own Comments" on page 115.

USER.INI is created the first time you send a **PROFILE...** command to the DT800. It resides in the DT800's battery-backed SRAM memory and is maintained through all types of reset. It is only lost/deleted if both the internal main battery and the internal memory-backup battery are removed from the DT800, or if you send the **FORMAT"B:"** command (see the DT800 Resets table on page 118). In addition, the DT800 automatically stores a backup copy of USER.INI in the DT800's Flash memory — see "Protecting Startup Files" on page 116.

When Do User Defaults Take Effect?

After you send a **PROFILE...** command to the DT800 to set values in the USER.INI file, the DT800's behaviour does not change until the next time the sub-system using that value is re-started. For example:

- If your USER.INI file contains special switch and parameter settings, these are applied (and re-applied) every time you carry out a firm reset[11]. They are not applied until the next reset after you send the **PROFILE...** command that specifies them.
- **Ethernet Settings** You firstly define a DT800's Ethernet settings using **PROFILE...** commands. Then, for the Ethernet settings to take effect, you must carry out a firm reset[11].

Persistent Settings

Settings specified using USER.INI can be considered "persistent" because they are reapplied at every restart. You can send individual commands to the DT800 (a parameter command such as **P32=2**, or a baud rate command such as **PH=1200**, for example) that immediately override one or more of the DT800's current/permanent settings, but these are only "temporary" in that they will be automatically replaced by the persistent **PROFILE...** settings the next time the related sub-system is restarted.

[11] A singlepush reset —see page 118.

Section	Initialization Key (Setting Name)	Legal Values	Factory Default	Comment
PARAMETERS	P*n*	*number*	See the Default Value column in the DT800 Parameters table on page 107.	Any parameter can be pre-set; for example, **PROFILE"PARAMETERS","P11"="60"** sets the DT800 for a 60Hz mains frequency environment, **PROFILE"PARAMETERS","P31"="2"** sets the DT800 to North American date format.
SWITCHES	A, B, C,...Z	OFF, ON	Varies with switch — see the Default column in the DT800 Switches table (page 111)	Any switch can be pre-set; for example, **PROFILE"SWITCHES","A"="ON"** Switches are checked and set before parameters on every soft or firm reset.
PPP	IP_ADDRESS	*nnn.nnn.nnn.nnn*	1.2.3.4	
	SUBNET_MASK	*nnn.nnn.nnn.nnn*	255.255.255.255	
	REMOTE_IP_ADDRESS	*nnn.nnn.nnn.nnn*	1.2.3.1	
	IP_ADDRESS_NEGOTIABLE	YES, NO	YES	
	REMOTE_IP_ADDRESS_NEGOTIABLE	YES, NO	YES	
	USER	*string*	ANONYMOUS	
	PASSWORD	*string*	PASSWORD	

Table: DT800 PROFILE Details (sheet 1 of 2)

Section	Initialization Key (Setting Name)	Legal Values	Factory Default	Comment
HOST_PORT (for DT800's Host RS-232 port)	**BPS**	**50**, **75**, **110**, **150**, **300**, **600**, **1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600**, **115200**	57600	Baud rate to use on DT800's Host RS-232 port
	DATA_BITS	**7**, **8**	8	Number of data bits per character
	STOP_BITS	**1**, **2**	1	Number of stop bits per character
	PARITY	**NO**, **EVEN**, **ODD**	NO	Type of parity to include with each character
	FLOW	**HARDWARE**, **SOFTWARE**, **BOTH**, **NONE**	SOFTWARE	Type of flow control to use on DT800's Host RS-232 port
HOST_MODEM (for modem connected to DT800's Host RS-232 port) See "DT800 Modem (Remote) RS-232 Connection" beginning on page 129.	**DIAL**	***string***	ATD	Issued as prefix to number specified in the **SETDIALOUTNUMBER** command
	INIT	***string***	ATE0Q1&D2S0=4&C1&S0	Automatically issued by the DT800 to initialise a connected modem (see "Modem Initialization Settings" on page 130). To disable automatic initialization, set to empty string.
	EXT_POWER_SWITCH	**0**, **-1**, ***n***	0	**PROFILE"HOST_MODEM","EXT_POWER_SWITCH"="-1"** instructs the DT800 to use its Serial Channel **12V** and **Gd** terminals to supply and control power for the modem — see page 132. **PROFILE"HOST_MODEM","EXT_POWER_SWITCH"="*n*"** instructs the DT800 to use its digital output channel ***n*** (***n*** = **1** to **8**) to control an external power supply to the modem — see page 132. **PROFILE"HOST_MODEM","EXT_POWER_SWITCH"="0"** disables this function.
	MAX_CD_IDLE	***number***	43200 seconds (12 hours)	***number*** must be ≥0. The number of seconds to wait while CD 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 800 Version 4.01` is sent whenever CD changes from inactive to active.
	COMMAND_PROCESSING_TIME	**1** to **32767**	1 second	The number of seconds the DT800 waits after sending a command to the modem (to give the modem time to respond).
ETHERNET See "DT800 Ethernet Communications" beginning on page 134.	**UDP_SUPPORTED**	**YES**, **NO**	NO	
	IP_ADDRESS	***nnn.nnn.nnn.nnn***	0.0.0.0	
	SUBNET_MASK	***nnn.nnn.nnn.nnn***	255.255.255.0	
	GATEWAY	***nnn.nnn.nnn.nnn***	0.0.0.0	
FTP_SERVER	**SUPPORTED**	**YES**, **NO**	YES	Starts the DT800's FTP server functionality (using **NO** frees a small amount of DT800 memory)
	USER	***string***	ANONYMOUS	FTP username; if you change it, anonymous user is not accepted
	PASSWORD	***string***	PASSWORD	FTP password; when user is anonymous, each password is approved
Create your own information sections, keys and values — see "Adding Your Own Comments" below.				

Table: DT800 PROFILE Details (sheet 2 of 2)

Adding Your Own Comments

You can place your own information in USER.INI. By doing this, the information is always kept with the particular DT800 and can be recalled at any time. You do this by creating your own sections with their own keys and values. For example, if you send

```
PROFILE"SiteInfo","BatteryDetails"="New main battery installed 10/2/01"
```

the battery installation date is kept in the DT800 ready for you to read at any future time using the **PROFILE"*section*"** command or the **TYPE"B…"** command (see the table below). The ***section*** and ***key*** here simply serve as ways of categorizing and sub-categorizing your comments.

Deleting USER.INI

Delete the working copy of USER.INI from the DT800's SRAM memory by sending

```
DELUSERINI
```

A new USER.INI will be created the next time you send a **PROFILE…** command. See also "Deleting the Backup Files from Flash" (page 117) to remove the copy of USER.INI that was automatically created by the DT800.

PROFILE… Commands

The following **PROFILE**… commands are available for reading and modifying the contents of the USER.INI file:

PROFILE"*section*","*key*"="*string*"	Sets the value of ***section***'s ***key*** in USER.INI to ***string***	
PROFILE"*section*","*key*"	Returns the current value of ***section***'s ***key*** in USER.INI	
TYPE"B:\\INI\\USER.INI"	Displays the entire USER.INI file	
PROFILE"*section*","*key*"=	Removes ***section***'s ***key***	
PROFILE"*section*","*key*"=""	Sets the value of ***section***'s ***key*** to the empty string	
PROFILE	Returns the currently-active profile settings	The currently-active settings may be different from the settings in USER.INI.
PROFILE"*section*"	Returns ***section***'s currently-active settings	

where

section	is the DT800 sub-system	See the DT800 PROFILE Details table beginning on page 113.
key	is the sub-category of the ***section***	
string	is the value to which the ***key*** is to be set	

No Checking

Note that ***section***, ***key*** and ***string*** contain free text that is not validated in any way. So, if you misspell an entry or use an illegal value, you are not warned. The DT800 simply ignores that line in the USER.INI file and instead starts up using the factory default for that setting. For example, if you accidentally send a **PROFILE…** command to set the baud rate to **9a00** or **9601** (instead of **9600**), the DT800 ignores the value and uses its factory default instead.

Examples — PROFILE… Commands

Set Ethernet user defaults:

```
PROFILE"ETHERNET","IP_ADDRESS"="192.168.1.168"
PROFILE"ETHERNET","UDP_SUPPORTED"="YES"
```

Set Host RS-232 port user defaults:

```
PROFILE"HOST_PORT","BPS"="115200"
PROFILE"HOST_PORT","MODEM_INSTALLED"="CHECK"
```

Set modem user defaults:

```
PROFILE"HOST_MODEM","DIAL"="ATDP"
PROFILE"HOST_MODEM","COMMAND_DELAY"="1"
```

Startup Job

The DT800 can automatically run a user-defined job

- every time it is restarted by a firm reset — see "ONRESET.DXC" below, or
- every time a PC Card containing a special version of the job is inserted into the DT800 — see "ONINSERT.DXC" below.

ONRESET.DXC

To make the DT800 automatically run a job as the DT800's startup job, send the command

```
RUNJOBONRESET"JobName"
```

to the DT800. This places the contents of the job named ***JobName*** (must already exist in the DT800) into a file named ONRESET.DXC in the root directory of the DT800's B: drive. Then the DT800 automatically runs this file every time it is restarted by a firm reset[12]. (Alternatively, you can copy an ONRESET.DXC file directly to the DT800's root directory from a memory card, by FTP transfer, or over Ethernet.)

You can delete the working copy of ONRESET.DXC from the DT800's SRAM memory, which stops this startup job function, by sending

```
DELONRESET
```

You can also backup the ONRESET.DXC file — see "Protecting Startup Files" on page 116. Note that if you delete the working copy, the automatic replacement-on-reset function described on page 116 ceases.

You cannot alter the contents of an ONRESET.DXC file. You must create a new file directly in the DT800 using the **RUNJOBONRESET"*JobName*"** command again.

ONINSERT.DXC

When a memory card is inserted into a DT800, the DT800 first looks for a subdirectory on the card named with its own serial number. If it finds such a subdirectory, it automatically loads and runs any ONINSERT.DXC file it finds in the subdirectory.

If there is no subdirectory named with the DT800's serial number, the DT800 automatically loads and runs any ONINSERT.DXC file it finds in the root directory of the card.

This auto-programming function means that you can insert a single memory card into a number of DT800s, one at a time, and

- automatically program all the DT800s 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 DT800s 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 — DT800s that do not find a subdirectory named with their serial number automatically load and run the ONINSERT.DXC file at the root level, and DT800s that find their specific subdirectory automatically load and run the ONINSERT.DXC file found there.

[12] A singlepush reset —see page 118.

You can create an ONINSERT.DXC file for use with a specific DT800 by sending

RUNJOBONINSERT"*JobName*"

while the card is inserted in that DT800. This copies ***JobName*** into the specific DT800 serial number directory on the card (named, for example, SN080271; creates directory if it does not exist) and calls it ONINSERT.DXC.

You can create a generic ONINSERT.DXC file (for use with any DT800) by sending

RUNJOBONINSERTALL"*JobName*"

while the card is inserted in the DT800. This copies ***JobName*** into the card's root directory and calls it ONINSERT.DXC. Then, when the card is inserted into a DT800, if no DT800-specific directory exists, the generic ONINSERT.DXC file is run.

Deleting ONINSERT.DXC

The following delete commands are available for deleting ONINSERT.DXC:

DELONINSERT	Deletes ONINSERT.DXC from the DT800's serial number directory on the card
DELONINSERTALL	Deletes ONINSERT.DXC from the root directory on the card

Protecting Startup Files

The DT800 has

- an automatic mechanism for protecting its USER.INI (user startup profile) file
- an optional mechanism for protecting its ONRESET.DXC (startup job) file

by making backup copies of these in Flash memory. (The original/working files reside in SRAM memory.) See Figure 70.

Then, whenever you carry out a firm reset, the DT800 automatically overwrites the working version (possibly corrupt) in SRAM memory with a copy of the backup version (clean/uncorrupt) from Flash memory and runs the new, clean copy in SRAM. See Figure 71.

Protecting USER.INI

This process is completely automatic.

The first time you send a **PROFILE...** command to the DT800, USER.INI (the working copy) is created in SRAM memory, and a backup copy of it is created in Flash memory. Subsequently, every **PROFILE...** command you send

- updates the working version of USER.INI
- causes the backup in Flash to be overwritten with a copy of the updated USER.INI.

In this way, the DT800 maintains an up-to-date backup of your latest profile settings.

Then, whenever a firm reset occurs, the DT800 automatically replaces its working USER.INI with a copy of the latest, clean backup version.

If you want to force the DT800 to make a Flash backup copy of its current USER.INI, send the command

FLASHUSERINI

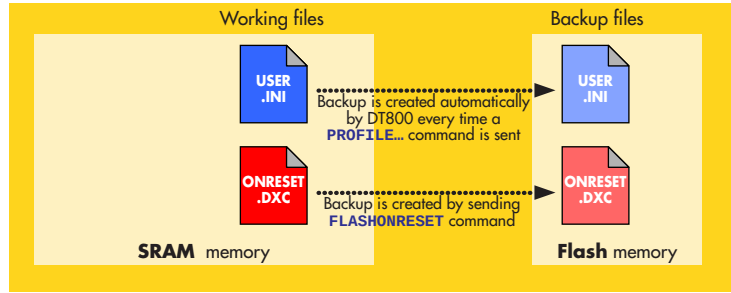


FIGURE 70 Protecting startup files 1 — backups are created

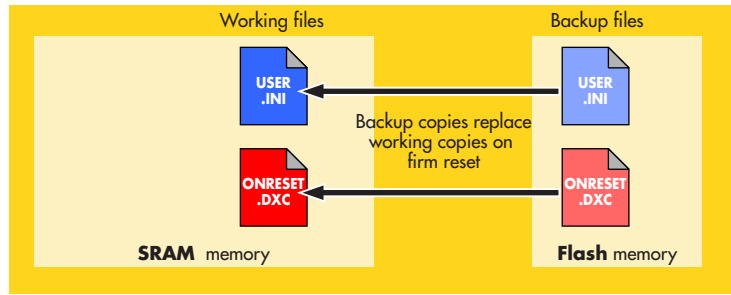


FIGURE 71 Protecting startup files 2 — backups replace working files on firm reset

Protecting ONRESET.DXC

To protect ONRESET.DXC send the command

FLASHONRESET

to the DT800. A copy of ONRESET.DXC is created in the DT800's Flash memory, and automatically replaces the working copy (in SRAM) every time a reset occurs.

If you change ONRESET.DXC, remember to send the **FLASHONRESET** command again to backup the new file.

Deleting the Backup Files from Flash

To delete the backup copy of USER.INI from the DT800's Flash memory, send

DELFLASHUSERINI

A backup will be re-created the next time you send a **PROFILE…** command.

To delete the backup copy of ONRESET.DXC from the DT800's Flash memory, send

DELFLASHONRESET

Without the backup file in Flash, the automatic protection mechanism described above no longer occurs.

Setting the DT800's Clock/Calendar

The DT800's real-time clock/calendar is based on a 24-hour clock that has a resolution of 122µs.

To make the timestamps and datestamps that the DT800 includes with your readings meaningful, you'll probably want to set the DT800's clock/calendar to your local time and date. This is described in the next two topics.

Of course, you can use the DT800 without setting its clock/calendar. When the date and time have not been set:

- Any time value included with your data is the interval since the DT800 was powered up — that is, power-up = 00:00:00. For example, a time value of `01:15:32` means 1 hour, 15 minutes and 32 seconds since power-up.
- Any date value included with your data is the DT800's **base date** (01/01/89) plus the days since it was last powered-up. For example, a date value of 01/09/89 (North American format) signifies 8 days since power-up.

Setting the DT800's Time (T=)

When setting the *dataTaker*'s clock you must use the time format defined by P39 and P40. For example, if P39=2 (in this case P40 does not matter), then the clock time must be set as a decimal value:

T=11.7528

Time is maintained through both software and hardware resets.

Time and date stamps can be added to real-time data and to logged data — see **/T** and **/D** in "Switches" on page 111. Time and date are automatically logged whenever data is stored.

Setting the DT800's Date (D=)

When setting the *dataTaker*'s calendar you must use the date format defined by P31. For example, if P31=2, then the date must be set in North American format:

D=02/01/2000

The DT800's date is maintained through both software and hardware resets.

Time and date stamps can be added to real-time data and to logged data — see **/T** and **/D** in "Switches" on page 111. Time and date are automatically logged whenever data is stored.

Setting Date and Time Together (DT=)

Use the **DT=** command to set the DT800's date and time simultaneously, independent of P31 (date format) and P39 (time format). The command's formats match those of the date and time fields of fixed-format mode records. The command looks like

DT=[*YYYY*/*MM*/*DD*,*hh*:*mm*:*ss*]

where

YYYY	is the year	
MM	is the month	Date
DD	is the day	
hh	is the hour	
mm	is the minute	Time
ss	is the second	

Here's an example:

DT=[2000/11/23,13:09:40]

Resetting the DT800

The DT800 provides the following methods for clearing and initializing its sub-systems:

Command/Action		Effect of Command/Action: Communications Environment — Current Comms Connections	Effect of Command/Action: Communications Environment — Comms Parameters	Effect of Command/Action: Program/Job Environment (Current Job/Program, CVs, IVs, $s, Spans and Polynomials)	Effect of Command/Action: Logged Data and Logged Alarms	Effect of Command/Action: ONRESET.DXC (Startup Job)	
RESET	Command sent to the DT800	Maintained	Maintained	Cleared	Maintained	Not run	**Soft** reset — simply clears your current work (the program/job environment).
SINGLEPUSH	Command sent to the DT800	Disconnected (unless, for Host RS-232 comms only, the DT800's RS-232 connection settings match its factory defaults; TCP/IP is always disconnected)	**Uses PROFILE... defaults** (see "User Startup Profile" on page 113)	Cleared	Maintained	Run (see "Startup Job" on page 115)	**Firm** resets — like soft reset, but DT800 restarts with user defaults and user startup job. Three methods of achieving the same result.
Hardware Reset ("hardware singlepush")	**One** push of the hardware reset button (see "Manual Reset Button" below)						
Power-Up Reset	Remove then replace all main power (external supply and internal main battery).				Maintained (unless internal backup battery is also removed)		
Triple-Push Reset	**Three** pushes of the hardware reset button **within 10 seconds** (see "Manual Reset Button" on page 119)	Disconnected (unless, for Host RS-232 comms only, the DT800's RS-232 connection settings match its factory defaults; TCP/IP is always disconnected)	Uses factory defaults (see "Factory Defaults" on page 119)	Cleared	Maintained	Not run	**Hard** reset — like firm reset, but DT800 restarts with factory defaults and does not load a new job. Ignores (does not delete) user defaults. For use in case of serious problems (for example, a faulty startup job, or inability to communicate with the DT800 because of invalid user defaults).
FACTORYDEFAULTS	Command sent to the DT800	Disconnected	Uses factory defaults (see "Factory Defaults" on page 119)	Cleared	Maintained	Deleted (and not run)	Like triple-push reset but also deletes user defaults.
FORMAT"B:"	Command sent to the DT800	Maintained	Maintained	All jobs (including the current job) are halted and removed; program/job environment is reset	Cleared (internal memory only; not PC Card)	Not run	For low-level system recovery and purging. Reformats SRAM RAM disk (see Figure 18 on page 30) and rebuilds standard directory structure.

Your logged data and alarms are not deleted.

Table: DT800 Resets

See also "Memory" on page 30.

Maintaining Parameters and Switches through a Reset

You can use **PROFILE...** commands to re-apply parameter and switch settings after every soft and firm reset. See the **PARAMETERS** and **SWITCHES** sections in the DT800 PROFILE Details table (page 113).

Wait after RESET

Do not send any commands to the DT800 for five seconds after sending either the **RESET** command or the **SINGLEPUSH** command.

For example, use the DeTransfer command **\W*n*** to force a pause after a **RESET** command:

```
RESET
\W5
```

Manual Reset Button

You initiate a hardware reset and a triple-push reset by pressing the DT800's manual reset button. To do this, insert a straightened paper clip (or similar object) through the small hole located between the DT800's Host RS-232 port and the USB port — see Figure 12 on page 25.

Factory Defaults

Both the triple-push reset and the **FACTORYDEFAULTS** command restart the DT800 using its factory defaults, which are kept in the DT800's Flash memory. (Some of these settings are listed in the Factory Default column of the DT800 PROFILE Details table on page 113.)

A triple-push reset ignores and does not delete any user defaults (USER.INI and ONRESET.DXC), but the **FACTORYDEFAULTS** command does the following:

- deletes USER.INI (working) from SRAM memory
- deletes USER.INI (backup) from Flash memory
- deletes ONRESET.DXC (working) from SRAM memory
- deletes ONRESET.DXC (backup) from Flash memory

To return a DT800 to its totally original, "as shipped" state, send the **FACTORYDEFAULTS** command, reconnect, then send the **FORMAT"B:"** command.

LEDs and Messages After a Reset

The DT800 does the following after you reset it (see also "LED Sequence on Startup" on page 28):

	Command /Action	LED Activity	Message Returned
Soft reset	**RESET**	Acquire LED flashes immediately, then resumes heartbeat flash*. Also, if externally powered, Charge LED off momentarily, then on.	`RESET` `Initializing...Done.`

Firm resets	**SINGLEPUSH**	All LEDs flash rapidly four times Then Attention LED on briefly, Acquire LED resumes heartbeat flash* and, if externally powered, Charge LED on.	`SINGLEPUSH`
			`dataTaker 800 Version 4.00`
			`Initializing...Done.`

	Hardware reset		`dataTaker 800 Version 4.00` `Initializing...Done.`
	Power-up reset		`dataTaker 800 Version 4.00` `Initializing...Done`

Hard reset	Triple-push reset	All LEDs flash rapidly four times, then Acquire LED resumes heartbeat flash*.	`dataTaker 800 Version 4.00`
			`Initializing...Done.`
			`=== SAFE MODE ===`
			`Now using factory default settings.`
			`Reset will enable user settings.`

	FORMAT"B:"	No change	`FORMAT"B:"`
			`Formatting drive B:. Please wait...`
			`Format of drive B: successful.`

* Heartbeat flash: see step 3 on page 28.

TEST COMMANDS

The **TEST...** commands force the DT800 to autozero itself, check the functionality of its hardware, and return a test report:

TEST	Returns a full test report	
TEST*n*	Returns line ***n*** of the test report	
TESTR	Forces continuous return cycles	Stop continuous reporting by sending a carriage return.
TESTR*n*	Forces continuous return cycles of line ***n***	

Test results that are out of range are flagged with a FAIL message.

Test Report (DT800 Health)

A typical test report returned when the DT800 is in free-format mode is shown in the first column of the following table. (When the DT800 is in fixed-format mode, less-verbose comma-separated results are returned.)

TEST Report	Line (n)	Description	Valid Range
TEST report generated on 05/12/2000 12:26:22			
dataTaker 800 Version 4.00.0001 Flash 2001/11/24 16:42:23	**0**	*dataTaker* model, firmware version (see "Upgrading DT800 Firmware" on page 193), version datestamp and timestamp	
Serial Number: 080325	**1**	*dataTaker* serial number	
Peripherals: None	**2**	None, or list of attached peripherals	
PC Card Installed: Nokia GSM Modem	**3**	None, or inserted PC Card type	
External Supply 13.4 V N/A	**4**	Voltage supplied from External Battery terminals	0 to 28V
Battery Voltage 14.4 V PASS	**5**	Voltage supplied by internal main battery	10.5 to 20V
System Voltage 12.9 V PASS	**6**	Voltage supplied by internal charger	9.5 to 30.5V
Lithium Battery Voltage 3.7 V PASS	**7**	Voltage supplied from internal memory-backup battery	3.2 to 4.3V
Analog Pos Supply 18.8 V PASS	**8**	Analog multiplexer positive supply voltage	18.0 to 20.0V
Analog Neg Supply -19.0 V PASS	**9**	Analog multiplexer negative supply voltage	–20.0 to –18.0V
Analog 5V Supply 5.3 V PASS	**10**	Analog positive 5V supply voltage	4.8 to 5.6V
Analog Ref Voltage 2.5 V PASS	**11**	Analog reference voltage	2.450 to 2.550V
Analog Zero Voltage -0.0 mV PASS	**12**	Analog ground voltage	–20.0 to 20.0mV
Common Mode Rejection Ratio 92.3 dB PASS	**13**	Common-mode rejection ratio	>80dB
dataTaker's health PASS	**14**	Overall *dataTaker* health: PASS or FAIL — FAIL does not necessarily mean that the DT800 cannot measure correctly (but you should investigate the FAIL lines in the TEST report).	

Table: DT800 TEST Report

Event Log

Background record-keeping of critical events

To aid in troubleshooting, the DT800 automatically logs significant events (power failures, temperature extremes, resets, program failures,...) into an **event log**, which is a file named EVENT.LOG in the EVENTS directory of the DT800 file system.

The event log may help you pinpoint the cause of any unexpected readings or failures, and will be used by *dataTaker* engineers if you ever have to return your DT800 for service.

The size of the event log file is limited to 500 entries. When EVENT.LOG is full, the DT800 makes a copy of it (overwriting any existing backup), names the copy EVENT.BAK, and creates a new log file. In this way the DT800 retains

- the most recent 500 events (in EVENT.LOG), and
- the previous 500 events (in EVENT.BAK).

You can also automatically log results of the **TEST** command to the event log — see **P10** on page 107.

Unloading the Event Log

Send

UEVTLOG

to unload the event log to the host computer. This command instructs the DT800 to return the entire contents of the event log. For example:

```
EVENT           ,2001/03/29,14:32:26.042234,"Event log created."
EVENT           ,2001/03/29,14:32:33.567891,"FORMAT 2.00.000065"
EVENT           ,2001/03/29,14:33:02.418964,"Reset 2.00.000065"
SELF_DIAGNOSTIC,2001/04/19,01:13:46.228619,"datafilemap.cpp 119"
EVENT           ,2001/04/24,19:23:09.624897,"Main power brownout"
⇓
```

The format of the information returned from the event log is the same whether the DT800 is in free-format mode or fixed-format mode.

Events — the Whole Log

The event log can only be unloaded in its entirety — it's not possible to unload just a defined timespan of the log. Once an unload of the event log has started, it must go to completion. There is no command to quit an unload part way through.

Clearing the Event Log

Clear the event log by sending

CEVTLOG

to the DT800. This command clears the EVENT.LOG file and initializes it with an "Event log created" entry such as

```
EVENT,2001/04/23,14:32:26.042234,"Event log created"
```

This entry is the first item unloaded in subsequent unloads of the event log.

STATUS Commands

STATUS

The STATUS command returns a report showing the status of the DT800's schedules, channels, alarms, memory and logging to the host computer.

The first line of the report shows the version, creation date and creation time of the DT800's firmware. The last line reports the DT800's current switch settings (see "Switches" on page 111"). Send the **/u** switch to make STATUS results less verbose.

STATUS Report	Line (n)
dataTaker 800 Version 4.00.0001 Flash 2001/02/18 21:28:18	0
none,F Scan Schedules Active,Halted	1
0,0 Alarms/IFs Active,Halted	2
0 Polynomials/Spans Defined	3
none,none Scan Schedules LOGON,LOGOFF	4
13650,0 Internal kB free,used	5
169260,0 External kB free,used	6
/A/B/C/d/E/f/G/h/I/J/K/l/M/N/o/r/S/t/U/v/w/x/y/Z	7

STATUS*n*

Each STATUS line can be returned individually. STATUS2 and 4 return extra information. There are also other status levels that are not returned by the general STATUS command (STATUS10, 12 and 14).

STATUS1

Returns version details of the DT800's current operating system. For example

```
dataTaker 800 Version 4.00.0001 Flash 2001/02/18 21:28:18
```

STATUS2

Returns information about the DT800's active halted schedules. For example

```
A, none Scan Schedules Active,Halted
```

STATUS4

Returns the DT800's current defined polynomials and spans. For example

```
2 Polynomials/Spans Defined
Y1=3.54,1.009"Deg C"
S7=0.0,100,0.0,1.0"kPa"
```

STATUS5

Returns the data logging status (lists schedules logging and not logging).

STATUS6

Returns the amount of free and used space in the DT800's internal memory (kB).

STATUS7

Returns the amount of free and used space in an inserted memory card (kB).

STATUS9

Returns the DT800's current switch settings. For example

```
/a/B/C/d/E/f/G/h/I/J/K/l/M/N/o/R/S/t/U/v/w/x/y/Z
```

STATUS10

Returns additional information about the current program in the DT800. For example

```
27113,1989,1,0,"",<A,"2S",H,<"Dry bulb","",0,0,5,4,3>,<"Wet
bulb","",0,0,5,4,3>,<"Humidity","%RH",0,0,5,4,3>>,<B>,<C>,<D
>,<X>
```

This comma-separated list provides details about the DT800's program. In order, they are program ID, base year, time resolution, card status and current $ string, followed by schedule fields that identify individual channels, their format and their units.

If there is no program the following is returned:

```
5,2000,1,0,"&",<*>,<X>,<A>,<B>,<C>,<D>,<E>,<F>,<G>,<H>,
<I>,<J>,<K>,<S>
```

STATUS12

Returns the date/time range of logged data — that is, the time and date of the first and last data points stored in the DT800's internal memory and inserted memory card. For example

```
00:11:33 on 05/11/1992,00:13:00 on 19/01/1993 Data Start,End
times
```

STATUS14

An extended version of STATUS10.

CHARAC Commands

The two techniques of automatic **autozeroing** (see "Speed versus Accuracy" on page 188) and **characterization** (next topic) keep the DT800 accurate. The DT800 requires no manual adjustments or other relatively unreliable techniques to stay calibrated.

Characterization

When each DT800 is born at the factory, it is firstly "burnt-in" to stabilize its electronic components against failure and drift due to aging. Then, prior to shipping, it is further calibrated by a process known as "characterization":

- The first part of this process is to read (and store in the DT800's EEPROM — see "Memory" on page 30) the identification numbers of some of the its internal components and assemblies.
- The second part of the process is to measure, for each individual DT800, the quantities listed in the DT800 Characterization Report below and compare them against certified absolute values. By comparing each measured value with its absolute value, a "characterization offset" is calculated for each quantity. In this way, the response of each individual DT800 to real-world stimulation is quantified and stored in the DT800's EEPROM. Then, during actual data acquisition, the offsets are automatically applied to the appropriate measurements.

These characterization offsets and identification numbers are returned by the following commands:

CHARAC	Returns the characterization report
CHARAC*n*	Returns line ***n*** of the characterization report

In the event of a problem, your *dataTaker* representative may ask you for **CHARAC** information. He can also arrange re-characterization of your DT800 if required.

Characterization Report (DT800 Calibration Factors)

A typical characterization report is shown in the following table.

When the DT800 is in fixed-format mode (**/H**), less-verbose comma-separated results are returned.

Characterization Report	Line (n)
dataTaker DT800 Version 4.00.0001 2001/02/18 16:55:53	0
Serial Number 080015	1
Kernel Assy # AS1156A0-000	2
Kernel Prodn # 0981-048	3
Kernel Characterized on 1999/11/10 14:06	4
RAM Size 4096 kB	5
RAM Speed 100 nS	6
Flash Size 2048 kB	7
Flash Speed 120 nS	8
Ethernet Address 00-90-2E-FF-FF-FF	9
Analog Assy # AS1157A0-000	10
Analog Prodn # 0978-004	11
Analog characterized on 1999/11/10 14:06	12

Table: DT800 Characterization Report (sheet 1 of 2)

Characterization Report (Continued)	Line (n)
ADC Gain 0 Zero -2.042155e+04 mV	13
ADC Gain 1 Zero -1.134232e+04 mV	14
ADC Gain 2 Zero -5.671932e+03 mV	15
ADC Gain 3 Zero -2.269055e+03 mV	16
ADC Gain 4 Zero -1.009365e+03 mV	17
ADC Gain 5 Zero -5.049590e+02 mV	18
ADC Gain 6 Zero -2.023327e+02 mV	19
ADC Gain 7 Zero -1.013634e+02 mV	20
ADC Gain 8 Zero -5.706687e+01 mV	21
ADC Gain 9 Zero -2.095882e+01 mV	22
ADC Gain 10 Zero -1.040179e+01 mV	23
ADC Gain 0 Slope 6.229697e-01 mV/b	24
ADC Gain 1 Slope 3.460011e-01 mV/b	25
ADC Gain 2 Slope 1.730177e-01 mV/b	26
ADC Gain 3 Slope 6.920385e-02 mV/b	27
ADC Gain 4 Slope 3.079038e-02 mV/b	28
ADC Gain 5 Slope 1.539662e-02 mV/b	29
ADC Gain 6 Slope 6.158512e-03 mV/b	30
ADC Gain 7 Slope 3.079199e-03 mV/b	31
ADC Gain 8 Slope 1.727936e-03 mV/b	32
ADC Gain 9 Slope 6.262056e-04 mV/b	33
ADC Gain 10 Slope 3.039779e-04 mV/b	34
Excite Path Resistance 1.000000e+02 Ohm	35
Return Path Resistance 1.000000e+02 Ohm	36
DAC Excite Zero 0.000000e+00 mV	37
DAC Excite Slope 4.028320e+00 mV/b	38
DAC Excite Full 1.650000e+04 mV	39
DAC Trigger Zero 0.000000e+00 mV	40
DAC Trigger Slope 4.028320e+00 mV/b	41
DAC Trigger Full 1.650000e+04 mV	42
PCMCIA Assy # AS1158A0-000	43
PCMCIA Prodn # 2323-789	44
PCMCIA Characterized on 1999/11/10 14:06	45
Charger Assy # AS1159A0-000	46
Charger Prodn # 12324-654	47
Charger Characterized on 1999/11/10 14:06	48
Terminal Assy # AS1160A0-000	49
Terminal Prodn # 0980-025	50
Terminal Characterized on 1999/11/10 14:06	51
Temperature Offset 0.00 degC	52
Charac'd at Temperature 23.00 degC	53

Table: DT800 Characterization Report (sheet 2 of 2)

Part I — Communications

The DT800 has the following mechanisms for communicating with other devices (usually a host computer):

RS-232	See • "DT800 RS-232 Basics" on page 125 • "DT800 Direct (Local) RS-232 Connection" on page 129 • "DT800 Modem (Remote) RS-232 Connection" on page 129.
Ethernet	See "DT800 Ethernet Communications" on page 134.
FTP	See "DT800 FTP Communications" on page 138.
USB	See "DT800 USB Communications" on page 138.
PPP	See "DT800 PPP Communications" on page 138.

Automatic Comms Port Arbitration

You can have more than one type of communications link connected to the DT800 at the same time. In this situation, the DT800 automatically switches between each link as required, responding back through the link from which the most recent communication was received.

A simple CR (carriage return) character or LF (line feed) character received from a link is sufficient to switch the DT800's connection to that link. Before switching, the DT800 notifies the host on the current link that it is about to change connection.

Password Protection — Comms Ports

You can protect the DT800's communications ports — the **Host RS-232** port, the **Ethernet** port and the **USB** port[13] — with a user-defined password so that communication through these ports is only possible after the password is entered.

Setting and Removing a Comms Port Password

Set a password by sending

`PASSWORD="password"`

to the DT800. ***password*** can be any text string (except command keywords) of up to 10 case-sensitive characters.

Remove a password by sending

`PASSWORD=""`

(two **"** characters, no space character between them).

Note The password is lost ("removed") if the DT800 performs a power-up reset, triple-push reset or **SINGLEPUSH** reset (see the DT800 Resets table on page 118).

Accessing Password-Protected Comms Ports

Any time you want to establish communication, simply send the password followed by a carriage return. If the password is correct, the DT800 responds with `Accepted` and opens the comms ports.

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.

Are the Ports Protected?

Simply send the command

`PASSWORD`

to determine if a comms port password has been set. If the DT800 responds with

- `PASSWORD 0` — no password has been set
- `PASSWORD 1` — a password has been set.

Regardless of the password state, the DT800 responds to the **DEL** character with `<< CR LF`. This response can be used to determine the baud rate setting of the DT800 even when a comms port password is active.

[13] The USB port is not yet operational — see "DT800 USB Communications" on page 138.

RS-232 Communications

Quick Start

- To connect your DT800 directly to a local host computer, go to "Setting Up a Direct Connection" on page 129.
- To connect your DT800 to a distant host computer using a modem communications link, go to "Setting Up a Remote Connection" on page 132.

DT800 RS-232 Basics

Host RS-232 Port

The DT800 has a 9-pin male connector for RS-232 serial communication to a computer (either directly, or by means of a pair of modems (Figure 72). This DTE interface, labelled **Host RS-232**, is the primary means by which you

- program (configure) and supervise the DT800 from the host computer
- retrieve stored data from the DT800 to the host computer.

The Host RS-232 port of the DT800 is electrically isolated from the rest of the system — see "DT800 Analog Sub-System" on page 148.

Host RS-232 Port Pinout

The pinout of the DT800's Host RS-232 connector is shown in Figure 72. The signal functions are defined in the RS-232 Pinouts table on page 190.

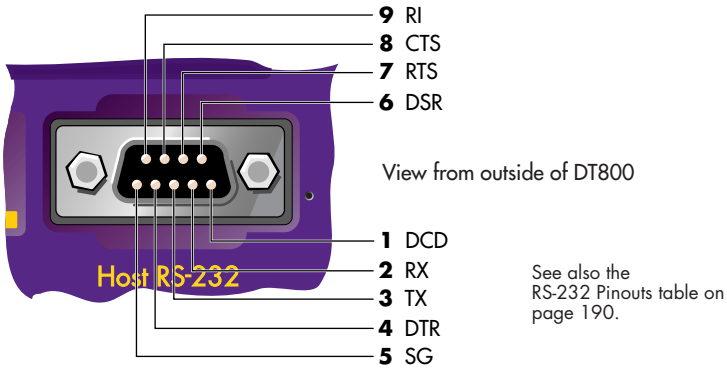


FIGURE 72 DT800 Host RS-232 port — connector pinout (DTE)

Automatic Device Detection

A DT800 running version 4.00 (or later) firmware uses the state of its Host RS-232 port's DSR terminal (Figure 72) to determine the type of device connected to the port as follows:

- If the DT800's DSR terminal is NOT held active by the connected device, the DT800 assumes that it's connected directly to the host computer (Figure 73) and operates accordingly.

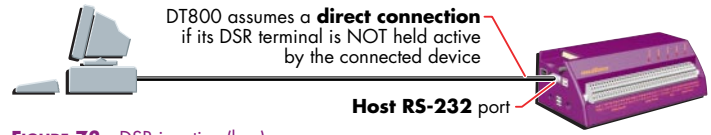


FIGURE 73 DSR inactive (low)

- If the DT800's DSR terminal IS held active by the connected device, the DT800 assumes that it's connected to a modem (Figure 74) 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.

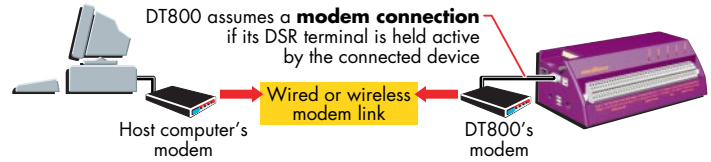


FIGURE 74 DSR active (high)

This feature greatly simplifies your RS-232 computer-to-DT800 comms link, which is discussed later in "DT800 Direct (Local) RS-232 Connection" (page 129) and "DT800 Modem (Remote) RS-232 Connection" (page 129).

Note A version 4.00 (or later) DT800 does not change its baud rate or flow control settings when it detects a modem. (Previous versions of the DT800 firmware set the DT800's Host RS-232 port to 57600 baud and enabled both hardware flow control and software flow control.)Host RS-232 Port Settings

The DT800 is shipped with its Host RS-232 port set to **57600** baud, **N**o parity, **8** data bits, **1** stop bit, and **S**oft**W**are **F**low **C**ontrol. If you need to change these settings, there are two methods you can use:

- Use one or more **PROFILE...** commands to "permanently" change the settings. The changes are stored in the DT800's USER.INI file and loaded every time the DT800 is restarted. See the HOST_PORT section of the DT800 PROFILE Details table (page 114).
- Send a **PH=** command — see "PH Configuration Commands" below. Doing this changes the settings "temporarily". That is, the DT800 uses the new settings only until you restart it, or until you send another **PH=** command.

Host RS-232 Port Commands

The following commands are available for configuring the DT800's Host RS-232 port, and for controlling the dial-out process of the modem connected to the DT800:

PH Configuration Commands

The **PH=** commands (**PH** ⇒ Port Host) set the comms attributes of the DT800's Host RS-232 port. (Note that **PH=** settings are not remembered through a reset; use HOST_PORT **PROFILE...** commands for settings that you want to be permanent.)

Command	Action	*b*	*p*	*d*	*s*	*f*
PH=*b*	Sets Host RS-232 port baud rate (DT800 default = 57600)	**50**, **75**, **110**, **150**, **300**, **600**, **1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600**, or **115200**	**N** (none), **O** (odd), or **E** (even)	**8** or **7**	**1** or **2**	**NOFC** (no flow control), **HWFC** (hardware flow control), or **SWFC** (software flow control) **SWHW** (both hardware and software flow control)
PH=*b*,*p*	Sets Host RS-232 port baud rate and parity					
PH=*b*,*p*,*d*	Sets Host RS-232 port baud rate, parity and databits					
PH=*b*,*p*,*d*,*s*	Sets Host RS-232 port baud rate, parity, databits and stopbits					
PH=*b*,*p*,*d*,*s*,*f*	Sets Host RS-232 port baud rate, parity, databits, stopbits and flow control					
PH	Returns the current Host RS-232 port comms settings (for example: `57600,N,8,1,NOFC`)					

Table: DT800 Host RS-232 port — configuration commands

SETDIALOUTNUMBER Command

Send the command

SETDIALOUTNUMBER

to the DT800 to specify the telephone number to be dialled by the **DIAL** command to establish a connection to the host computer.

DIAL Command

Sending the command

DIAL

to the DT800 causes it 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 DT800 to dial out when an alarm condition arises (see "Alarm Action Processes" on page 100).

HANGUP Command

Sending the command

HANGUP

to the DT800 causes it 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 DT800 to hangup a call in progress when an alarm condition arises (see "Alarm Action Processes" on page 100).

Example — Modem Control Commands

The use of the DT800's modem control commands is demonstrated in the following program:

```
BEGIN
SETDIALOUTNUMBER"12345678"
   RA10M
   'Read boiler temp
   1TK(=1CV,W)
   IF(1CV>120){[DIAL]}
END
```

Every 10 minutes, the program instructs the DT800 to initiate a dial-out to phone number **12345678** if the boiler temperature is greater than 120°C.

ASCII Comms

All communications with the DT800 use the ASCII character set. The eighth bit is normally a **0** (zero), but an extension to the character set (for the text strings and for special display characters) is possible if this bit is set to a **1**. For all commands other than switches and text strings, the DT800 ignores lowercase characters.

Echo

By default, the DT800 **echoes** commands that it receives. That is, it automatically transmits received commands back to the host (see also "echo" on page 206).

You can turn this function off by sending the echo switch **/e** (see page 111). Also, it's forced off when the DT800 is in fixed-format mode for returned data (see page 21).

Flow Control

Flow control is the means by which communicating devices (such as the DT800 and its host computer) control each other's transmission of characters to avoid data loss. Flow control causes the receiver to disable transmissions by the sender if the receiver's input buffer is at risk of overflowing and thereby losing data.

The DT800 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** (NFC)
- **SWHW** (both software and hardware flow control)

You'll often use the DT800 set to SWFC, which is its default. But there may be times when you'll have to change the DT800 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 75 summarizes the recommended flow control settings for the two general types of RS-232 connections between a DT800 and its host computer.

Software Flow Control (SWFC)

In SWFC mode, the receiver controls the flow of characters by transmitting

- the XOFF character (ASCII 19 or Control Q) to stop the sender from sending further characters
- the XON character (ASCII 17 or Control S) to allow the sender to resume sending characters.

During data return to the host computer, if the DT800 receives an XOFF character from the host, it stops transmission within two character periods. Then when the host has processed the buffered data and is ready to receive again, the host should transmit an XON character to the DT800. This is the "resume transmission" signal to the DT800. (If the DT800 is left in the XOFF mode by the host at the end of a transmission session, the DT800 carries out an auto-XON after 30 seconds — see P26 on page 108.)

Similarly, the DT800 can control the transmission of commands and programs sent to it from the host computer. To do this, the DT800 issues

- an XOFF character when its input buffer is 50% full (and at 75% full and 90% full)
- an XON character when its input buffer is empty.

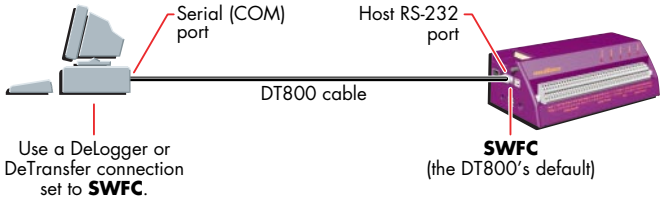
Hardware Flow Control (HWFC)

In HWFC mode, the transmission of characters is managed by the RTS (Ready 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 its RTS line when it's 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 DT800 communications cable (product code IBM-6) has the RTS/CTS lines connected in this crossover manner — see Figure 139 (page 191).

HWFC is inherently more reliable than SWFC and is therefore preferred, especially if there is any line or other noise on the communications link. SWFC can become confused if the flow control characters are corrupted or lost, whereas HWFC has constant levels that enforce the current flow control state at all times, making it highly resistant to line and other noise.

Direct connection
Use SWFC at both ends.



Modem connection
For maximum reliability when using modems, we recommend HWFC for all devices.

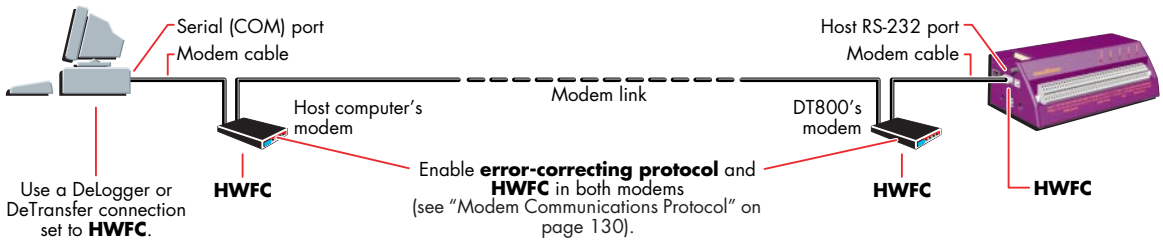


FIGURE 75 Recommended flow control settings for the two types of connections

No Flow Control (NFC)

The DT800 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 DT800's SWHW setting is provided for backwards compatibility. It enables both software flow control and hardware flow control at the same time.

Special Characters

The following characters have special significance for the DT800:

XOFF		ASCII 19	Stops DT800 transmitting
XON		ASCII 17	Allows DT800 to transmit
BS	Backspace	ASCII 8	Deletes previous character (DT800 echoes BS BS)
DEL	Delete	ASCII 127	Used to auto-baud detect (DT800 echoes << CR LF)
CR	Return	ASCII 13	Terminates a command line (DT800 echoes CR LF)
LF	Line feed	ASCII 10	Ignored
	Space	ASCII 32	Command separator
	Tab	ASCII 9	Command separator
'	Single quote		Designates that what follows is a comment only (comment ends at next carriage return)

Input Buffer (How the DT800 Receives and Processes a Program)

The DT800's input buffer can hold a maximum of 250 characters at any one time. This means that any single command or line of a program that you send to the DT800 must be shorter than 250 characters. So when writing long command and/or comment lines, be sure to keep their length below 250 characters.

A command or program line is terminated by a carriage return character (this is how the DT800 recognizes the end of the command or line), and the DT800 begins to process the input buffer when each carriage return is received. A full 250 characters of program takes up to 500ms to process if the DT800 is not scanning, and up to 5 seconds if it is running long schedules and many alarms. Any digital delay periods (such as **1DSO(1000)=0**) or general delay periods (such as **DELAY=1000**) add to this time.

The host must ensure that the DT800 has sufficient time to process each line of a downloaded program. This is achieved by using either

- software flow control, or
- hardware flow control, or
- time delays between transmissions if no flow control is used.

Comms Wakes the DT800

If you communicate with the Host RS-232 port of the DT800 while it's in sleep mode (low-power mode), the first few characters you send wake the DT800. But characters received in the first 75ms from the start of transmission are either lost or may generate communications errors, so don't send important commands until the DT800 is fully awake.

To reliably wake a DT800 that may be in sleep mode, we recommend that you send a carriage return or line feed character then wait 300ms before sending any commands. You can include these instructions at the start of your program.

See also "Low-Power Operation" on page 43.

DT800 Direct (Local) RS-232 Connection

A common (and the most simple) way of communicating with the DT800 (for configuring it, programming it and retrieving data from it) is to connect its Host RS-232 port directly to a serial port on the host computer using the RS-232 comms cable supplied. This is known as a **direct connection** to a **local** DT800 (Figure 76).

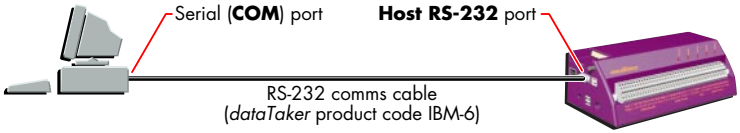


FIGURE 76 Direct (local) connection

The information in "DT800 RS-232 Basics" (begins on page 125) applies to a direct connection.

For recommended flow control settings for this type of connection, see Figure 75 (page 127).

Direct RS-232 Cable

Communications cables for connecting the DT800 (a DTE device) directly to a 9-pin computer comms port or 25-pin computer comms port (DTE devices) are detailed in Figure 139 (page 191). A suitable 9-pin cable (*dataTaker* product code IBM-6) is supplied with your DT800 for this purpose.

Because the DSR line is not connected (inactive) in these cables, the DT800 automatically assumes a direct connection to the host computer — see "Automatic Device Detection" on page 125 (Figure 73 in particular).

Cable Length

Although the RS-232 standard specifies a cable of not more than 4 metres (15 feet), you can use longer cables. 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 you may have to use a slower baud rate.

Setting Up a Direct Connection

To set up a direct connection:

1. Connect a suitable comms cable between a serial port on the host computer and the DT800's Host RS-232 port as shown in Figure 76.
   Suitable cables are discussed in "Direct RS-232 Cable" above.
2. Run suitable *dataTaker* or terminal software — for example, DeTransfer, DeLogger or HyperTerminal.
3. Configure the software to communicate using the same communications parameters that the DT800 is set to (it defaults to 57600 baud, 8 data bits, 1 stop bit, no parity and software flow control).
4. Get the software to communicate by means of the computer comms port that is connected to the DT800.

DT800 Modem (Remote) RS-232 Connection

Another common way of communicating with the DT800 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 (Figure 77). This way, the DT800 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** DT800.

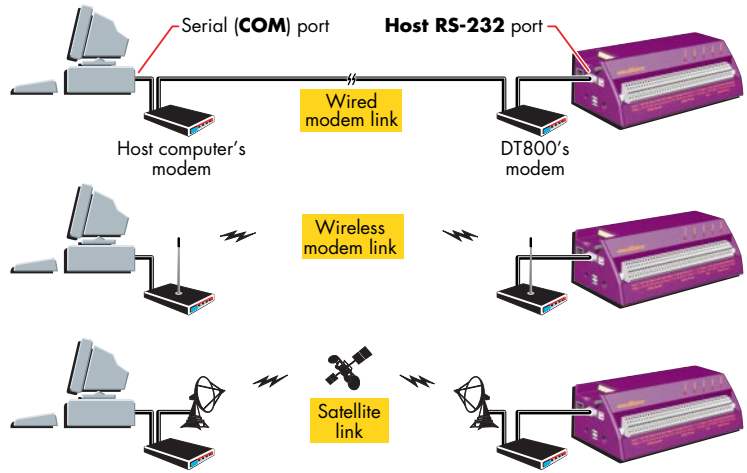


FIGURE 77 Modem (remote) connections

The information in "DT800 RS-232 Basics" (page 125) applies to a modem connection.

The DT800 can power the modem directly, or control the modem's power supply. If you use this facility, the DT800 can automatically reset the modem if it determines that this is necessary. See "Powering the DT800's Modem" on page 130.

DT800-to-Modem Cable

For the DT800 to recognise that it's connected to a modem and operates accordingly, the DT800 must see the signal at its DSR terminal as active (discussed in "Automatic Device Detection" on page 125). There are two ways to ensure this:

- Connect the DT800's Host RS-232 port (a DTE device) to a modem (a DCE device) using a straight-through (full-parallel) comms cable as shown in Figure 140 (page 192). When you do this, because the DSR line is connected-through in these cables and because most modems drive DSR active when they are on, the DT800 automatically assumes that it's connected by modem to the host computer and operates accordingly. This is the preferred method.
- If you're using a modem that does not drive its DSR line active when turned on, we recommend that you hardwire DSR to DTR at the DT800 end of the modem cable. This simulates an active DSR terminal, convincing the DT800 that it's connected to a modem.

Modem Initialization

The DT800 monitors its modem and manages it automatically, as follows:

1. When the DT800 detects that a certain initialization condition exists (see "Modem Initialization Conditions" below)…
2. the DT800 resets the modem by turning its power off and then on again (you must use the **EXT_POWER_SWITCH** profile key to specify to the DT800 how the modem power is controlled — see "Automatic Modem Power-Down Reset" on page 132)…
3. then sends initialising commands to the modem (see "Modem Initialization Settings" below).

Modem Initialization Conditions

The DT800 automatically attempts to initialise the device attached to its Host RS-232 port whenever it detects any of the following conditions:

- The state of the DT800's DSR terminal changes from inactive to active.
  When this occurs, the DT800 assumes that a modem has just been connected and therefore needs to be initialised.
- DSR is active at the time the DT800 powers up or is reset.
  When this occurs, the DT800 assumes that the attached modem may not have been powered up or reset at the same time as the DT800 and therefore needs to be initialised.
- DSR is active but CD has been inactive for a specified period of time.
  When this occurs, the DT800 assumes that the attached modem may be in an error state or locked-up, and therefore needs to be initialised. (Even if the modem is not in this state, the initialization does no harm.)
  You set this period using the **MAX_CD_IDLE** profile key — see page 114 (the DT800's default is 12 hours).

The DT800 initialises its modem by automatically sending the commands and settings specified in the HOST_MODEM section of the DT800 PROFILE Details table (page 114) to the modem when any of the above situations occur.

Modem Initialization Settings

The DT800's modem must be initialised with the following settings:

- Auto-answer incoming calls after a specified number of rings (default is 4 rings).
  The DT800 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. The modem command **ATS0=4** instructs the modem to automatically answer after four rings.
- Don't echo commands.
  Echo of commands is not required and only serves to confuse the DT800 if it attempts to interpret the command echo as a command. The modem command **ATE0** turns echo off.
- Don't report results of commands.
  Results codes and strings only serve to confuse the DT800 as it does not use these codes and may accidentally interpret them as commands. The modem command **ATQ1** enables quiet mode which stops result codes from being returned.
- The modem should keep DSR active at all times.
  This behaviour is ensured by sending the modem command **AT&S0**.
- If DTR goes inactive, the modem must terminate its call (if one has been made).
  This functionality is used by the DT800 to hang-up its modem. This behaviour is ensured by sending the modem command **AT&D2**.
- The modem must make CD active whenever it establishes a connection with a remote modem, and make CD inactive when the connection is broken.
  The DT800 interprets an active CD line to mean that it is connected to a remote host via a modem. This behaviour is ensured by sending the modem command **AT&C1**.

All of the above settings are included in the **INIT** profile key (see the HOST_MODEM section of the DT800 PROFILE Details table on page 114) — the default value of the **INIT** profile key is **ATE0Q1&D2S0=4&C1&S0**.

The DT800 issues these commands and their settings to its modem whenever it determines that the modem should be initialised (see "Modem Initialization Conditions" above).

AT Command Set

The DT800 supervises its modem using standard AT commands and common modem default settings where possible, and therefore supports the majority of modems.

Modem Automatic Baud Rate Selection

Modems that are compatible with the AT command set — that is, most modems — automatically set their local baud rate to match that of their host when they receive the first AT command. So when the DT800 sends the initialization string to the modem, it automatically sets the modem's local baud rate to that of the DT800 (in addition to initializing the modem).

In other words, the modem's local baud rate does not have to be set manually.

Modem Communications Protocol

To ensure reliable communication between modems, we recommend that you use some type of communications protocol between them. Moreover, for complete end-to-end integrity, the protocol should ideally exist between the DT800 and the host computer.

The DT800 supports PPP (Point-to-Point Protocol) for this purpose — see the PPP section of the **PROFILE...** commands (page 113).

If this is not possible or desirable, we recommend that you use an error-correcting protocol[14] between the modems, along with local[15] flow control. To enable an error-correcting protocol ("error control") in DeTransfer or DeLogger, ensure that this option is enabled in the host computer's modem settings. You do this by means of Windows' **Modems** control panel, in the modem's Advanced Connection Settings dialog box. See Figure 78.

Powering the DT800's Modem

When using a modem connection between the DT800 and its host computer, you can

- power the DT800's modem directly from the DT800's Serial Channel
- use the DT800's Serial Channel to control an external power supply to the DT800's modem
- use one of the DT800's digital output channels to control an external power supply to the DT800's modem.

These options are shown in Figure 79 and described below.

Powering/Controlling the Modem from the DT800's Serial Channel

Modems can be powered from the DT800's Serial Channel supply — the DT800's **12V** and **Gd** terminals. (Beware: the supply here is not necessarily 12 volts — see "Serial Sensor Power Supply" on page 165.) If you power the modem this way, you can then program the DT800 to switch its **12V** terminal on/off to reduce modem power consumption and to power-down reset the modem if it locks up.

You can also power modems from suitable supplies other than the DT800 (for example, 9Vac is commonly used). If you power the modem from one of these external supplies, you can similarly use the DT800 to switch the supply on/off to reduce power consumption and to power-down reset the modem if it locks up. To do this, wire the coil of a 12Vdc relay between

[14] Error-correcting protocols: LAPM or MNP2-4, for example.
[15] That is, between the computer and its modem, and between the DT800 and its modem.

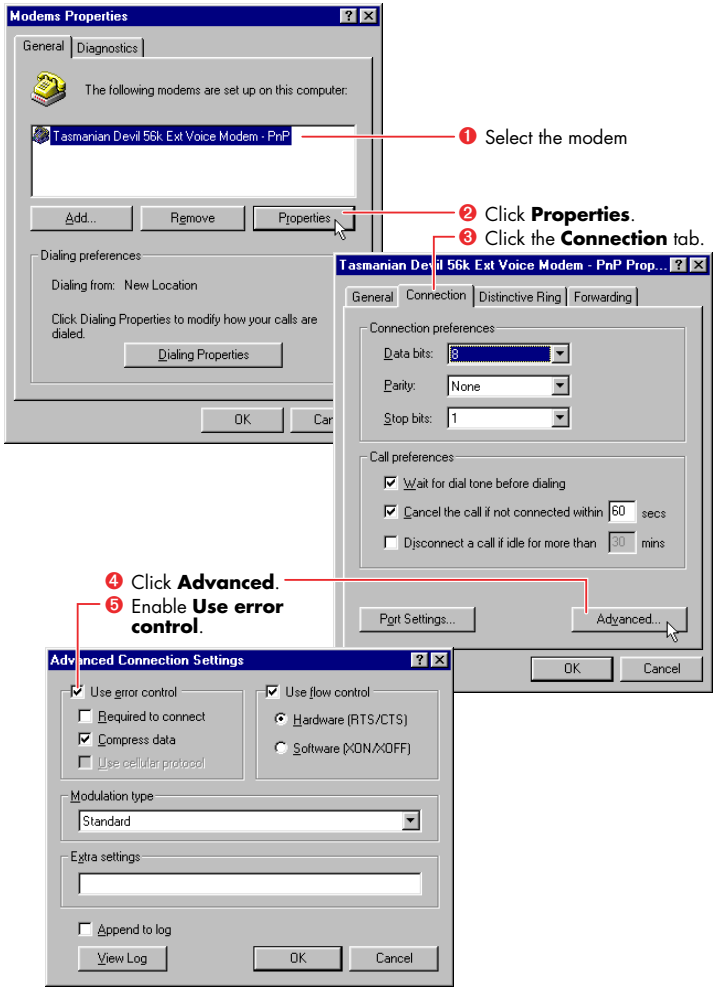


FIGURE 78 Enabling error control (Windows 98 example)

the Serial Channel **12V** and **Gd** terminals, and use the relay contacts to switch the modem's power supply.

For either situation, the command

`1SSPWR=1` powers the modem up, and `1SSPWR=0` powers the modem down.

(See "Serial Sensor Power Supply" on page 165.) These commands must be issued explicitly by the DT800 program to power the modem as required. For example, in remote sites it's often important to minimize power consumption, and one approach is to power the modem for only a short time each day to allow dialling in. You can do this with a schedule such as

```
RA1H
 IF(T>10:00){[1SSPWR=1]}
 IF(T>11:00){[1SSPWR=0]}
```

which powers the modem for one hour (between 10:00 and 11:00) each day.

Controlling Modem Power from a DT800 Digital Output

Alternatively, you can wire the coil of a relay to one of the DT800's digital output channels (**D1** to **D8**) and use the relay contacts to switch a modem's power supply (the state of the digital output controls the modem power). See "Digital State Outputs" on page 154.

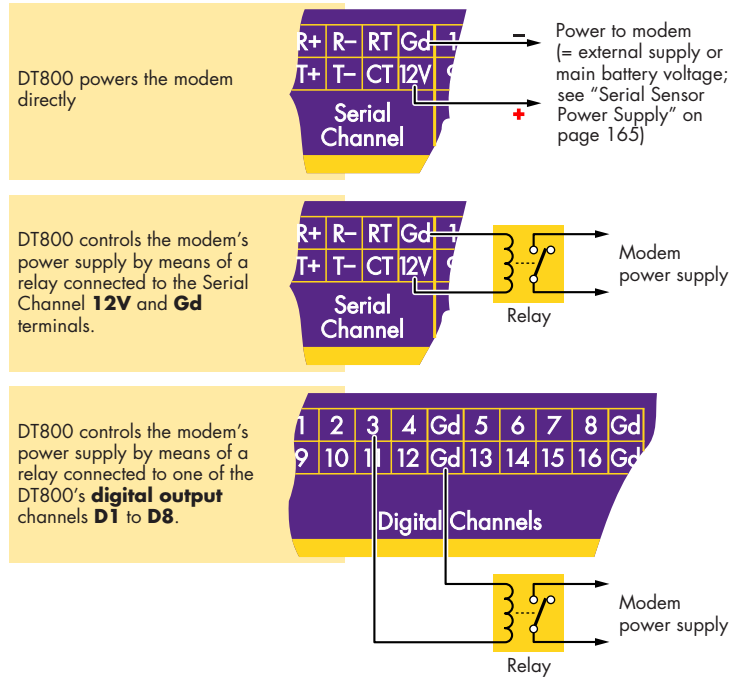


FIGURE 79 The DT800 can power a modem directly, or control the modem's supply

Automatic Modem Power-Down Reset

None of the above modem power arrangements provide for automatic power-down resetting of the DT800's modem if the modem ceases to respond.

To enable this DT800 feature:

1. Send one of the following **PROFILE...** commands (see the HOST_MODEM section of the table on page 114) to the DT800:
   - If the modem is powered from the DT800's **12V** and **Gd** terminals, send the command
     ```
     PROFILE"HOST_MODEM","EXT_POWER_SWITCH"="-1"
     ```
   - If the modem is powered from one of the DT800's digital output channels ***n*** (where ***n*** = **1** to **8**), send the command
     ```
     PROFILE"HOST_MODEM","EXT_POWER_SWITCH"="n"
     ```
   - If the modem is not powered by either of the above, send the command
     ```
     PROFILE"HOST_MODEM","EXT_POWER_SWITCH"="0"
     ```
2. Carry out a firm reset of the DT800 (see "Resetting the DT800" on page 118) to load the **PROFILE...** command.

From then on, the DT800 automatically power-down-resets the modem if it detects it to be unresponsive — see "Modem Initialization" on page 130.

Modem Communications Operation

Dialling In

Because the modem is initialised to automatically answer incoming calls, the DT800 does not have to monitor the RI signal at its Host RS-232 port or request the modem to answer the call. But the DT800 does have to monitor the CD signal to determine when a call has been established.

The DT800 does not receive commands or transmit data and status information unless it determines that a call has been established between itself and a host. When a modem is attached (DSR active), the DT800 monitors the CD signal to determine when it can transmit data and status information, and receive commands:

- When CD is active the DT800 accepts commands, return data and status information.
- When CD is inactive the DT800 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 DT800 does not send characters to the modem that the modem may interpret as commands to switch into a different operating state.

Dialling Out

The **SETDIALOUTNUMBER** command is used to set the number to dial, and the **DIAL** and **HANGUP** commands are used to initiate and terminate calls by the DT800's modem to the host computer's modem.

These are discussed in"Host RS-232 Port Commands" on page 126.

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 in the DT800 System Variables table on page 69.

Setting Up a Remote Connection

To set up a remote connection:

1. Configure the following DT800 **PROFILE...** parameters using a direct connection between a local computer (running DeTransfer, DeLogger or HyperTerminal) and the DT800 (Figure 80):
   a) Use **PROFILE"HOST_PORT","BAUD_RATE"...** if you want to set the DT800's baud rate to something other than its default, which is **57600** baud. (When the DT800 initializes its modem, the modem is automatically set to this baud rate — see "Modem Automatic Baud Rate Selection" on page 130.)
   b) Use **PROFILE"HOST_PORT","FLOW_CONTROL"...** to set the DT800's flow control. Hardware flow control is preferred.
   c) Use **PROFILE"HOST_MODEM","EXT_POWER_SWITCH"...** if you want the DT800 to power-down reset the modem (see "Automatic Modem Power-Down Reset" on page 132).
   d) Reset the DT800 so that these **PROFILE...** settings take effect. Use a firm reset (send the **SINGLEPUSH** command, or carry out a hardware reset or a power-up reset — see "Resetting the DT800" on page 118).

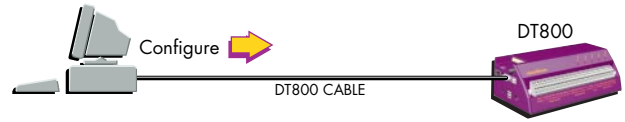


FIGURE 80 Use a direct connection to configure the DT800 for a remote connection

Here's an example of commands you might send:

```
PROFILE"HOST_PORT","BAUD_RATE"="115200"
PROFILE"HOST_PORT","FLOW_CONTROL"="HARDWARE"
PROFILE"HOST_MODEM","EXT_POWER_SWITCH"="-1"
SINGLEPUSH
```

2. Provide power to the modem and, if required, control its power as discussed in "Powering the DT800's Modem" on page 130.
3. Connect a suitable comms cable between the serial port on the DT800's modem and the DT800's Host RS-232 port.
   Suitable cables are discussed in "DT800-to-Modem Cable" on page 129.

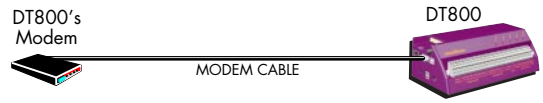


FIGURE 81 Connect the DT800 to its modem

4. At the remote/host end of the link, connect a suitable comms cable between the serial port on the host computer and its local modem.
   This cable is normally supplied with the modem.
5. On the host computer, launch suitable *dataTaker* or terminal software (for example, DeTransfer, DeLogger or HyperTerminal).
   Note Users of DeTransfer software: carry out the procedure in "Extend DeTransfer's Response Timeout" on page 133.
6. Configure the host software's communications parameters as required.
   We recommend hardware flow control and the use of an error-correcting protocol.

7 Wait a few minutes after connecting the DT800 to its modem (step 3 above), then attempt to connect to the remote DT800 from the host computer.
The few minutes' wait gives the DT800 time to automatically initialize its modem (usually occurs within a minute of detecting the modem's presence but, for certainty, we recommend that you wait longer).

Extend DeTransfer's Response Timeout

FIGURE 82 Step 5 of "Setting Up a Remote Connection" above

This step is for users of DeTransfer software. (HyperTerminal users may need to make a similar modification.)

When DeTransfer is connecting to a DT800 or communicating with it, there's a timeout within which the DT800 must respond to commands sent by DeTransfer. When DeTransfer is connecting to a remote DT800 through a modem link it's possible that this timeout will be exceeded. This can occur, for example, if the network is busy or has long latencies, or if there is line noise causing frequent retries. When this happens, the connection fails (times out) and it appears as if the modem or DT800 has hung.

So, if you're connecting to your DT800 by PSTN or GSM networks, we strongly recommend that you extend this DeTransfer timeout as follows:

1 From Windows' **Start** menu, choose **Run...**.

2 In the Run dialog box that opens, type **REGEDIT** in the Open field, then click **OK**.
REGEDIT starts and displays a folder list.

3 Navigate through **HKEY_CURRENT_USER** > **Software** > **dataTaker** and click the **DeTransfer** folder to show its contents in the Registry Editor's right-hand window.

4 Double-click **Extended Delay** and, in the Edit... dialog box that opens, change the value from 0 to the number of milliseconds delay that you want. For example, type **2000** here to extend the timeout by 2 seconds.

5 Click **OK** to close the dialog box.

6 Quit (exit) REGEDIT.
Your changes are automatically saved.

7 Close DeTransfer if it's running, then restart DeTransfer to load the new setting.

Installing the Host Computer's Modem

If you're running DeTransfer or DeLogger Pro on the computer, install your local modem into Windows using the **Modems** control panel. Then you can access the modem settings from within DeTransfer's or DeLogger Pro's connection properties.

See Figure 75 (page 127) and the appropriate *dataTaker* software manual for instructions on making this connection.

Using the Modem Connection

Once initialization is complete, you can dial into the remote DT800/modem site from the host computer and its local modem to program the DT800, recover data, change program variables, and so on.

Note that you can program the DT800 and configure it during the same connection session.

Visits to Site

If you visit the site where the DT800 and its modem are installed, you can communicate directly with the DT800 from a PC/Notebook by unplugging the cable from the modem to the DT800 at the DT800 end and then plugging in a direct cable, as supplied with your DT800, from your PC/Notebook to the DT800.

Differences in the cable wiring allow the DT800 to determine the type of connection and to respond appropriately.

DT800 Ethernet Communications

By means of the RJ-45 Ethernet port on the DT800's side, you can communicate with the DT800 to send commands to it and retrieve data from it over a 10BaseT Ethernet network.

Ethernet Protocol

TCP/IP is the DT800's default Ethernet protocol. See "TCP/IP" on page 210.

Configuring a DT800 for Ethernet

To use a DT800 on a TCP/IP network, you must configure the DT800 with three numbers:

- an **IP address** number
- an **IP subnet mask** number
- an **IP gateway** number.

There is only one way to do this. You use **PROFILE** commands to include these settings in the DT800's USER.INI file, which the DT800 automatically reads and applies after a firm reset (but not after a hard reset — see "Resetting the DT800" on page 118). This is covered later in this section.

Ethernet LEDs

The two LEDs on the DT800's Ethernet port (Figure 12 on page 25) indicate the following:

Green LED	Ethernet transmit activity from the DT800
Amber LED	Ethernet network activity

Ethernet Concepts

IP Address

Every device — a computer or a DT800, for example — on an Ethernet network using one of the TCP/IP protocols must have its own unique address, called its **IP address**. No two devices in the same network can have the same IP address — a device's IP address uniquely identifies it to all other devices on the network.

The IP address you assign to the DT800 is constructed from

- your network's **network number,** and
- an available (unused) **node number** (see later) for your DT800.

The topic "DT800 Ethernet Setup" on page 136 explains how to construct a suitable (that is, valid and unused) IP address.

Important Do not connect your DT800 to the network until you've configured the DT800 with a suitable IP address. Connecting a device with an invalid or used IP address may generate incorrect address information in other devices on the network. (The DT800 is pre-set with a "safe" IP address to avoid this happening in the unlikely event that you accidentally connect a powered DT800 to the network without firstly configuring it with a valid, unused IP address.)

An IP address has the general form **n.n.n.n**, where each **n** is a one, two or three-digit number between 0 and 255 (**169.254.97.8**, for example).

Assigning an IP Address

In general, a device can be assigned an IP address in two ways:

- **Dynamic IP Address**
  The network automatically assigns an IP address to the device whenever it logs on to the network. The address is temporary and may be different each time the device logs on.
- **Static IP Address**
  The user assigns an available IP address to the device. The address is stored in the device and reused every time it logs on to the network.

The DT800 has a Static IP Address— that is, you must assign an IP address to the *dataTaker*.

You do this by means of a **PROFILE...** command, which stores the address in the DT800's initialization file USER.INI. The command has the form

```
PROFILE"ETHERNET","IP_ADDRESS"="w.x.y.z"
```

For example:

```
PROFILE"ETHERNET","IP_ADDRESS"="192.9.200.15"
```

Two-Part Construction

An IP address consists of two parts:

- A **network number** — the part of the IP address that is the same for a group of computers and devices on the same network. The network number identifies the network.
- A **node number** — the part of the IP address that uniquely identifies the node (the network device; a computer or a DT800, for example). No two devices on the network should have the same node number. Often called a **host number**.

There are five **classes** of IP addresses, and the two parts of an IP address vary according to the class of the IP address.

The Bottom Line

Take great care when obtaining a correct, valid, unused IP address for assigning to your DT800. The topic "DT800 Ethernet Setup" on page 136 contains step-by-step instructions for doing this.

IP Subnet Mask, IP Gateway

Small networks are usually a single entity — just one network of nodes (devices) connected together by an Ethernet cable. In these cases, individual nodes on the network only require an IP address.

But larger networks may comprise a number of smaller networks, called **subnets**, connected together. These require an **IP subnet mask** (IPSN), another 32-bit number that determines how each IP address is subdivided into network portion and node portion. For example, network portions (underlined) can be 192.9.200.15 or 192.9.200.15 or 192.9.200.15. In other words, the subnet mask specifies which part or parts of every IP address are to be read as the larger network's address and which parts (the remaining parts) are to be read as the node's address.

Large networks may also be connected together through **gateways** (computers that connect multiple networks together, translating and routing one network's format, protocol or application information for use on another network). If a node on one network needs to communicate with a node on another network through a gateway, each node also requires an **IP Gateway** address. The DT800 is shipped with its IP gateway set to **0.0.0.0** — that is, no gateways.

In summary, all devices on a subnet must have

- the same network number (that is, the portion of the IP address that represents the network number must be the same for all devices on the same network)
- unique node numbers (the node number is what distinguishes one device from another)
- the same subnet mask (determines how the IP address is subdivided into network portion and node portion).

External or Local Networks

- **External Network** If you want your networked DT800 to communicate with other networks or subnets (that is, across one or more gateways), you must specify the **IP Subnet Mask** for the DT800's subnet, and possibly an **IP Gateway** to access other networks.
  Do this by sending the appropriate **PROFILE...** command to the DT800:
  `PROFILE"ETHERNET","SUBNET_MASK"="s.t.u.v"`
  and/or
  `PROFILE"ETHERNET","GATEWAY"="o.p.q.r"`
  See "Ethernet Commands" below.
- **Local Network** If you only want your networked DT800 to communicate locally (that is, only within the network or subnet to which it is connected), then you don't need to specify an **IP Subnet Mask**. The DT800 automatically applies a default IP subnet mask (`255.255.255.0`) that indicates "no subnets, no gateways".

Ethernet Protocols

For Ethernet network communications, the DT800 supports both the simplified UDP protocol and the more-advanced TCP/IP protocol.

TCP/IP is the DT800's default. To select UDP, send the command
`PROFILE"ETHERNET","UDP_SUPPORTED"="YES"`
To return to TCP/IP, send
`PROFILE"ETHERNET","UDP_SUPPORTED"="NO"`
to the DT800.

Ethernet Settings are Preserved

The Ethernet settings you specify using **PROFILE...** commands (IP address, IP subnet mask, IP gateway and protocol) are preserved through resets and full power-downs because they're stored in the DT800's USER.INI file, which the DT800 maintains in its lithium battery-backed SRAM memory.

IP Port Number

The DT800 uses IP port number **8** for TCP/IP protocol communications, and IP port number **7** for UDP protocol communications.

Therefore, when setting up an Ethernet software connection between the DT800 and, say, DeTransfer or DeLogger, be sure to enter the correct IP port number for the selected protocol in the software's port number field — unless a gateway you're using requires a different port number. This is restated in "DT800 Ethernet Setup" beginning on page 136.

Network Adapter Address

The DT800's **network adapter address** is the physical hardware address of the DT800's Ethernet port. This address is unique for each DT800 and cannot be changed. If you need a particular DT800's adapter address, you can see it in line 9 of the **CHARAC** report (see "CHARAC Commands" on page 123).

Don't confuse the DT800's network adapter address with its IP address — they are not the same.

Ethernet Always

The DT800 never sleeps when it's connected to an Ethernet network. And, no matter which mode the DT800 is in or which comms port is in use, you can always communicate with it through its Ethernet port.

Ethernet Commands

The DT800 supports the following Ethernet-related commands:

`PROFILE"ETHERNET","IP_ADDRESS"="w.x.y.z"`	Sets the DT800's IP address to *w.x.y.z* in the DT800's initialization file	Use when adding the DT800 to an Ethernet network. For example: `PROFILE"ETHERNET","IP_ADDRESS"="192.168.2.3"`	**Note** For **PROFILE...** commands to take effect, the DT800 must be restarted using any of the firm resets — see "Resetting the DT800" on page 118 and "User Startup Profile" on page 113.
`PROFILE"ETHERNET","SUBNET_MASK"="s.t.u.v"`	Sets the DT800's IP subnet mask to *s.t.u.v* in the DT800's initialization file	Only required if the network uses subnets. The DT800 is shipped with a default IP subnet mask of `255.255.255.0`	
`PROFILE"ETHERNET","GATEWAY"="o.p.q.r"`	Sets the IP gateway to *o.p.q.r* in the DT800's initialization file	Only required if the DT800 is to communicate with other networks by means of an Ethernet gateway. The DT800 is shipped with its IP gateway set to `0.0.0.0` — that is, no gateways.	
`IP`	Returns the DT800's current IP address		
`IPSN`	Returns the DT800's current IP subnet mask		
`IPGW`	Returns the DT800's current IP gateway		
`EAA`	Returns the DT800's Ethernet network adapter address		Set at factory (read-only)
`PROFILE"ETHERNET","UDP_SUPPORTED"= "YES"`	Enables UDP, disables TCP/IP		
`PROFILE"ETHERNET","UDP_SUPPORTED"= "NO"`	Disables UDP, enables TCP/IP		

DT800 Ethernet Setup

To configure your DT800 for use on an Ethernet network — even for a simple "network" of one computer and one DT800 using Ethernet — carry out the following steps in the order they're presented.

1. Obtain a valid, unused IP address for use on the Ethernet network.

There are two ways you can do this:

- Ask your network administrator.
  If this is possible, do it and write the number down, then jump straight to step 2.
- If this is not possible, continue below.

a) Switch **on** every device on the network to which you're going to connect the DT800.
This is vital to ensure automatic detection of every existing static IP address on the network (occurs later in this procedure).

b) Using a Windows computer that has TCP/IP installed, configured and operating correctly on the network, do the following:

i. From the **Start** menu, choose **Run**.
The Run dialog box opens.

ii. In the Open field, type **WINIPCFG** (see footnote [16]) then click **OK**.
The IP Configuration dialog box opens displaying the computer's current TCP/IP settings.

iii. Write down the four-part number shown in the **IP Address** field (for example, **192.168.30.3**).
The first three parts of this number (**192.168.30** in the example) constitute the network number you'll need for the DT800's IP address.

iv. If you want the DT800 to communicate beyond the local network to which you're about to connect it, also write down the numbers shown in the **Subnet Mask** and **Default Gateway** fields of the IP Configuration dialog box.
You'll use these later.

v. From the **Start** menu, choose **Programs** > **MS-DOS Prompt**.
An MS-DOS window opens. Here you'll use the **PING** command to locate a vacant IP address on the network using the network number found in substep iii above (**192.168.30**) followed by a fourth number between 1 and 254 (the node number) — we recommend you start with **1** as the fourth number.

vi. At the flashing DOS prompt, type **PING**, then a space, then the network number found in iii above, then a period (**.**), then the number **1** (**PING 192.168.30.1**, for example), then press the **Enter** key on your keyboard.
If a "Destination host unreachable" message appears, the IP address **192.168.30.1** is not in use on the network and you may use it for the DT800.
If a "Reply from…" message appears, the address is currently in use on the network and you can't use it for the DT800. In this case, type the PING command again at the DOS prompt but with a different last digit (**PING 192.168.30.2**, for example) and press **Enter**. Repeat this procedure (changing the last digit each time) until the "Destination host unreachable" message appears to indicate that you've found an unused IP address.

vii. When you've identified an unused IP address, write it down and go to step 2.

2. Communicate with the DT800 using RS-232.

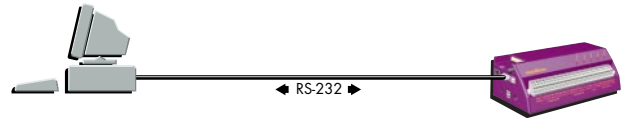


FIGURE 83 Ethernet setup — RS-232 communication first

a) Using the DT800's Host RS-232 port (not Ethernet), connect the DT800 to a computer and establish Host RS-232 communication using, say, DeTransfer software.
Do not connect the DT800 to the Ethernet network yet.

3. Add the IP address to the DT800's initialization file.

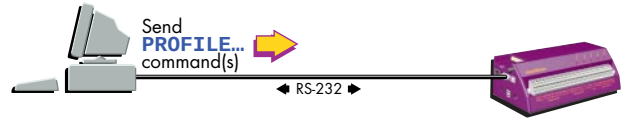


FIGURE 84 Ethernet setup — use RS-232 comms to add Ethernet settings to your DT800's user startup defaults

a) From DeTransfer, send the command
PROFILE"ETHERNET","IP_ADDRESS"="*w.x.y.z*"
where ***w.x.y.z*** is the unused IP address you wrote down in substep 1-b-vii above.
The DT800's initialization file USER.INI now contains the IP address.

b) If you want the DT800 to communicate beyond its local network (that is, with external networks or subnets that are connected to your local network by gateways), go to step 4. But if you only want the DT800 to communicate on the local Ethernet network, jump to step 5.

4. Add your network's IP subnet mask and IP gateway to the DT800's initialization file — only required if the DT800 is to communicate beyond its local network.

a) From DeTransfer, send
PROFILE"ETHERNET","SUBNET_MASK"="*s.t.u.v*"
where ***s.t.u.v*** are the numbers you wrote down from the **Subnet Mask** field in substep 1-b-iv above.

b) From DeTransfer, send
PROFILE"ETHERNET","GATEWAY"="*o.p.q.r*"
where ***o.p.q.r*** are the numbers you wrote down from the **Default Gateway** field in substep 1-b-iv above.
The DT800's initialization file USER.INI now contains the local network's IP subnet mask and IP gateway.

[16] Use **WINIPCFG.EXE** in Windows 95 and 98, **WNTIPCFG.EXE** in Windows NT, and **IPCONFIG.EXE** in DOS.

5. **Reset the DT800.**

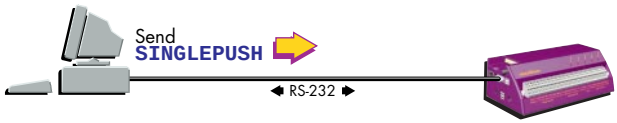


FIGURE 85 Ethernet setup — reset the DT800 to load the Ethernet PROFILE… setting(s)

a) From DeTransfer, send
`SINGLEPUSH`
or carry out a firm reset (see "Resetting the DT800" on page 118) to force the new settings in USER.INI to take effect. The DT800 is now ready for Ethernet operation.

6. **Connect the DT800 to the Ethernet network (or directly to a computer's Ethernet port) and test for correct operation.**

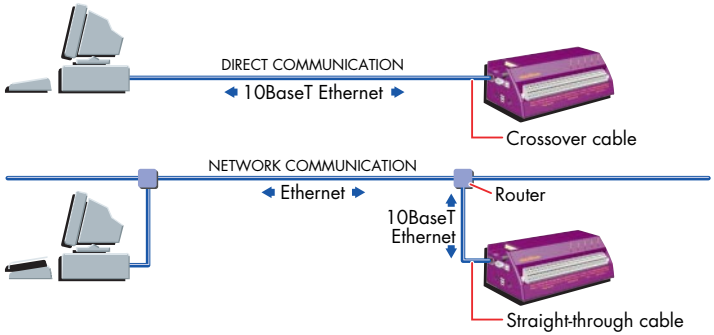


FIGURE 86 Ethernet connection types — direct connection and network connection

a) Using the correct cable — a standard 10BaseT cable (no crossover) — connect the DT800's Ethernet port to a socket on the Ethernet network (or directly to a computer's Ethernet port using a crossover cable).
b) Ensure that the DT800 is powered.
c) Using *dataTaker* host software — DeTransfer, DeLogger or DeLogger Pro — running on a computer also connected to the Ethernet network (or, for a direct connection, on the computer to which the DT800 is directly connected), create a software connection for communication with the DT800. The connection must use **TCP/IP** protocol, the DT800's IP address (from substep 1-b-iv) and port number **8** (see Figure 87).
d) Send the **TEST** command and check that the DT800 returns a test report (see page 120). If there's more than one DT800 on the network, verify that the report is from the correct DT800 by checking that the reported serial number is that of the DT800 you're testing.

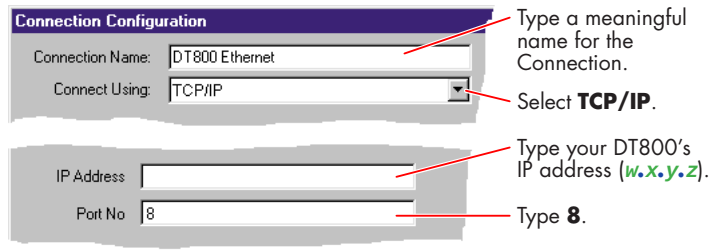


FIGURE 87 Example software connection for a DT800 on an Ethernet network (DeTransfer software shown)

DT800 FTP Communications

The DT800 can function as an FTP server. You can use this mechanism (and third-party FTP software) to transfer an ONRESET.DXC (startup job) file or other files to and from the DT800, and to upgrade the DT800's firmware over the internet. Contact your *dataTaker* representative for more information.

This functionality is enabled by default in the DT800, along with the FTP username **ANONYMOUS** and the FTP password **PASSWORD**. See the **FTP_SERVER** section of the DT800 PROFILE Details table (page 114).

DT800 USB Communications

Although the DT800 hardware supports USB, its firmware currently does not. However, USB firmware support will be added in the future and made available as a free firmware upgrade from **www.dataTaker.com**.

DT800 PPP Communications

Point-to-Point Protocol (PPP) allows TCP/IP-based protocols to be run over the Host RS-232 port of the DT800.

To initiate a PPP session with the DT800, clients must send the word **CLIENT** to the DT800 by means of its Host RS-232 port. The DT800 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 DT800, clients can simply close the PPP session from their end. Alternately, you can send the command **CLOSEDIRECTPPP** to the DT800 and it will close the PPP session.

Part J — Sensors and Channels

This part contains physical details (including wiring diagrams) of the sensors and channel types supported by the DT800.

Analog Channels

Analog Sensors and Measurement

4–20mA Current Loops

Wiring Diagrams: see "Current Inputs" on page 170

The DT800 supports current loop measurements. Unlike the DT500/600 series of *dataTakers*, the DT800 requires that you provide the external shunt resistor in the current loop — see Figure 105 (page 170).

The channel type for 4–20mA current loop measurement is

L(*shuntResistance*)

where

shuntResistance	is the channel factor; the value of the shunt resistor you're using in the loop (default is 100Ω)

The DT800 assumes a shunt resistance of 100Ω (the default). If you use a different shunt resistance, you must specify this using the **L** channel type's channel factor (see page 64) — for example, **3L(50.0)**.

Current-loop measurement is essentially the same as voltage measurement — the DT800 measures the voltage across the external shunt resistor and, knowing the shunt resistance, calculates the loop current.

Examples — Current Loop Measurement

The schedule

RA2S 4L

instructs the DT800 to measure, every 2 seconds (**2S**), the current in the 4–20mA loop sensed by analog channel **4** across a 100Ω shunt resistor (the default).

The schedule

RA2S 4L(50.0)

instructs the DT800 to measure, every 2 seconds (**2S**), the current in the 4–20mA loop sensed by analog channel **4** across a 50Ω shunt resistor.

Extending Common-Mode Range

Because the normal common-mode range of the DT800 is –10V to +10V, you should install the shunt resistance at the "ground end" of the current loop as shown in Figure 105.

But if this isn't practical (because of loop length/wiring considerations), you can make the DT800's common-mode range 0V to +20V by setting **P49=1** (see page 109). Then you can place the shunt resistance closer to the 24V side of the loop supply.

Note This requires the ground of the 24V loop supply to be connected to the DT800's **Ac** terminal.

Terminal boards containing shunt resistors are available as an option for your DT800 — contact your *dataTaker* representative.

AC Voltage (RMS)

Wiring Diagrams: see "Voltage Inputs" on page 168

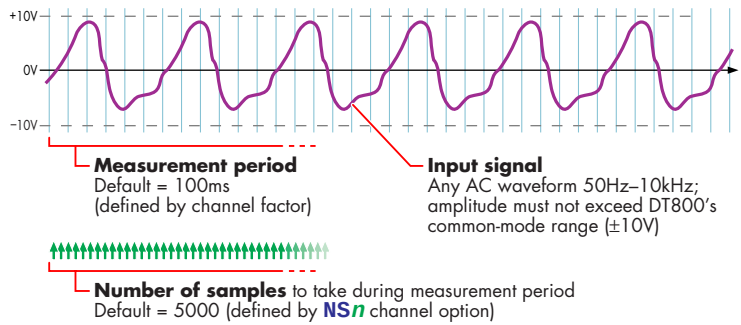


Figure 88 AC voltage measurement

The DT800 can return the RMS (root mean square) voltage of any AC waveform within the frequency range 50Hz–10kHz. The peak magnitude of the AC signal must not exceed the DT800's common-mode range[17]:

- For a sine wave centred at zero (that is, no DC component), this maximum is approximately 7Vrms.
- For non-sinusoidal waveforms, this maximum depends on the signal's **crest factor** (see page 205).

The DT800 makes an AC voltage measurement by taking a burst of fast samples then digitally processing the results using the following algorithm:

$$V_{ac} = \sqrt{\frac{\sum_{m=0}^{m=n} (v_{in_m} - \overline{v})^2}{n+1}}$$

where

m	is an integer $0 \le m \le n$
n	is the number of samples (default is 5000)
V_{in_m}	is the sampled value
$\overline{V}$	is the mean value of all samples

Like burst mode, AC voltage measurement uses all of the DT800's resources and so, during this measurement, the DT800 does nothing else.

AC voltage measurement can not be part of a burst schedule.

[17] The DT800's default common-mode range is –10V to +10V. You can change this to 0V to +20V by setting P49=1. See page 109.

The channel type for AC voltage measurement is

VAC(*period*,NS*n*)

where

period	is the channel factor; the measurement period during which the samples are taken (default is 100ms)
NS*n*	is the number of samples taken during the measurement period (default is 5000 samples)

It can be used with any of the DT800's analog input channels and returns readings in mVrms. See also the DT800 Channel Types table (page 64).

Measurement Period (Channel Factor)

VAC's channel factor channel option (see page 64) defines the measurement period in ms. The DT800's default is 100ms. If you need to change this, be sure to keep it long enough to include many cycles of the AC signal. (The measurement is not synchronized to the start or end of the signal cycle. Therefore, if fewer cycles are measured a greater error is introduced due to a partial cycle being included.)

AC voltage measurement uses the same autoranging and gain-locking facilities as DC voltage measurement. Autoranging uses the peak voltages obtained during a measurement period to determine the correct gain to use.

More Samples = More Accurate

The accuracy of AC voltage measurement is

- ±1.0% for –10 to 70°C
- ±1.5% for –45 to 80°C.

Because of the number of samples the DT800 takes and the calculation time necessary, AC voltage measurement can take a relatively long time. For example, taking 5000 samples in 100ms (the default settings) takes 850ms (100ms measurement period + 750ms calculation overhead) to return a single reading.

If you need to, you can reduce this time by reducing the number of samples that the DT800 takes within each measurement period — but with a corresponding reduction in accuracy. Be aware that not taking enough samples may result in inaccurate readings. (When using default settings, the accuracy of the DT800's AC voltage measurement is 1%.)

As you'd expect, taking more samples increases the accuracy of the reading (up to a certain point, after which accuracy gains are negligible).

Number of Samples Use the **NS*n*** channel option (see page 73) to define the number of samples to be taken during the measurement period (default = 5000). The DT800 spaces the samples evenly through the measurement period. If you specify more samples than the DT800 can fit into the measurement period, the DT800 automatically increases the period (by an appropriate multiple of the original measurement period).

Other Factors — Frequency and Waveform

Accuracy also depends on the following factors:

- **Signal Frequency** The defaults for AC voltage measurement are optimized for AC signals with a 50Hz or 60Hz fundamental frequency (along with their harmonics) by measuring over an integral number of mains cycles in 100ms. Therefore signals with low frequency components, or signals of a non-periodic nature, show greater measurement error (partial cycles have more effect). Also, signals with high frequency components (above 10kHz) may show measurement errors if the DT800 cannot take enough samples per cycle, and because of the absence of any built-in anti-aliasing filter.
- **Signal Waveform** Signals with a high crest factor are measured with lower accuracy because most of the samples are taken at a less-than-optimal gain.

Examples — AC Voltage Measurement

The schedule

RA10S 4VAC

instructs the DT800 to measure, every 10 seconds (**10S**), the RMS value of the signal on analog channel **4** using the default values for measurement period (100ms) and number of samples (5000).

The schedule

RA10S 4VAC(1000,NS2000)

instructs the DT800 to measure, every 10 seconds (**10S**), the RMS value of the signal on analog channel **4** by taking **2000** samples within a measurement period of **1000**ms.

Frequency

WIRING DIAGRAMS: see "Voltage Inputs" on page 168

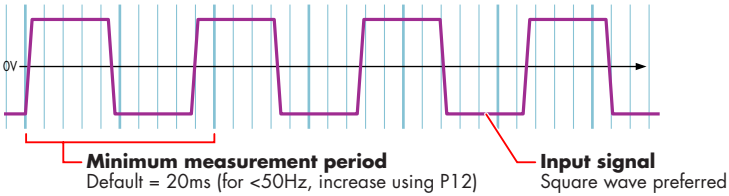


FIGURE 89 Frequency measurement

The DT800 can measure frequencies in the 50Hz to 10kHz range. You can extend this range down to 0.5Hz or up to 50kHz (see "Minimum Measurement Period (P12)" below), but these extreme measurements may be less accurate.

Although frequency measurement works with square, sine and sawtooth waveforms, square waves provide the best accuracy. For lower frequencies (below 1kHz), use a square wave whenever possible. You can amplify the signal (either externally, or using a DT800 channel option — see "Fixed Gain" below) so that it becomes clipped and therefore approaches the shape of a square wave.

Although it's not necessary for the waveform have a 50:50 mark:space ratio, its minimum pulse width must be 20µs.

Like burst mode, frequency measurement uses all of the DT800's resources and so, during this measurement, the DT800 does nothing else.

Frequency measurement can not be part of a burst schedule.

The channel type for frequency measurement is

`F(threshold,GLn)`

where

threshold	is the channel factor; the midpoint of the waveform (default is 0.0mV)
GL*n*	is the fixed gain setting used during frequency measurement (default is GL1V)

It can be used with any of the DT800's analog input channels and returns readings in Hz. See also the DT800 Channel Types table (page 64).

Threshold

F's channel factor channel option (see page 64) defines the threshold (in mV) against which the input signal is compared. Set this to the midpoint (that is, the average voltage) of the waveform being measured. Because signals commonly swing either side of zero, the DT800's default value for this channel option is 0.0mV.

Fixed Gain

The DT800 always uses a fixed gain for frequency measurements (that is, no autoranging). But you can select the gain using any of the **GL...** channel options (see "Gain" on page 72). The DT800's default is the 1V range (**GL1V**) because this provides a useful input for 5Vp-p signals regardless of the wave shape.

Warning When clipping a waveform, don't amplify the input signal excessively. As well as over-driving the input circuitry, this can cause a recovery period after each cycle that may result in poor measurements.

Minimum Measurement Period (P12)

The default minimum measurement period for frequency is 20ms, which is suitable for frequency measurement in the range 50Hz–10kHz.

To measure lower frequencies, make the measurement period larger by increasing the value of P12 (see page 107). At least one complete waveform cycle must occur within this period to measure a frequency. Therefore to measure down to 1Hz, for example, you need to set **`P12=1000`**.

Reducing P12 causes other processes and measurements to be held up for less time.

Accuracy

The accuracy of frequency measurement is

- ±0.02% for –10 to 70°C
- ±0.04% for –45 to 80°C.

Examples — Frequency Measurement

The schedule

`RA5S 4F`

instructs the DT800 to measure, every 5 seconds (**5S**), the frequency of the signal on analog channel **4** using the default values for threshold (0.0mV) and fixed gain (GL1V), and return the value in Hz.

The schedule

`RA5S 4F(500,GL2V)`

instructs the DT800 to measure, every 5 seconds (**5S**), the frequency of the signal on analog channel **4** with a threshold/midpoint of **500**mV and the DT800's gain fixed on the **2V** range, and return the value in Hz.

Thermocouples

WIRING DIAGRAMS: see "Voltage Inputs" on page 168

Thermocouple Theory

A thermocouple is 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 (see "Grounded Thermocouples" on page 143).

Using Thermocouples with the DT800

Thermocouples are wired to the DT800 as for any voltage signal — see Figure 100 and Figure 101 (page 168). The channel type is a **T*t*** where ***t*** is the thermocouple type (**TB**,**TC**,...**TT**).

Using the thermocouple channel type reads the channel as a voltage and automatically applies cold junction compensation and linearization.

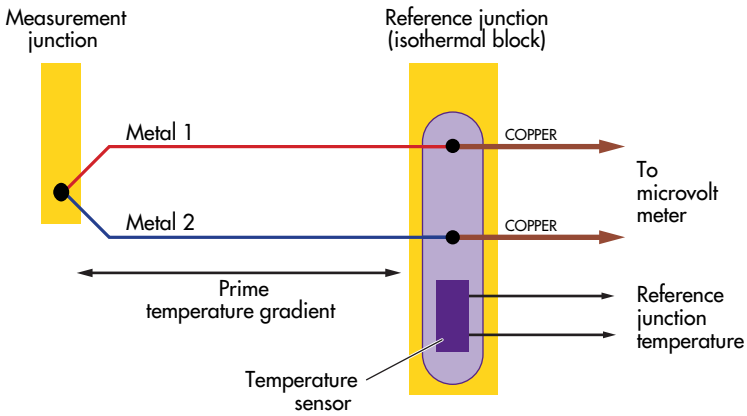


FIGURE 90 Thermocouple theory

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 DT800 makes this correction in software. The software approach allows support for any thermocouple type without hardware dependence.

Isothermal Block

Generally the reference junctions 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.

Reference Junction Support

By default, the DT800 uses the internal temperature sensor (see **REFT** channel type on page 67) as the reference junction sensor. The internal sensor has an accuracy of ±0.5.

ITS90

In 1990 the definition of the International Temperature Scale changed. The DT800 is calibrated to ITS90.

Thermocouple Types

The DT800 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%min, 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.

Grounded Thermocouples

Frequently, 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 DT800's analog common terminal **Ac**.

See also "Ground Loops" on page 23.

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 DT800
- the linearization accuracy of the DT800.

The accuracy of temperature measurement with thermocouples is a function of the accuracy of the thermocouple, and not of the DT800.

Reference Junction Error

The most significant source of error is the reference junction. The DT800 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 DT800's reference temperature sensor is positioned behind analog channels 6 and 7. Therefore, when precise temperature measurements are required, attach thermocouples here for the least temperature differential from the *dataTaker*'s reference temperature.

Linearization Error

The DT800's linearization errors are much lower (< 0.1°C over the full range) than other error sources.

Thermistors

WIRING DIAGRAMS: see "Resistance Inputs" beginning on page 171

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 DT800 has channel types for many 2-wire YSI[18] thermistors and, for other thermistor types, the DT800 supports thermistor scaling — see page 91.

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		–45
YS03	1000	44003A, 44101A		–20
		44035		
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	

[18] Yellow Springs Instruments — YSI Incorporated (www.ysi.com)

RTDs

WIRING DIAGRAMS: see "Resistance Inputs" beginning on page 171

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 DT800 supports four RTD types:

Metal	Alpha	Standard
Platinum (PT385)	α = 0.003850	DIN43760
Platinum (PT392)	α = 0.003916	JIS C1604
Nickel (Ni)	α = 0.005001	
Copper (Cu)	α = 0.00390	

The alpha is defined by

$$\alpha = \frac{R_{100} - R_0}{100 R_0} \quad \Omega/\Omega/°C$$

where

R_0	is the resistance at 0°C
R_{100}	is the resistance at 100°C

The RTD channel types (see **PT385**, **PT392**, **NI** and **CU** on page 65) are connected as for a resistance. 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 default connection is for a 3-wire measurement, but 4-wire can be specified as a channel option for greater accuracy. For example

```
PT385(4W,50.0)
```

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 DT800 supports the following commonly-available IC temperature sensors:

Sensor	Channel Type	Output	Wiring Diagrams
Semiconductor current source types (Analog Devices)	**AD590** **AD592** **TMP17**	1µA/K 1µA/K 1µA/K	Figures 126 and 128
Semiconductor voltage output types (National Semiconductor Corp.)	**LM135** **LM235** **LM335**	10mV/°C 10mV/°C 10mV/K	Figures 132 to 134
Semiconductor voltage output types (National Semiconductor Corp., Analog Devices)	**LM34** **LM35** **LM45** **LM50** **LM60** **TMP35** **TMP36** **TMP37**	10mV/°F 10mV/°C 10mV/°C + 500mV 10mV/°C + 500mV 6.25mV/°C + 424mV 10mV/°C 10mV/°C + 500mV 20mV/°C	Figures 129 to 131

For more details, see the DT800 Channel Types table (page 65).

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.

The DT800 provides a slope (or scale) correction capability on a per sensor basis using the channel factor — see page 65. Frequently, a slope correction based on a single point calibration point is enough for reasonable accuracy. The pivot point for the slope correction depends on the sensor type.

Sensor	Slope Pivot T_p	Channel Factor	Formula
AD590	0.0K (–273.15°C)	Series resistor R (Ω)	= R x C
LM335	0.0K (–273.15°C)	Attenuation factor A	= A x C
LM34	0°F (–17.78°C)	Calibration factor	= C
LM35	0°C	Calibration factor	= C

The calibration factor is calculated from

$$C = 1 - \frac{\Delta T}{T - T_p}$$

where

ΔT	is the temperature error	All temperatures must be in the same units.
T	is the temperature of the calibration	
T_p	is the pivot temperature	

Example — AD590

For the AD590 sensor, the channel factor represents the value of the series resistor used to measure the output current (default value is 100.0Ω). Without changing the actual resistor, this channel factor can be adjusted as follows.

If the temperature error is determined to be 1.7°C higher than actual at 100°C, the channel factor correction is

$$\text{Channel factor} = R\left(1 - \frac{\Delta T}{T - T_p}\right) = 100\left(1 - \frac{1.7}{100 - (-273.15)}\right) = 99.544$$

and is applied as follows:

```
5AD590(99.544)
```

Bridges

WIRING DIAGRAMS: see "Bridge Inputs" beginning on page 174

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.

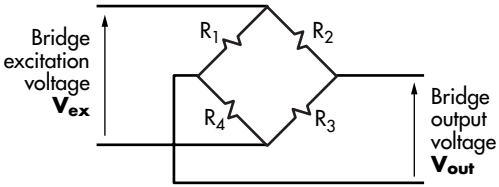


FIGURE 91 Wheatstone bridge

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 DT800 can resolve this problem:

Voltage Excitation BGV

The DT800 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 "6-Wire BGV Inputs" beginning on page 174). This is termed **voltage excitation**.

Constant-Current Excitation BGI

The alternative lead compensation method is to apply a constant-current (default is 10mA) to the bridge — assuming the bridge resistance is known and constant — and then calculate the excitation voltage Vex. See "4-Wire BGI Inputs" beginning on page 178.

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 DT800. 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 DT800.

Scaling

The DT800 scales all bridge channel types to a ratiometric form with units of parts per million (ppm):

$$\text{Reading } B_{out} = \left(\frac{V_{out}}{V_{ex}}\right)10^6 \text{ ppm}$$

where

V_{out}	is measured as a voltage
V_{ex}	is measured by a reference channel for voltage excitation, but calculated for constant current excitation

To convert to other engineering units, apply a polynomial, span or use calculations (see "Manipulating Data" beginning on page 85).

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 initial length
ΔL	is the length change
R	is the initial 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 DT800'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 DT800's bridge channel (BGV or BGI) result
G	is the gauge factor
N	is the number of active gauges in the bridge

The conversion can be done in the DT800 by applying a polynomial as a channel option (see page 91):

```
Y1=0,k"uStrain"     'Polynomial definition
3BGV(Y1)            'Bridge channel
```

where

$$k = \frac{4}{G \times N}$$

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 (as in "Example — Humidity Measurement" above), 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):

```
Y1=6.1,0.44,0.014,2.71E-4,2.73E-6,2.75E-8 'SVP polynomial
BEGIN
  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, a `1` is returned.
- When the input is below the threshold, a `0` is returned.

The default threshold is 2500mV, but can be set to any value in mV — see **AS** in the DT800 Channel Types table (page 66).

Example — Analog State Input

The channel list

```
1AS(1750)
```

configures analog channel **1** as an analog state input with a threshold of **1750**mV.

DT800 Analog Sub-System

Two important sub-sections of the DT800 are completely isolated:

- the DT800 analog section is electrically isolated from the rest of the DT800 (see Figure 92)
- the DT800 Ethernet interface is transformer-isolated from the outside world.

Isolated Analog Section

Because the analog section is electrically isolated from the rest of the DT800, sensor-to-equipment ground loops (see Figure 95 on page 151) are unlikely to arise.

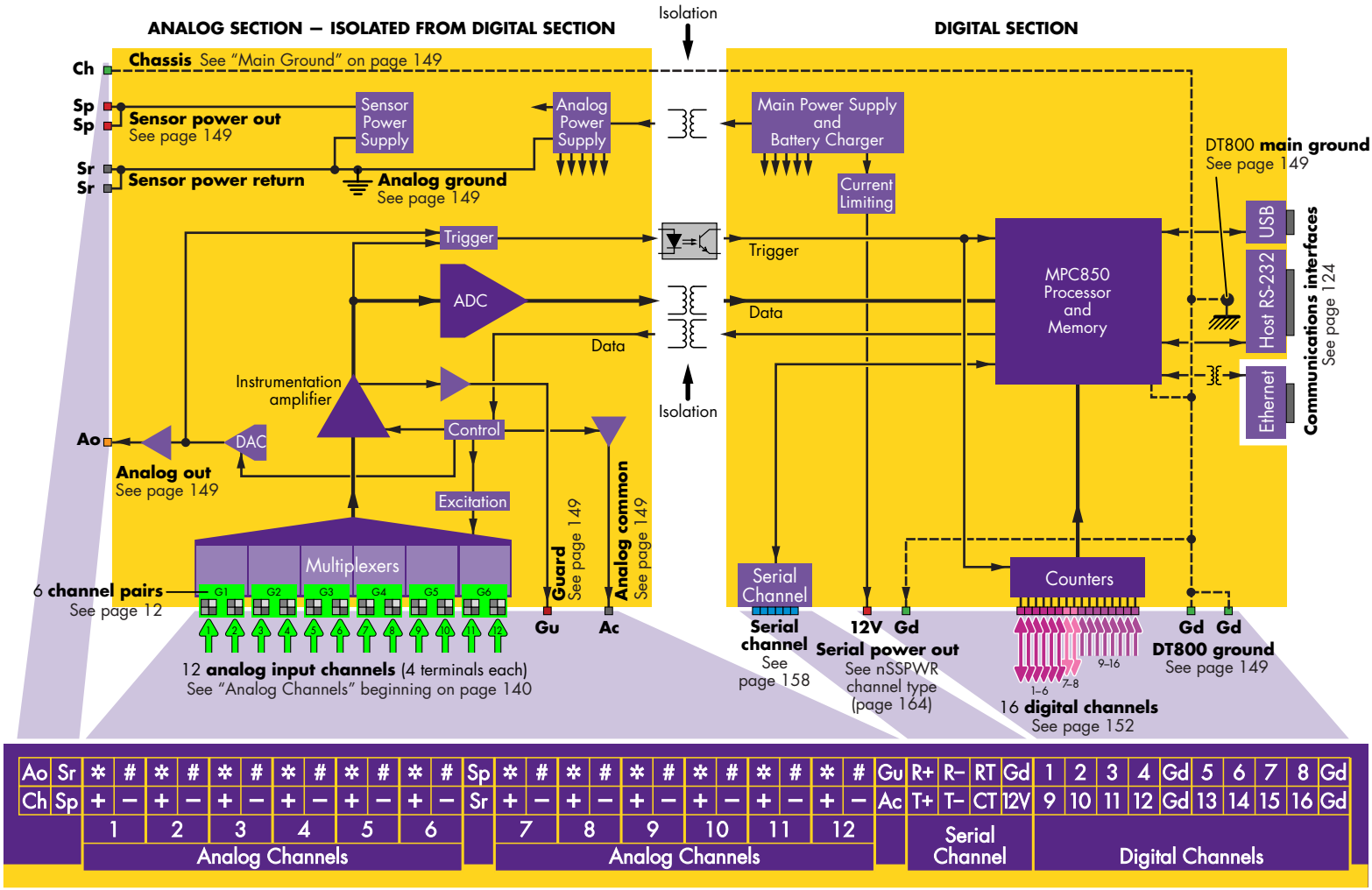


FIGURE 92 DT800 block diagram — see also Figure 11 on page 24

Wiring p168 Schedules p44 Channel Types p63 Channel Options p71 Alarms p96 Parameters p107 Switches p111 Error Messages p197

UM-0068-A2

Special Analog Terminals

The DT800 has a number of terminals associated with the analog sub-system that provide additional versatility in the measuring system. See Figure 92 on page 148.

Sensor Power Terminal

The sensor power terminal **Sp** can provide either 5V or 10V to power sensors. P47 selects the voltage:

P47=	Sensor Power	Max. Current
0	Off	
5	5V	100mA
10	10V	50mA

Sensor Return Terminal

The sensor return terminal **Sr** is the current return path for sensors powered from **Sp**. It is connected internally to the isolated analog electrical common.

Analog Common Terminal

The analog common terminal **Ac** is actively-driven so that, in normal operation, it is set to the electrical common (see "Sensor Return Terminal" below) of the DT800's isolated analog system. You can set the **Ac** terminal to –10V below electrical common using P49:

	Common-Mode Range		
P49=	Relative to Ac	Relative to Sr	
0	–10V to +10V	–10V to +10V	Default
1	0V to 20V	–10V to +10V	Note that this setting does not affect the DT800's maximum analog input voltage, which remains at ±13V.

Don't confuse the **Ac** terminal with the **Sr** (analog ground) terminals. They are not the same.

Wiring Configurations and the Ac Terminal

The **Ac** terminal should in general be connected to an external ground point in a way that ensures the voltage on any analog input terminal (**+**, **–**, ***** or **#**) is within the selected common-mode range. For example, in a 4–20mA loop system, this can be connected as shown in Figure 93.

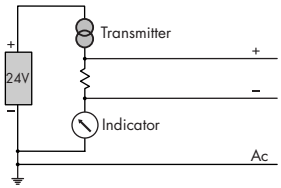


FIGURE 93 Using the **Ac** terminal in a 4–20mA loop measurement

In "Wiring Configurations — Analog Channels" beginning on page 168, the **Ac** link is generally not shown, but the need for it is indicated by the ⏚ symbol. This grounding link is shared by all channels.

The symbol indicates that the signal source must be at a potential of less than ±10V relative to the **Ac** terminal. In other words, the voltage applied to an analog input terminal should not exceed ±10V relative to **Ac** for accurate measurement.

Analog Output Terminal

The DT800 has an analog output labelled **Ao** (see Figure 11 on page 24) that provides a fixed voltage you can use to drive analog devices such as indicators, annunciators, or relays, contactors and other actuator-operated equipment.

It can provide up to ±10V (12-bit/10mV resolution) at 20mA (nominal output impedance <1Ω).

Guard Terminal

When the sensor has a high output impedance, and cable capacitance and insulation leakage are significant, we recommend using a **guard** — that is, an actively-driven shield around the sensor cables that is maintained at the common-mode voltage of the input signal.

The DT800 provides a terminal its the front panel for this purpose: the guard terminal, labelled **Gu** (see Figure 11 on page 24). The DT800 always provides a voltage at this terminal equal to the common-mode voltage of the analog input currently being sampled.

Foe examples of wiring configurations using a guard, see Figure 136 and Figure 137 (page 183).

DT800 Ground Terminals

The DT800 has two ground systems — **main ground** and **analog ground** (Figure 94):

Main Ground

As shown in Figure 92 (page 148), the DT800's true, "quiet", internal ground reference point — its "main ground" — is brought out to four terminals on the front panel. Three are labelled **Gd** (Ground) and the fourth is labelled **Ch** (Chassis) — see Figure 11 (page 24). The similarities and differences are as follows:

	Gd (Three Terminals)	**Ch** (One Terminal)
Connected internally to	DT800 main ground	DT800 case, which is connected internally to the DT800 main ground
For	Digital input and output returns, and **12V** terminal return (power for serial and other devices)	Connection of shields and lightning protection
Not for...	Not for use with analog inputs (≠ analog ground)	

Analog Ground

The floating analog section of the DT800 has its own ground, "analog ground", which is brought out to the two **Sr** (Sensor power return) terminals in the analog section of the front panel. This analog ground is totally isolated from the DT800's other ground (the **Gd** terminals and the **Ch** terminal).

See also "Sensor Return Terminal" above.

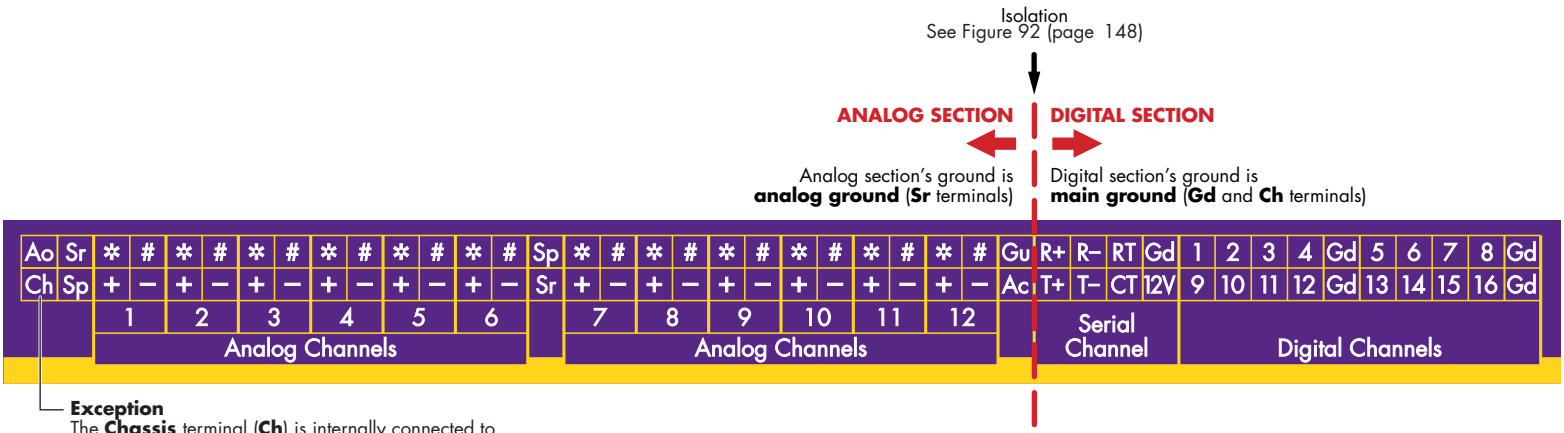


FIGURE 94 The DT800 has two ground systems

Grounds, Ground Loops and Isolation

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 95 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).

Avoiding Ground Loops

Generally, avoidance is better than cure, so break the ground loop.

There are several strategies for minimizing the impact of ground potentials:

- Use only one system ground point.
- Ground sensors at one end of the sensor cable only. This should nearly always be at the DT800 end.
- Wire the DT800 analog sub-system making full use of its internal isolation — see Figures 92 and 94.

Isolation

The traditional method of ground loop avoidance is to use an electrically-isolated measuring system, where each channel is electrically isolated from others. But this "ideal" approach is very expensive and slow if it relies on relays for multiplexing (channel selection). Furthermore, its accuracy and stability can be inferior.

The DT800 takes a different approach to avoid these disadvantages. It provides over 100V isolation of the analog sub-system from the rest of the DT800, but limits inter-channel isolation to 40V. This allows for fast multiplexing and comprehensive error detection and correction.

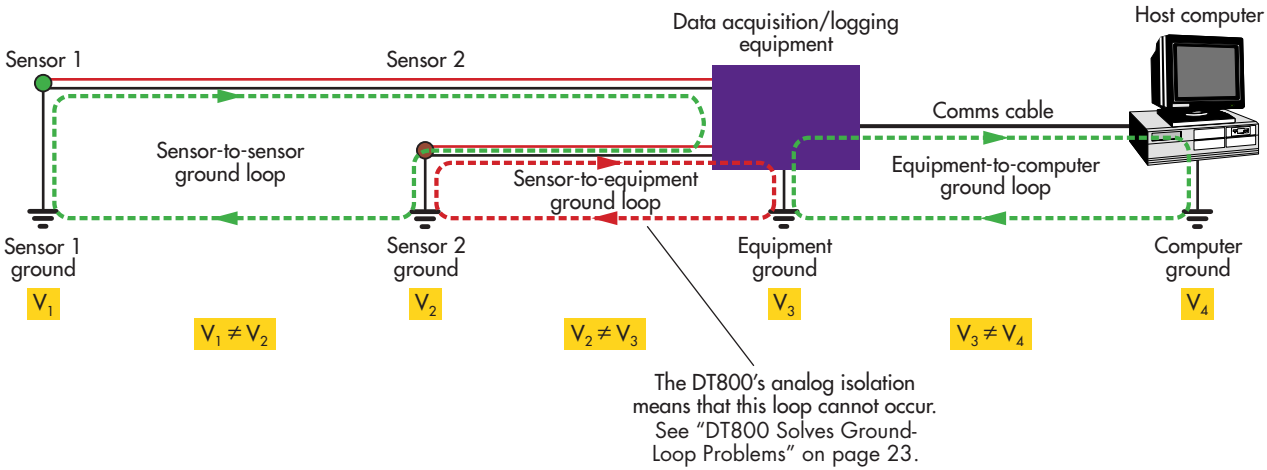


FIGURE 95 Some of the possible ground-loops in a measurement system

DIGITAL CHANNELS

See "Digital Channels — Introduction" on page 14 for a quick overview of the DT800's sixteen digital channels.

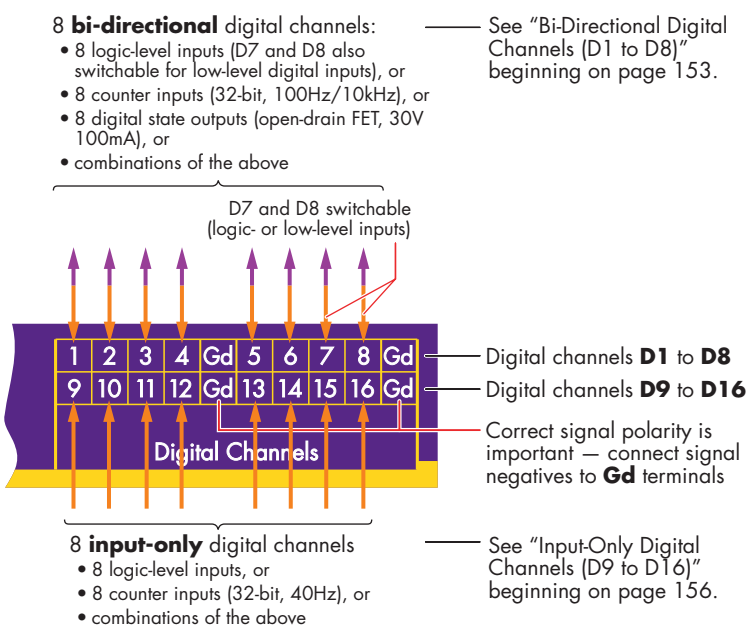


FIGURE 96 The DT800's digital channels

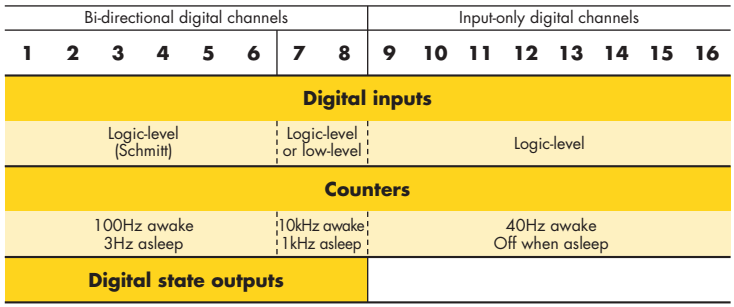


FIGURE 97 Anatomy of the DT800's digital channels

Digital Channel Specifications

D1*–D6	Maximum continuous voltage Minimum continuous voltage	30Vdc –0.6Vdc
	Note Voltages outside this range can permanently damage the channel.	
	Maximum input high voltage (Schmitt trigger input) Minimum input low voltage (Schmitt trigger input)	2.4V 0.6V
	Maximum continuous current that input can sink Maximum pulsed current that input can sink (300µS pulse every 15mS)	220mA 880mA
	Note Currents outside this range can permanently damage the channel.	
	Minimum hysteresis	200mV
	Low-pass filter – voltage-free contacts (for example, relay) Low-pass filter – active drive (for example, logic device)	200Hz 200kHz
	Maximum counter speed – awake Maximum counter speed – asleep	100Hz 3Hz
	Maximum output current (open-drain FET)	100ma
D7–D8	Maximum continuous voltage Minimum continuous voltage	30Vdc –0.6Vdc
	Note Voltages outside this range can permanently damage the channel.	
	Maximum input high voltage (comparator with hysteresis) – normal mode Minimum input low voltage (comparator with hysteresis) – normal mode	2.2V 1.1V
	Maximum input high voltage (comparator with hysteresis) – low threshold mode Minimum input low voltage (comparator with hysteresis) – low threshold mode	5mV 1.6mV
	Maximum continuous current that input can sink Maximum pulsed current that input can sink (300µS pulse every 15mS)	220mA 880mA
	Note Currents outside this range can permanently damage the channel.	
	Minimum hysteresis	500mV
	Low-pass filter – voltage-free contacts (for example, relay) Low-pass filter – active drive (for example, logic device)	200Hz 200kHz
D9–D16	Maximum continuous voltage Minimum continuous voltage	50Vdc –10Vdc
	Note Voltages outside this range can permanently damage the channel.	
	Maximum input high voltage (Schmitt trigger input) Minimum input low voltage (Schmitt trigger input)	2.4V 0.6V
	Maximum continuous current that input can sink Maximum pulsed current that input can sink (300µS pulse every 15mS)	220mA 880mA
	Note Currents outside this range can permanently damage the channel.	
	Minimum hysteresis	200mV
	Low-pass filter	200Hz

* In this manual, digital channel numbers are prefixed by the letter **D** to distinguish them from analog channel numbers, which are not given a prefix.

Warning The DT800's digital inputs are NOT reverse-polarity-protected. Therefore ensure signal polarity is correct — positive to numbered terminals, negative to **Gd** terminals — before connecting signals to the DT800's digital inputs.

Warning Do not apply more than 50Vdc to inputs D1–D8, and do not apply more than 30Vdc to inputs D9–D16.

Important Beware of conflicts when using the DT800's bi-directional digital channels (**D1** to **D8**). For example, if a device such as a PLC is actively driving one of these channels but you program the channel as an output (for example, `1DSO=0`), a conflict exists that has the potential to damage the digital channel or the driving source. We recommend the following:

- To avoid potential conflicts, use digital channels **D9** to **D16** for digital inputs (instead of **D1** to **D8**).
- If you must use **D1** to **D8**, consider 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.

Configuring Digital Channels

You instruct the DT800 to configure a particular channel for operation as an input, a counter, or an output by using the appropriate channel type in a schedule command — see the Digital category and the Counter category in the DT800 Channel Types table (page 66).

Wiring Digital Channels

See "Wiring Configuration — Digital Channels" on page 184.

Bi-Directional Digital Channels (D1 to D8)

Any of the DT800's eight bi-directional digital channels (channels D1 to D8 in Figure 96) can be used as either

- a digital input (for monitoring digital states), or
- a counter input, or
- a digital state output (for driving relays and other control devices).

Digital State Inputs D1 to D6

You can configure the DT800's digital channels D1 to D6 to operate as Schmitt-triggered logic-level digital state inputs that recognize input levels as follows:

1	above 2.4V
0	below 0.6V

Digital State Input Channel Types

The following channel types are provided for defining digital inputs:

DS	State input on a digital channel (1 bit/input)	See the Digital category in the DT800 Channel Types table (page 66).
DN	Digital nybble (4 bits/input)	
DB	Byte input on a pair of digital channels (8 bits/input)	
DW	Digital word (16 bits/input)	
AS	Digital state input on an analog channel — see "Analog Logic State Inputs" on page 147	

Triggering on Digital State Inputs

State changes on digital inputs can trigger DT800 schedules. See "Trigger on External Event" on page 46 and "Trigger While" on page 48.

Digital State Input Channel Options

The following channel options are provided for modifying digital inputs:

TRR	Time from rising edge to rising edge	See the Digital Manipulation category in the DT800 Channel Options table (page 74).
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	

Digital State Inputs D7 and D8 (Low-Level)

Digital state inputs D7 and D8 are switchable from logic-level operation to low-level (high-sensitivity) operation by the use of the **LT** channel option (see page 72):

Logic-Level Operation (without **LT** channel option)		**Low-Level Operation** (with **LT** channel option)	
1	above 2.2V	**1**	above 5.0mV
0	below 1.1V	**0**	below 1.6mV

They can then be used with sensors whose output is only a few millivolts (inductive-pickup flow sensors, for example). The **LT** channel option applied to digital channels D7 or D8 enables state recognition upon 10mV threshold crossing in both directions.

The **LT** channel option works for digital channels 7 and 8 when they are defined as either **DS** (digital state) or **C** (counter) inputs.

With the exception of the **LT** channel option, the channel types, triggering, channel options, and wiring details you use for D7 and D8 are the same as for "Digital State Inputs D1 to D6" above.

Example — Low-Level Input

The channel list

```
8C(LT,100)
```

instructs the DT800 to read digital channel **8** as a low-threshold (**LT**) counter, resetting every **100** counts (see "Counter Channel Options" on page 157).

Counter Inputs C1 to C8

See "Counters" on page 157.

Digital State Outputs

The DT800's eight bi-directional digital channels (channels D1 to D8 in Figure 96) can be used to output digital states for driving relays, solenoids and other control devices directly.

The outputs can be addressed as either digital bit outputs or digital byte outputs (see "Switching Digital Byte Outputs" on page 155), and are buffered.

While a digital state output of the DT800 is ON, it cannot be used to monitor digital inputs.

Because the digital state output channels (**DSO**) are bi-directional, you can determine their state at any time by reading them as logic bit inputs (**DS**).

Connecting to the Digital State Outputs

You can use the digital state output channels to interface to TTL or CMOS circuits directly and, because they can sink up to 100mA at 30Vdc (open-drain FET, with a 15kΩ pull-up resistor to 5V), they are suitable for driving low-voltage actuator devices directly. To do this, connect the actuator between an external power supply and a digital state output, then connect the negative side of the external power supply to one of the DT800's **Gd** terminals. The power supply can be an external power supply of 5–30Vdc, or the switched output at the DT800's **12V** terminal. See Figure 138 (page 184).

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.

The outputs are active when high — that is, they switch high (to +3.3V) when turned ON. They can drive up to two TTL loads, and are referenced to the DT800's **Gd** terminals.

Digital State Output Channel Types

The digital state outputs can be set

- individually to provide bit outputs, or
- in groups of channels to provide nybble or byte outputs.

The following channel types are provided for setting the digital outputs to required states:

DSO	Output on a single digital channel **1** = ON and high/active (3.3V), **0** = OFF and low (0.6V)	**Digital bit output**
DNO	Nybble output on a pair of digital channels	**Digital nybble output**
DBO	Byte output on a pair of digital channels	**Digital byte output** (see "Switching Digital Byte Outputs" below)

See the Digital category in the DT800 Channel Types table on page 66.

Setting a digital state output OFF results in the voltage at its output going to a low state of approximately 0.6V. For example, the commands

```
1DSO=1 3DSO=0
```

turn digital state output 1 ON (goes to a high state of approximately 3.3V) and digital state output 3 OFF (goes to a low state of approximately 0.6V).

Digital State Output Channel Options

To allow external conditions to settle before continuing to execute the DT800's program, an optional delay period can be invoked after switching the digital state outputs. During the delay period, only time-keeping, counting and servicing of the serial interfaces is maintained. The delay can be from 1 to 65535ms. See "Delay Period and Reset" on page 154.

In addition. you can specify an optional reset when switching digital state outputs. This resets the outputs to the opposite state to that selected, thereby producing a pulse output. If a delay period is specified, the output is reset after the delay. See "Delay Period and Reset" below.

The digital state output channels (**DSO**, **DNO** and **DB0**) are switched by commands in the general formats

```
nDSO(delay,R)=state
```

and

```
m..nDSO(delay,R)=state
```

where

n	is the channel number of a single digital state output channel	
m..n	is a sequence of digital state output channel numbers	
DSO	is the digital bit output channel type	
delay	is an optional delay period after switching	Specified as an integer within the range of 1 to 65535ms (a little more than one minute). If there is no delay period specified, the digital state output is switched and the DT800 continues without delay. See "Delay Period and Reset" below.
R	is an optional reset to opposite state after switching	If not specified, the digital output remains in the state to which it is switched. See "Delay Period and Reset" on page 154.
state	is the state to which the output is to be set	Can be specified as a constant, a CV or an expression. If the value for state is a decimal value, then the value is rounded to the nearest integer. A state value of 0 turns the output OFF, while any value other than 0 turns the output ON.

Delay Period and Reset

The **delay period** channel option causes the DT800 to pause before proceeding with its program. During the delay period only time-keeping, counting and servicing of the serial interfaces is maintained. Long delay periods slow the DT800's response when being polled for data by the host.

The **reset** channel option (**R**) resets the digital outputs to the opposite state to that set, thereby producing a pulse output.

If a delay period is not specified, then the digital state output is toggled quickly, producing a pulse width of 10–100ms. However, if a delay period is specified, the output resets after the delay period producing a longer pulse.

The delay period and reset channel options have limited application when switching the digital state outputs directly from commands, but have considerable application when switching digital state outputs from within report schedules — see "Digital State Outputs and Report Schedules" on page 155.

Examples — Digital State Delay and Reset

The command

```
4DSO(1000)=1
```

turns digital state output 4 ON, then pauses for 1000ms (1 second) before continuing.

■ The command

```
2DSO(7500,R)=1
```

turns digital state output 2 ON, pauses for 7.5 seconds, then resets the output OFF.

■ The commands

```
3DSO(0,R)=1
```

and

```
3DSO(R)=1
```

demonstrate two ways of turning digital state output 3 ON then immediately turning it OFF.

Switching Digital Byte Outputs

The DT800 is also able to switch its digital state outputs (digital channels 1 to 8) as **byte outputs**. Digital byte outputs are implemented as pairs of successive digital bit output channels.

The digital byte output command can be used to switch pairs of digital state output channels simultaneously, producing a bit pattern that is represented by decimal values assigned to the byte channel. For example, the command

```
1DBO=13
```

represents the bit pattern 00001101, which turns channels 1, 3 and 4 ON, and channels 2, 5, 6, 7 and 8 OFF.

Digital bytes are switched beginning at digital output **D1**, for the next eight successive digital outputs.

The digital byte output is specified in a channel list by

```
nDBO(mask)=pattern
```

where

n	is the first channel to begin switching the byte — must be **D1** (***n*=1**)	
DBO	is the digital byte output channel type	
mask	is an optional bit mask of which bit channels are to be included in the byte, beginning from ***n***	Allows individual digital outputs within the pair to be selected, as specified by the bits that are set in the mask. Channels associated with bits that are not set in the mask are not switched by the byte output.
pattern	is the decimal value of the bit pattern required	Can be specified as a constant, a CV or an expression. If pattern is a decimal value, the value is rounded to the nearest integer.

The **digital byte output number** is defined as the first digital bit output channel to be switched.

If a byte mask is not specified, then the mask defaults to decimal 255 (11111111) and all channels from the nominated first channel are switched.

Note The delay and reset options for digital bit outputs are not supported for digital byte outputs.

Note While a digital state output channel of the DT800 is turned ON, it cannot be used to monitor digital inputs. Therefore, before using the digital channels for monitoring digital logic states or for counting, it is good practice to send the command

```
1DBO=0
```

to ensure all outputs are OFF.

Example — Digital Byte

■ The command

```
5CV=11 1DBO=5CV
```

turns digital state outputs 1, 2 and 4 ON (**11** ⇒ 00001011), and turns digital state outputs 3 and 5–8 OFF.

Digital State Outputs and Report Schedules

The digital bit, nybble and byte outputs can be made to change state at intervals determined by report schedules, with delay and reset channel options.

The delay period and reset operate in the same manner as for direct digital state output commands. During the delay period all analog-to-digital conversion is suspended.

Switching digital state outputs in this way allows them to be used for tasks such as generating pulse trains and controlling relays that switch power to sensors only when the sensors are to be read.

Digital Bit Scans Digital bit output channels can be switched at regular intervals of time by repeat scan schedules. For example, the schedule commands

```
R10M 1DSO(5000)=1 1DSO=0
```

and

```
R10M 1DSO(5000,R)=1
```

show two ways of turning digital state output 1 ON for 5 seconds every 10 minutes.

Note The switching can be completed either by turning the digital state output OFF again with no delay, or by adding the reset option (**R**).

Digital Byte Scans The digital byte output can be used to switch pairs of digital state outputs at regular intervals of time by the repeat scan schedules. For example, the command

```
R30M 1DBO=6
```

turns digital state outputs 2 and 3 ON (**6** ⇒ 00000110) after 30 minutes.

R*x*X Schedule and Digital State Outputs The polled schedule (see "Trigger on Schedule-Specific Poll Command" on page 47) can also be used to switch digital state output channels. For example, the command

```
RBX 3DSO(5500,R)=1
```

turns digital state output 3 ON for 5.5 seconds when an **XB** poll command is received.

Examples — Digital State Outputs and Report Schedules

■ The schedule command

```
RA2S 1DS0(1000,R)=1
```

produces a pulse train from digital state output 1, which is ON for 1 second and OFF for 1 second.

■ In the schedule command

```
RA20M D T 4DSO(1000)=1 1..5V 4DSO=0
```

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.

Counting Digital State Output Switching

Changes of state of digital state outputs in response to switching commands can be counted by the counters associated with each of the digital channels 1 to 8.

The counters are incremented when the output switches ON, and can be read by report schedules.

Input-Only Digital Channels (D9 to D16)

Any of the DT800's eight input-only digital channels (channels D9 to D16 in Figure 96) can be used as either

- a digital input (for monitoring digital states), or
- a counter input.

Digital State Inputs D9 to D16

You can configure the DT800's digital channels D9 to D16 to operate as logic-level digital state inputs that recognize input levels as follows:

1	above 2.2V
0	below 1.1V

The channel types, triggering, channel options, and wiring details you use for D9 to D16 are the same as those for D1 to D6 — see "Digital State Inputs D1 to D6" on page 153.

Counter Inputs C9 to C16

See "Counters" on page 157.

Counters

You can configure the DT800's digital channels to operate as digital counters (named **C1**[19] to **C16**).

The counters increment on rising transitions.

All counters are 32-bit:

- Counters C1 to C6 are 4-bit in hardware (roll over every 16 counts), and can count at rates of up to 3Hz when the DT800 is asleep and 100Hz when the DT800 is awake.
- Counters C7 and C8 are 12-bit in hardware (roll over every 4096 counts), and can count at rates of up to 1kHz when the DT800 is asleep and 10kHz when the DT800 is awake.
- Counters C9 to C16 are 32-bit in software, and can count at rates of up to 40Hz when the DT800 is awake. These counters do not operate when the DT800 is asleep.

Counter Channel Type

The following channel type is provided for defining counter inputs:

C	Up counter	Counters 1 to 6: 100Hz max; counters 7 and 8: 10kHz max; counters 9 to 16: 40Hz max (maximum count speed is achievable only when DT800 is awake)

See the Counter category in the DT800 Channel Types table (page 66).

Counters as Schedule Triggers

Counters can trigger schedules

- by events — see "Trigger on External Event" on page 46
- while a condition is true — see "Trigger While" on page 48.

This function works only while the DT800 is awake, and at a maximum speed of 40Hz.

Counter Channel Options

When reading a counter, use the form

***n*C(R,*wrap*)**

where

n	is the counter channel number	
R	automatically resets the counter to zero after it has been read — see **R** in the DT800 Channel Options table (page 73)	Optional
wrap	is a number of counts after which the counter automatically resets — see ***wrap*** in the DT800 Channel Options table (page 73) and "Example — Counter Wrap" below	Optional

[19] In this manual, the numbers of digital channels used as counters are prefixed by the letter **C**.

When setting a counter — that is, assigning a value or the result of an expression to it — use the form

n*C(*wrap*)=*expression

where

n	is the counter channel number	
wrap	is a number of counts after which the counter automatically resets — see ***wrap*** in the DT800 Channel Options table (page 73) and "Example — Counter Wrap" below	Optional
expression	is a numeric value or expression — see "Example — Setting Counters" on page 157	

R is not valid when setting a counter.

Counter Rollover Rate

The DT800 wakes each time one of its digital counters rolls over. Therefore, if you're trying to get maximum life from the DT800's internal battery, use C7 and C8 (12-bit in hardware) where possible because these two roll over less frequently than counters 1 to 6 (4-bit in hardware). C9 to C16 do not operate while the DT800 is asleep.

Example — Counter Wrap

When reading a counter, counter wrap is set as a channel option and has a maximum value of $2^{32}-1$. For example

```
1C(3)
```

sets the range of counter **1** to **3**. On the third input pulse, the counter resets to zero:

Input pulse number	0	1	2	3	4	5	6	7	8	9	10	11
Counter reading	**0**	**1**	**2**	**0**	**1**	**2**	**0**	**1**	**2**	**0**	**1**	**2**

Example — Setting Counters

When setting a counter, the **R** channel option cannot be used. Here are two examples:

```
1C=15
2C(10)=1CV/100*SIN(2CV/3CV)
```

You can also include such assignments in a schedule so that they are executed on each scan.

Troubleshooting Digital Channels

- For channels D1 to D6, applying a signal that exceeds the recommended maximum frequency for these channels (100Hz) may cause slowing of other DT800 functions.
- Using any of channels D1 to D8 as both a counter channel for triggering schedules and as a data channel at the same time is not recommended (different signal restrictions apply to these functions) and may result in the counter having different values for each function.
- By default, using the digital counter channels does not keep the DT800 awake. If you need the DT800 to stay awake for longer than its pre-sleep period (default is 30 seconds of inactivity), alter P15 — see "Controlling Sleep" on page 43.

SERIAL CHANNEL

The DT800's Serial Channel (see Figure 98) 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

- is a DTE device (see "DTE" on page 206)
- can be configured for either the RS-232, RS-422, RS-485 or SDI-12 comms standard (where RS-232 supports single devices, and the other standards support multiple devices in a multidrop configuration)
- has a differential transmitter and receiver that provide for the different serial standards
- has RTS/CTS handshake lines
- supports baud rates of 50 to 115200 baud
- has switched power available (on the **12V** terminal — see Figure 98).

The output or prompt string is programmable, and commands are available to interpret incoming strings.

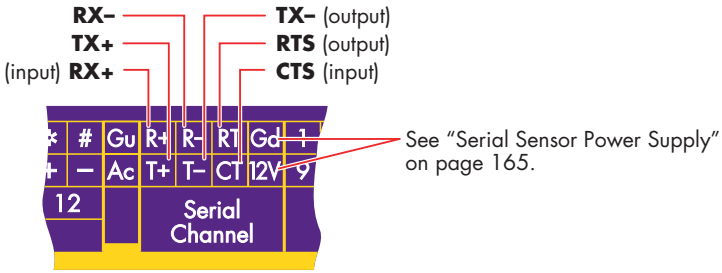


FIGURE 98 The DT800's Serial Channel terminals (DTE)

Setting Serial Channel Quantifiers

The Serial Channel communications quantifiers are set by the command

`PS=baud,parity,databits,stopbits,flowcontrol`

where

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**.
parity	can be **N** (none), **O** (odd) or **E** (even)
databits	can be **7** or **8**
stopbits	can be **1** or **2**
flowcontrol	can be **NOFC** (no flow control), **HWFC** (RTS/CTS hardware flow control) or **SWFC** (XON/XOFF software flow control)

For example, the command

`PS=9600,N,8,1,SWFC`

sets the Serial Channel comms quantifiers to 9600 baud, no parity, 8 data bits, 1 stop bit and XON/XOFF flow control. See also "Configuring the Serial Channel" on page 167.

These communications settings are stored in the DT800's SRAM and are therefore retained through a soft reset. (After a hard reset, the settings default to PS=1200,N,8,1,NOFC.)

Setting Serial Channel Quantifiers for SDI-12

Whenever the Serial Channel is used to interface to SDI-12 devices, define the communications settings as follows:

`PS=1200,E,7,1,NOFC`

Serial Channel Commands

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
- interpretation and parsing of input strings into *dataTaker* variables
- general management of the Serial Channel
- monitoring and debugging of the operation of the Serial Channel.

The same Serial Channel commands are used to supervise RS-232, RS-422, RS-485 or SDI-12 devices connected to the Serial Channel terminals.

The general form of the Serial Channel command is

`nSERIAL("control_string",options)`

where

n	is the Serial Channel number. For a DT800, this is always **1**.
control_string	is a string of commands that specify the required output and input actions of the Serial Channel. See "Control String" on page 159.
options	is a list of channel options that can be used to modify general features of the Serial Channel. There are options to select the configuration of the Serial Channel, the timeout, and whether response codes are returned. See "Serial Channel — Channel Options" on page 164. The options can be placed either before or after the **"*control_string*"** in the Serial Channel command.

See also "Serial Channel Examples" on page 166.

DeTransfer — Use a Double Backslash

Important The DeTransfer program, which is often used to supervise the DT800, has a number of commands that have a leading **** (backslash) character (see the DeTransfer Help). The backslash character indicates to DeTransfer that the following command is to be interpreted and executed by DeTransfer. For example:

- **\w3** instructs DeTransfer to wait 3 seconds before communicating again with the *dataTaker*.
- **\t** instructs DeTransfer to read the computer clock and format a current time string.
- **\013** instructs DeTransfer to send the single character carriage return to the connected *dataTaker*.

The DeTransfer backslash commands are not sent to the connected *dataTaker*.

However, DeTransfer has an escape mechanism to do this: if a command is preceeded by a double backslash (that is, ***command***), the command is sent to the connected DT800 as

a single backslash command (that is, ***command***) and is not interpreted and executed by DeTransfer.

Coincidentally, a number of the Serial Channel commands also have a leading **** (backslash) character and, if entered into DeTransfer as such, are not sent to the connected DT800. Therefore, the Serial Channel commands that have a leading **** (backslash) character must be entered into the Send window of DeTransfer with a double backslash. For example:

- To send the Serial Channel command **\e**, enter **\\e** into DeTransfer.
- To send the Serial Channel command **\r1**, enter **\\r1** into DeTransfer.
- To send the Serial Channel command **\w[1000]**, enter **\\w[1000]** into DeTransfer.
- To send the Serial Channel command **\013**, enter **\\013** into DeTransfer.

Note that this rule only applies to DeTransfer. Although other terminal programs can be used to supervise the DT800, most do not interpret the **** (backslash) character, and so the Serial Channel backslash commands must be entered as single backslash commands. Consult the terminal program's documentation for details.

Control String

The **"*control_string*"** component of the Serial Channel command comprises three parts:

- **Output actions** — commands, prompts or text strings that are to be sent from the DT800 to the device connected to the serial channel. The various output actions available are detailed in the section "Output Actions" on page 161. All output actions are enclosed by **{}**.
- **Input actions** — commands to manage the DT800'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 "Input Actions" on page 162. Input actions are not enclosed by **{}**.
- **Quotation marks** — enclose the control string.

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 DT800 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
  1SERIAL("{output actions}",options)
  1SERIAL("input actions",options)
END
```

This 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.

Example — Control String

The control string in the Serial Channel command

```
1SERIAL("{WN\013}\w[1000]%d[1CV],%f[2CV]{C\013}\w[2000]\e")
```

specifies the following output and input actions for supervising electronic weighing scales connected to the serial channel of a DT800 (the scales have a Weigh Now command **WN**, which instructs the scales to perform a weighing operation):

{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.)

\w[1000]%d[1CV],%f[2CV]

An input action.

\w[1000] is a DT800 command to wait one second for the scales to return data.

These scales return two values: a batch number as an integer, and the weight as a floating-point value.

- **%d[1CV]** is a DT800 command to 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]** is a DT800 command to 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]\e

An input action.

\w[2000] is a DT800 command to wait two seconds for the scales to clear.

\e erases any extraneous characters returned into the receive buffer of the serial channel during the two seconds.

Remember that if this control string is sent to the DT800 from DeTransfer, the backslash commands must be sent as double backslashes:

```
"{WN\\013}\\w[1000]%d[1CV],%f[2CV]{C\\013}\\w[2000]\\e"
```

(See "DeTransfer — Use a Double Backslash" on page 158.)

Executing Serial Channel Commands in Schedules

The Serial Channel command is placed into any scan schedule, which determines when the Serial Channel is accessed. Any other input channels, calculations, alarms and so on can be in the same schedule if desired.

Polling the Serial Channel from Schedules

If the Serial Channel is to be polled on a regular basis, this is usually done from a schedule triggered at a regular interval of time, for example

```
BEGIN
   RA1M 1SERIAL("control_string",options)
END
```

where the device connected to the Serial Channel is polled every minute according to the defined control string and options.

Whenever the device on the Serial Channel is polled in this way, the receive buffer pointer is pointing at the first character of the response received into the receive buffer. Interpretation of the input string is performed from this point, using appropriate input actions.

Accessing the Serial Channel by Serial Trigger

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. Any schedules (**Ra**) can be defined to trigger on the receipt of specified characters at the Serial Channel as follows:

```
Ra1SERIAL"text" 1SERIAL("control_string",options)
```

where a match for ***text*** in the incoming data stream produces a trigger,
or

```
Ra1SERIAL"control_char" 1SERIAL("control_string",options)
```

where a match for ***control_char*** in the incoming data stream produces a trigger (the ***control_char*** are defined using the ^*char* convention — for example, **^B** is STX, **^M** is carriage return, **^G** is bell),
or

```
Ra1SERIAL"" 1SERIAL("control_string",options)
```

where any character received into the Serial Channel produces a trigger.

Whenever the Serial Channel produces a trigger by any of these methods, the receive buffer pointer is pointing at the first character that produces the interrupt. Interpretation of the input string is performed from this point, using appropriate input actions.

Output Actions

The table below lists the commands, prompts and text strings that can be sent from the DT800 to the device connected to the Serial Channel. These must be enclosed by **{}** in the control string.

Note that some output actions are also allowed in input actions (see page 162).

Plain text	***text***	A sequence of printable characters to be **sent** (ASCII 32 to126, and 128 to 255; case-sensitive)
Special characters	**\nnn**	Defines any ASCII character by decimal code (1 to 3 digits) to be **sent** — for example, **\013** is **CR** (carriage return)
Special characters	**^*char***	Defines any ASCII control character (ASCII 1 to 31) to be **sent** — for example, **^M** is carriage return
Break character	**\b[*n*]** or **\b[*n*CV]**	Break for ***n*** of ***n*CV** character periods
Handshake RTS	**\r1**	Set RTS (to a value >+3.5V)
	\r0	Clear RTS (to a value <–3.5V)
Wait	**\w[*n*]**	Wait ***n*** milliseconds
	\w[*n*CV]	Wait ***n*CV** milliseconds
Erase receive buffer	**\e**	Erase (clear) all characters from the receive buffer (sets read pointer to buffer start)
Print receive buffer	**\p**	Print current contents of receive buffer to host comms port then erase buffer
Enable transmit while receiving	**\x1**	Set **TXON**
	\x0	Set **TXOFF**
% character	**%%**	Output **%** character
Format value	**%{*flag*}{*width*}{*.precision*}*type*[*n*CV]**	Output ***n*CV**
	%{*flag*}*{*.precision*}*type*[*w*CV,*n*CV]	Output ***n*CV** with width specified by ***w*CV**
	%{*flag*}*.*.*type*[*w*CV,*p*CV,*n*CV]	Output ***n*CV** with width specified by ***w*CV** and precision specified by ***p*CV**

Note Braces **{}** here signify "optional". For ***flag*** and ***type***, see sub-tables.

flag	
–	Left justify
+	Prefix positive value with **+**
space	Prefix positive values with a space and negative values with **–**
# applied with type **o**	First digit will always be **0** (zero)
# applied with type **g**, **G**, **e**, **E** or **f**	Forces the output value to contain a decimal point
0 (zero)	Pad to field width with zeros

type		**Example (where *n*CV=71.36)**
d or **i**	Output ***n*CV** as a decimal integer	71
o	Output ***n*CV** as an octal integer	107
x or **X**	Output ***n*CV** as a hexadecimal integer	47
f	Output ***n*CV** as a floating-point real	71.36
e or **E**	Output ***n*CV** as a floating-point real with exponent	7.136e01 7.136E01
g or **G**	Best of **f** or **e** formats for the width	7.136e01 7.136E01
c	Outputs ***n*CV** as a single ASCII character represented by this decimal	G
s or **S**	Outputs ***n*$** as a character string	

Table: DT800 Serial Channel — output actions

Input Actions

The table below lists the commands available to manage the DT800's Serial Channel and to interpret the information coming back into the Serial Channel from the serial device. Input actions are not enclosed by **{}** in the control string.

Note that some input actions are also allowed in output actions (see page 161).

Plain text	***text***	A sequence of printable characters to be **skipped** (ASCII 32 to126, and 128 to 255; case-sensitive)
Special characters	***nnn***	Defines any ASCII character by decimal code (1 to 3 digits) to be **skipped** — for example, **\013** is **CR** (carriage return)
Special characters	**^*char***	Defines any ASCII control characters (ASCII 1 to 31) to be **skipped** — for example, **^A** is SOH, **^M** is carriage return
Handshake	**\c1[*n*]**	Wait ***n*** milliseconds for CTS to be set (to a value >+3.5V)
	\c1[*n*CV]	Wait ***n*CV** milliseconds for CTS to be set (to a value >+3.5V)
	\c0[*n*]	Wait ***n*** milliseconds for CTS to be cleared (to a value <–3.5V) or for disconnect
	\c0[*n*CV]	Wait ***n*CV** milliseconds for CTS to be cleared (to a value <–3.5V) or for disconnect
Wait	**\w[*n*]**	Wait ***n*** milliseconds
	\w[*n*CV]	Wait ***n*CV** milliseconds
Erase receive buffer	**\e**	Clear all previously arrived characters from the serial input buffer
Transmit enable	**\x1**	Set **TXON**
	\x0	Set **TXOFF**
Match text	**\m[*text*]**	Discard incoming characters until ***text*** found; keep all characters thereafter
	\m[*n*$]	Discard incoming characters until text in ***n*$** found; keep all characters thereafter
Print receive buffer	**\p**	Print current contents of receive buffer to host comms port then erase buffer

Table: DT800 Serial Channel — input actions (sheet 1 of 2)

Scan value

`%{width}type[nCV]`	Scan result into `nCV`
`%*{width}type`	Scanned but not assigned

Note Braces `{}` here signify "optional". For `type`, see sub-table.

`type`		Example: where 123.456 is input, `nCV` contains...	
`d` or `i`	Interpret as an optionally-signed decimal integer	**123**	(**.456** is left in the receive buffer)
`o`	Interpret as an optionally-signed octal integer	**73**	(**.456** is left in the receive buffer)
`x` or `X`	Interpret as an optionally-signed hexadecimal integer	**273**	(**.456** is left in the receive buffer)
`g`, `G`, `e`, `E` or `f`	Interpret as an optionally-signed floating-point real number	**123.456**	(nothing is left in the receive buffer)
`c`	Interpret as a single decimal ASCII character	`%c` or `%1c` reads the first character (**1**) and loads 49 into `nCV` (ASCII 1 = decimal 49; **23.456** is left in the receive buffer). `%3c` skips the first two characters, reads the 3rd character (**3**) and loads 51 into `nCV` (ASCII 3 = decimal 51; **.456** is left in the receive buffer).	
`b`	Interpret as a binary number. `%b` reads one byte, or a `width` can be specified (for example, `%4b` causes four bytes to be read and treated as an unsigned 32-bit number). Where more than one byte is read it is assumed that first byte scanned is the most significant byte (that is, byte order is "big-endian").		
`[abc]`	Interpret as a scanset. That is, receive a single string containing only the letters `a`, `b` and `c`. For example, `%[pq3rs][1$]` instructs the Serial Channel to receive the specific string **pr3rs** only, and place it into the text channel **1$**. The scanset can be negated using the caret character (^). For example, `%[^1234567890]` instructs the Serial Channel to receive a string that contains characters other than digits. See also the **Note** in `['string1','string2',...,nCV]` below.		
`s`	Interpret as longest possible string up to width (80 chars maximum if no width specified). Spaces and tabs are included in string, but not carriage return or linefeed.		
`S`	Interpret as longest possible string up to width (80 chars maximum if no width specified) or until first whitespace is found. Spaces, tabs, carriage return and linefeed are not included in string. See also the **Note** in `['string1','string2',...,nCV]` below.		
`s['string1','string2',...,nCV=m]`	Interpret as a chain of text strings: If the incoming data matches `string1`, assign **0** to `nCV`. If the incoming data matches `string2`, assign **1** to `nCV`. If the incoming data matches `string3`, assign **2** to `nCV`. ⇓ If the incoming data matches none of the specified strings, assign the optional default value `m` (if included) to `nCV`. For example, `%s['ab','345','fgfg4',2CV=8]` (where `=8` is the optional default value) instructs the Serial Channel to interpret the incoming data as a string and do the following: • If the incoming string matches `ab`, assign **0** to `2CV`. • If the incoming string matches `345`, assign **1** to `2CV`. • If the incoming string matches `fgfg4`, assign **2** to `2CV`. • If the incoming string matches none of the specified strings, assign **8** to `2CV`. **Note** `['string1','string2',...,nCV]` can be applied to `%s`, `%S` and `%[abc]`.		

Table: DT800 Serial Channel — input actions (sheet 2 of 2)

Table: DT800 Channel Options

Serial Channel — Channel Types

Category	Signal/Sensor Details	Channel Type	Example	Default Channel Option(s)	Channel Factor (channel option for scaling, etc.)	Resolution	Output Units	Wiring Configuration and Default	Fundamental Samples	Comments
			See Channel Options table below.							
Serial Channel	Transmit to and receive from serial device via RS-232, RS-422, RS-485 or SDI-12	*n*SERIAL	1SERIAL	RS232	Timeout to receive the first characters, in seconds. Defaults to P53 if not specified.		State 0=success; otherwise see reported error code for details.	Figure 99	0	See page 158 for prompt and scan definition. Use W or NR channel options to prevent state output.
	Switched output from the **12V** terminal of the Serial Channel (see "Serial Sensor Power Supply" on page 165)	*n*SSPWR	1SSPWR=0 1SSPWR=1	1SSPWR=0			State (0 or 1)		0	12V supply for serial sensors 0 = power off 1 = power on
	Option to turn transceiver on/off at other than default times	*n*SSPORT	1SSPORT=0 1SSPORT=1	1SSPORT=1			State (0 or 1)		0	Power supply for transceiver 0 = power off 1 = power on
	Serial Channel negative source voltage (from External Power input)	SSVN	SSVN				V		2	–6.5V±1V

Table: DT800 Serial Channel — Channel Types

Serial Channel — Channel Options

Category	Channel Option	Mutual Exclusions Function	Range of Option (n)	Order of Application	Comment	
Communication Type	RS232	Specifies default transmitter and receiver behaviour for RS-232			RS232 is the default if none of these is specified. These four channel options control the configuration of the Serial Channel. See "Configuring the Serial Channel" on page 167.	
	RS422	Specifies default transmitter and receiver behaviour for RS-422				
	RS485	Specifies default transmitter and receiver behaviour for RS-485				
	SDI12	Specifies default transmitter and receiver behaviour for SDI-12				
Timeout	*n*	Timeout to receive character input (seconds)	1 to 32767		Default is 10 seconds	
Logging of State	NL	Disables logging of state after execution			1SERIAL( ) logs **n State** if logging is enabled to indicate success or failure	See the Destination category in the DT800 Channel Options table (page 75).
Return of State	NR	Disables return of state after execution			1SERIAL( ) returns **n State** if returns are enabled to indicate success or failure	
Logging and Return of State	W	Disables logging or return of state after execution			1SERIAL( ) normally generates **n State** to indicate success or failure	

Table: DT800 Serial Channel — Channel Options

Serial Channel State

Whenever the Serial Channel command is executed, it normally returns a state value that indicates success or otherwise of the execution.

If the execution is successful, then **0 State** is returned. If there has been a problem, **n State** is returned, where **n** is the error state code relating to the problem. Here are the DT800's Serial Channel error state codes:

Code	Error	Code	Error
0	OK	**15**	Expecting input action arguments
1	Port not available	**16**	Integer too big
2	Transmit buffer overflow	**17**	Bad CTS definition
3	Receive buffer overflow	**18**	Bad RTS definition
4	Special char too large	**19**	Bad wait definition
5	CTS detect timeout	**20**	Receive timeout
6	Transmit timeout	**21**	Bad end of out block
7	Invalid output action format	**22**	Bad start of out block
8	Invalid output action	**23**	Bad end of spec character
9	Expecting output action format	**24**	Bad break definition
10	Expecting CV	**25**	Null control string
11	CV out of range	**26**	Bad match definition
12	Expecting end of arguments	**27**	n$ out of range
13	Illegal input action width	**28**	Expecting n$
14	Illegal input action definition		

Return of the Serial Channel state can be disabled by the `W` channel option — see page 164.

Serial Sensor Power Supply

The Serial Channel has a power supply terminal for powering serial and other devices if required. Although the terminal is labelled **12V** (see Figure 11 on page 24), it actually supplies the following:

- the voltage of the internal main battery (8.5Vdc to 13Vdc) when the DT800 is operating from its internal main battery only
- the voltage provided at the DT800's external power **11–28Vdc** inputs (see Figure 12 on page 25) when either of these is used — "flow-through" voltage.

Current from this terminal is limited to 250mA over the 8.5Vdc–28Vdc range.

The power supply can be switched on by the command `1SSPWR=1` and switched off by the command `1SSPWR=0`.

These commands return state values to indicate whether the supply is switched off (**0 State**) or on(**1 State**).

The serial sensor power supply terminal is usually managed within the same schedule when access to the Serial Channel occurs. For example, the program

```
BEGIN
   RA15M
   1SSPWR(W)=1
   1SERIAL(RS232,"\w[2000]{READ\013}\w[1000]%f[10CV]\e")
   1SSPWR(W)=0
END
```

powers up the sensor, waits 2 seconds (`\w[2000]`) while the sensor stabilizes, issues a READ command (`{READ\013}`), waits 1 second (`\w[1000]`) while the sensor responds, parses the return data into 10CV (`%f[10CV]`), then erases whatever is left in the buffer (`\e`).

The `W` channel option disables the return of the Serial Channel state.

See also "Powering the DT800's Modem" on page 130.

Serial Channel Debugging Tools

P56 Debugging

The DT800 has some useful debugging modes available for the Serial Channel. These allow the user to send information about the progressive operation of the Serial Channel from the host communications port, for viewing in DeTransfer or other terminal software.

These debugging modes are activated by setting Parameter56 as follows:

`P56=0`	No debugging information (default)
`P56=1`	Show contents of the receive buffer when each character is read as `ReadByte[contents of receive buffer<EndOfBuf>]`

`P56=2`	When `\e` command is executed, show contents of receive buffer before and after erasure:
	`EraseBefore[contents of receive buffer<EndOfBuf>]`
	`EraseAfter[contents of receive buffer<EndOfBuf>]`
	When `\w` command is executed, show contents of receive buffer before and after the wait period:
	`WaitBefore[contents of receive buffer<EndOfBuf>]`
	`WaitAfter[contents of receive buffer<EndOfBuf>]`

`P56=4`	Show contents of transmit buffer before and after transmission:
	`Transmit:b[contents of receive buffer<EndOfBuf>]`
	`Transmit:a[contents of receive buffer<EndOfBuf>]`

`P56=8`	Returns type of parsing scan effected each time data is parsed:
	`%*zScan:NoAssign[ScanControl]` where z is any scan ***type*** (c, d, f, g, i, o, x,...)
	`%d[ ]Scan:Integer[ScanControl]`
	`%f[ ]Scan:Real[ScanControl]`
	`%s[ ]Scan:String[ScanControl]`

`P56=16`	Indicates where a `1SERIAL("control_string")` command failed

`P56=32`	Shows the contents of the receive buffer after a trigger:
	`NullTrig[contents of receive buffer<EndOfBuf>]`
	`CharTrig[contents of receive buffer<EndOfBuf>]`

Combinations of debug information can be returned by combining those required in the P56 setting. For example, setting `P56=17` shows contents of the receive buffer (P56=1) when each character is read, and where the 1SERIAL("control_string") was in error if an error occurred (P56=16).

Error Messages

If the DT800 is in free-format mode (/h), any syntax error message for the serial command string will include a reference to the error's position in the command string.

Serial Loopback

A useful technique for testing your parsing commands is to implement a serial loopback in the RS-232 mode. Simply connect the **R-** and **T-** 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.

For example, if analog loopback is implemented, the commands

```
1SERIAL("\\e{ABCD,1234\\013}%4s[1$],%4d[1CV]") 1$ 1CV
```

return `ABCD` and `1234.0` into `1$` and `1CV` respectively.

This test also allows you to differentiate between a failed sensor and program errors.

Serial Channel Examples

Control String	Action
`1SERIAL(RS232,"XyZ ")`	Discard the text **XyZ** and one or more whitespace characters; case-sensitive
`1SERIAL("%d[1CV] %8f[2CV] %c[3CV]")`	Decode integer, float and character with separating whitespaces
`1SERIAL("\C1[2000]%d[1CV]")`	Wait 2 seconds for CTS to become high, then decode integer
`1SERIAL("{1..3TK\013\010}")`	Send **1..3TK** followed by a carriage return command (**\013**) and a line feed command (**\010**) to a device on the Serial Channel
`1SERIAL("{%f[1CV] %d[2CV] %c[3CV]}")`	Output **1CV** as float, **2CV** as integer and **3CV** as character to a printer or display connected to the Serial Channel
`1SERIAL("{\R1\W[1CV]\R0}")`	Raise RTS, wait **1CV** milliseconds, then drop RTS
`1SERIAL(RS485,"{D2READ\013}%2.2f[1CV],%2.2f[2CV]")`	Send a **READ** command to the sensor at address **D2** on multidrop **RS485** network, decode two returned floats separated by a comma into the two CVs

Note When sending Serial Channel commands from DeTransfer, add a second backslash to any part of the command that requires a backslash. For example, **\13** becomes **\\13**, and **\10** becomes **\\10**.

Schedule	Action
`RA1SERIAL"STX" 1SERIAL("STX%d[1CV]") 1CV=1CV+1`	Trigger schedule **A** on arrival of **STX**, scan the integer into **1CV** and increment it (barcode scanner)
`RA1M 1SERIAL("{Num Temp Press\13\10%3d[1CV] %4.1f[2CV]}")`	Send **Num Temp Press** then **CRLF** and, on new lines, **1CV** as integer and **2CV** as float (report generation)

`BEGIN`
`RA1M`
`1V(=1CV) 2TK(=2CV)`
`1SERIAL("{Flow %10.1f[1CV]\13\10 Temp %10.1f[2CV]\13\10}")`
`END`
	Read data from analog channels and report as text to a printer connected to the serial channel, in the same schedule (report generation)

`PS=9600,N,8`
`1SSPWR=1`
`BEGIN`
`RA1SERIAL"$GPRMC"`
`1SERIAL(RS232,",%f[8CV]%c[9CV],%f[10CV]%c[11CV]\e",2,W)`
`D T 8..11CV(FF2)`
`END`
	Program to read latitude (**8CV**) and North or South (**9CV**), longitude (**10CV**) and East of West (**11CV**) from the NMEA 183 data stream issued continuously from a GPS unit. Data is logged with date and timestamp. Each read is triggered by the **$GPMRC** header at the start of each transmission.

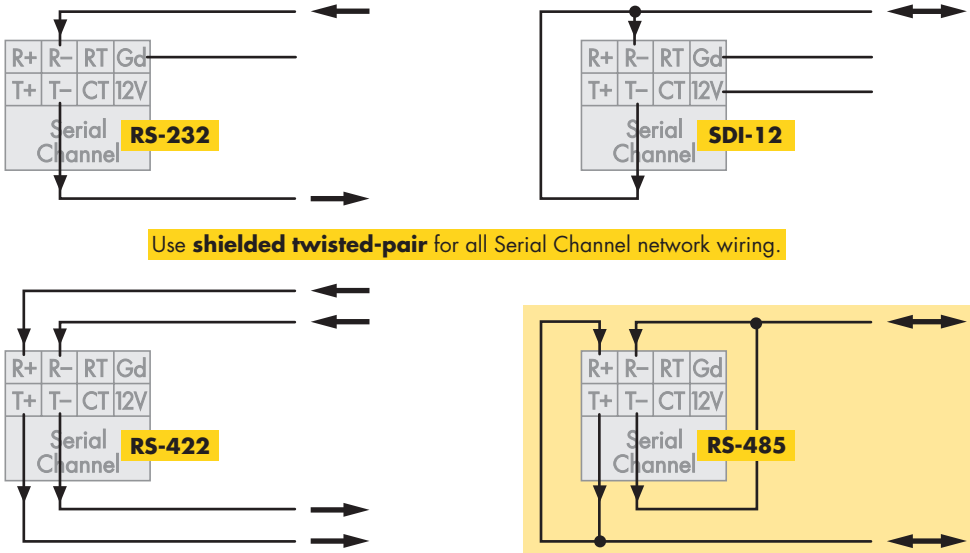
`BEGIN`
`PS=1200,E,7,1,NOFC`
`1SSPWR=1`
`1SERIAL(SDI12,"{\e}",W)`
`RA15M`
`1SERIAL(SDI12,"{\b[1]0M!}",W)`
`1SERIAL(SDI12,"\w[1000]%1d[1CV]%3d[2CV]%1d[3CV]\p\e",W)`
`1CV("Address ",FF0)`
`2CV("Proc Time ",FF0)`
`3CV("No. Values ",FF0)`
`1SERIAL(SDI12,"{\b[1]0D0!}",W)`
`1SERIAL(SDI12,"\w[1000]%1d[4CV]%6f[5CV]\e",W)`
`4CV("Address ",FF0)`
`5CV("River Height",FF2)`
`END`
	Program to read various status information and the unit address and river height from an SDI-12 river height sensor every 15 minutes. Note use of break character (**\b[1]**) to precede each command to the SDI-12 device. Sampling data is a two-step process: first instruct the SDI-12 device to make a new reading, then instruct it to return the new data.

Note When sending Serial Channel commands from DeTransfer, add a second backslash to any part of the command that requires a backslash. For example, **\13** becomes **\\13**, and **\10** becomes **\\10**.

Configuring the Serial Channel

Communications Type	Channel Option	Multidrop	DT800 Terminals R+	R–	T+	T–
RS-232 (default)	**RS232**	No		Used		Used
RS-422	**RS422**	Yes	Used	Used	Used	Used
RS-485	**RS485**	Yes	T+	T–	R+	R–
SDI-12	**SDI12**	Yes		Used		Used

← That is, T+ is connected to R+ and T– is connected to R–.



Use **shielded twisted-pair** for all Serial Channel network wiring.

Terminating RS-485 Networks

An RS-485 network driven by the Serial Channel of a DT800 does not normally require termination.

However — if you're experiencing data communications problems (which may arise because the network is operating at a high data speed, over long lines or in a noisy environment) — we recommend that you add a terminating resistor (R_T) at the DT800 and at the node that is most distant from the DT800.

Each terminating resistor must be 1kΩ or greater. (Do not use 100Ω as is sometimes done in RS-485 networks.)

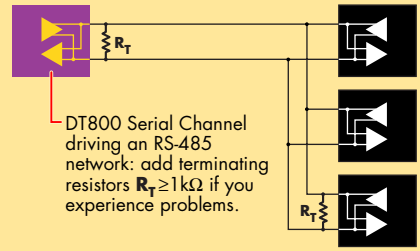


FIGURE 99 Serial Channel transmit and receive flows

Use the following commands to configure the comms characteristics of the DT800 Serial Channel (**PS** ⇒ Port Serial):

Command	Action	*b*	*p*	*d*	*s*	*f*
PS=*b*	Sets Serial Channel baud rate (DT800 default = 1200)	**50**, **75**, **110**, **150**, **300**, **600**, **1200**, **2400**, **4800**, **9600**, **19200**, **38400**, **57600**, or **115200**	**N** (none), **O** (odd), or **E** (even)	**8** or **7**	**1** or **2**	**NOFC** (no flow control), **HWFC** (hardware flow control), or **SWFC** (software flow control)
PS=*b*, *p*	Sets Serial Channel baud rate and parity					
PS=*b*, *p*, *d*	Sets Serial Channel baud rate, parity and databits					
PS=*b*, *p*, *d*, *s*	Sets Serial Channel baud rate, parity, databits and stopbits					
PS=*b*, *p*, *d*, *s*, *f*	Sets Serial Channel baud rate, parity, databits, stopbits and flow control					
PS	Returns the current Serial Channel comms settings (for example: `1200,N,8,1,NOFC`)					

Wiring Configurations — Analog Channels

This section contains configuration diagrams for wiring signals and sensors to the DT800's analog channels (channels 1 to 12 on the left of its front panel).

"Analog Channels — Introduction" beginning on page 12 covers important concepts you need to be familiar with to successfully use the wiring configurations presented here; concepts such as the DT800's terminal designations, channel pairs, independent inputs and shared-terminal inputs, and sensor excitation. "Which Analog Input Configuration Should I Use?" (page 14) also contains useful information.

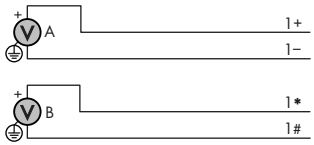
Voltage Inputs

Channel types:	**V**	voltage (see page 63)
	VAC	AC voltage (see page 64)
	TB, **TC**, **TD**, **TE**, **TG**, **TJ**, **TK**, **TN**, **TR**, **TS**, **TT**	thermocouples (see page 65)
	AS	analog state input measured as a voltage (see page 66)
	F	frequency (see page 64)

Independent Voltage Inputs

"Independent Analog Inputs" on page 13 introduces this wiring configuration.

Maximum number of independent voltage inputs to a DT800: 24 (two per analog channel).) Wiring gauge, length and environment is non-critical.



*	#
+	–
1	

Voltage source A: **1V**
Voltage source B: **1*V**

FIGURE 100 Wiring one or two independent voltage inputs to a single channel

Example commands for reading inputs of the type shown in Figure 100:			
4V	**5V**	**7*V**	**12*V**
4*V	**5*V**	**8V**	**11V**

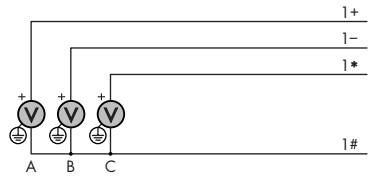
Shared-Terminal Voltage Inputs

"Shared-Terminal Analog Inputs" on page 13 introduces this wiring configuration.

Up to three of these inputs can share the **#** terminal on any single channel (Figure 101), or up to seven can share the even channel's **#** terminal in any channel pair (Figure 102).

Maximum number of shared-terminal voltage inputs on a DT800: 7 per channel pair x 6 channel pairs = 42. Wiring gauge, length and environment is non-critical.

Shared-Terminal Voltage Inputs on One Channel



*	#
+	–
1	

Voltage source A: **1+V**
Voltage source B: **1–V**
Voltage source C: **1*V**

FIGURE 101 Independent voltage inputs sharing a channel's **#** terminal

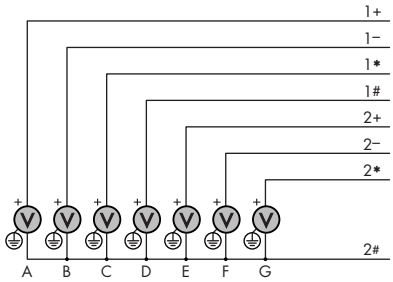
Example commands for reading inputs of the type shown in Figure 101:			
3+V	**4+V**	**7*V**	**10–V**
3–V	**4–V**	**8*V**	**12*V**
3*V	**4*V**	**9–V**	**12+V**

LEGEND

Voltage signal/sensor

See "Wiring Configurations and the Ac Terminal" on page 149.

Shared-Terminal Voltage Inputs on a Channel Pair



*	#	*	#
+	–	+	–
1		2	

Voltage source A: **1+V(#)**
Voltage source B: **1–V(#)**
Voltage source C: **1*V(#)**
Voltage source D: **1#V(#)**
Voltage source E: **2+V**
Voltage source F: **2–V**
Voltage source G: **2*V**

FIGURE 102 Independent voltage inputs sharing a channel pair's common return terminal, which is the even channel's # terminal

Example commands for reading inputs of the type shown in Figure 102:			
3+V(#)	**4+V**	**9+V(#)**	**10+V**
3–V(#)	**4–V**	**9–V(#)**	**10–V**
3*V(#)	**4*V**	**9*V(#)**	**10*V**
3#V(#)		**9#V(#)**	
Channel pair 3&4		Channel pair 9&10	

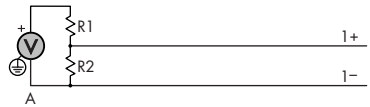
Attenuated Voltage Inputs

An attenuated voltage input
- lets you measure large voltages
- extends the common-mode range
- provides greater input protection.

Half-Attenuated Voltage Input

Attenuated voltage inputs for situations where one signal line is always close to ground potential. Can be used in independent or shared-terminal configurations (Figures 100, 101 or 102).

External attenuation resistors can be located near the DT800 terminals or at the signal source.



Attenuation $A = \frac{R1 + R2}{R2}$

*	#
+	–
1	

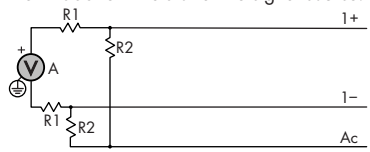
Attenuated voltage source A: **1V(*A*)**

FIGURE 103 Half-attenuated voltage input

Attenuated Voltage Input

For sensors with built-in amplification, the attenuation factor can be less than one, or negative for a sign reversal. Can be used in independent or shared-terminal configurations (Figures 100, 101 or 102).

External attenuation resistors can be located near the DT800 terminals or at the signal source.



Attenuation $A = \frac{R1 + R2}{R2}$

*	#
+	–
1	

Attenuated voltage source A: **1V(*A*)**

FIGURE 104 Attenuated voltage input

Example commands for reading inputs of the type shown in Figures 103 and 104:			
4V(10)	**5*V(50)**	**5+V(100)**	**9#V(100,#)**

LEGEND
Voltage signal/sensor
See "Wiring Configurations and the Ac Terminal" on page 149.

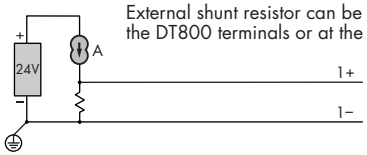
Current Inputs

Channel types:	I	current (see page 64)
	L	4–20mA current loop (see page 64)

Independent Current Input with External Shunt

You MUST adhere to the DT800's common-mode voltage limits (default is ±10V relative to analog common) for this configuration to operate correctly — see "Extending Common-Mode Range" on page 140.

One channel can read two independent current inputs — see Figure 100.



✻	#
+	–
1	

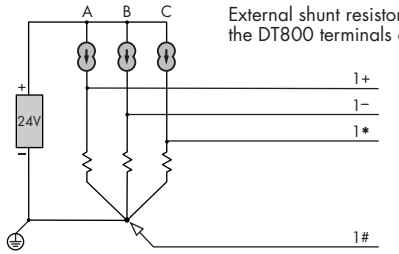
Current source A: `1I`
4–20mA current source A: `1L`

FIGURE 105 Independent current input

Example commands for reading inputs of the type shown in Figure 105:			
`4I`	`5I`	`7*I`	`12*I`
`4*I`	`5*I`	`8I`	`11I`

Shared-Terminal Current Inputs with External Shunts on a Single Channel

To avoid cross-channel coupling, connect the bottom of the shunts with the minimum of shared resistance to the sense point.



✻	#
+	–
1	

Current source A: `1+I`
4–20mA current source A: `1+L`

Current source B: `1–I`
4–20mA current source B: `1–L`

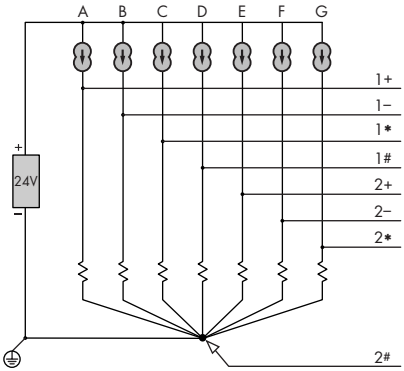
Current source C: `1*I`
4–20mA current source C: `1*L`

FIGURE 106 Shared-terminal current inputs on a single channel

Example commands for reading inputs of the type shown in Figure 106:			
`3+I`	`4+I`	`7*I`	`10–I`
`3–I`	`4–I`	`8*I`	`12*I`
`3*I`	`4*I`	`9–I`	`12+I`

Shared-Terminal Current Inputs with External Shunts on a Channel Pair

To avoid cross-channel coupling, connect the bottom of the shunts with the minimum of shared resistance to the sense point.



External shunt resistors can be located near the DT800 terminals or at the current sources.

✻	#	✻	#
+	–	+	–
1		2	

Current source A: `1+I(#)`
4–20mA current source A: `1+L(#)`

Current source B: `1–I(#)`
4–20mA current source B: `1–L(#)`

Current source C: `1*I(#)`
4–20mA current source C: `1*L(#)`

Current source D: `1#I(#)`
4–20mA current source D: `1#L(#)`

Current source E: `2+I`
4–20mA current source E: `2+L`

Current source F: `2–I`
4–20mA current source F: `2–L`

Current source G: `2*I`
4–20mA current source G: `2*L`

FIGURE 107 Shared-terminal current inputs on a channel pair

Example commands for reading inputs of the type shown in Figure 107:			
`3+I(#)`	`4+I`	`7*I(#)`	`10–I`
`3–I(#)`	`4–I`	`8*I`	`12*I`
`3*I(#)`	`4*I`	`9–I(#)`	`12+I`
Channel pair 3&4			

LEGEND: Kelvin sense point; Current source; Power supply (24V); See "Wiring Configurations and the Ac Terminal" on page 149.

Resistance Inputs

Current-excited voltage measurement

Channel type:	**R**	resistance (see page 64)
	PT385, **PT392**, **NI**, **CU**	RTDs (see page 65)
	YS01 to **YS07**, **YS16**, **YS17**	thermistors (see page 65)

4-Wire Resistance Inputs

* 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.

Independent 4-Wire Resistance Input on One Channel

Maximum number of independent 4-wire resistance inputs on a DT800: 1 per channel x 12 channels = 12.

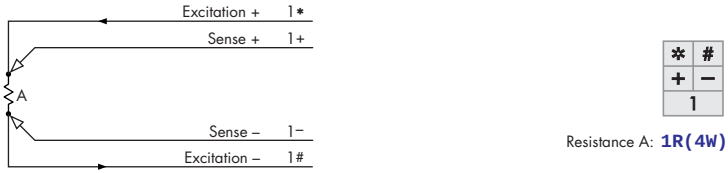


FIGURE 108 Independent 4-wire resistance input on one channel

Example commands for reading inputs of the type shown in Figure 108:			
2R(4W)	**3R(4W)**	**5R(4W,I)**	**12PT392(4W)**

Shared-Terminal 4-Wire Resistance Inputs on a Channel Pair

Maximum number of shared-terminal 4-wire resistance inputs on a DT800: 3 per channel pair x 6 channel pairs = 18.

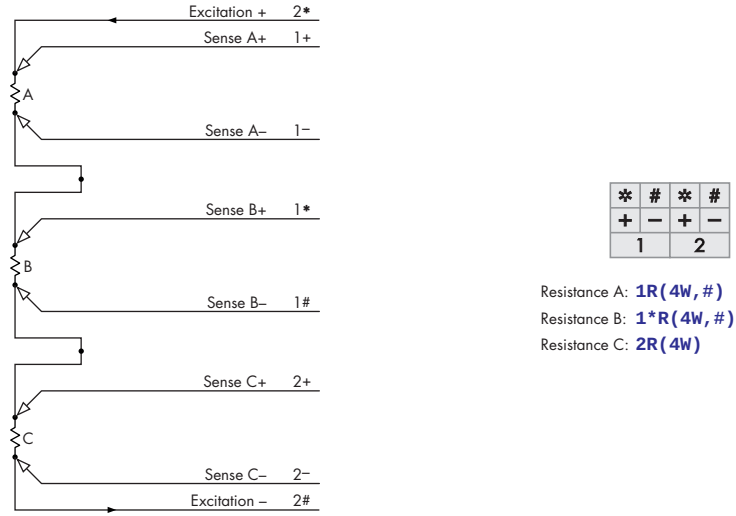


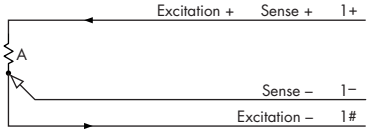
FIGURE 109 Shared-terminal 4-wire resistance inputs on a channel pair

Example commands for reading inputs of the type shown in Figure 109:	
3R(4W,#)	**9R(4W,#)**
3*R(4W,#)	**9*R(4W,#)**
4R(4W)	**10R(4W)**
Channel pair 3&4	Channel pair 9&10

LEGEND: Kelvin sense point; Excitation wire

3-Wire Resistance Inputs

Independent 3-Wire Resistance Input on One Channel



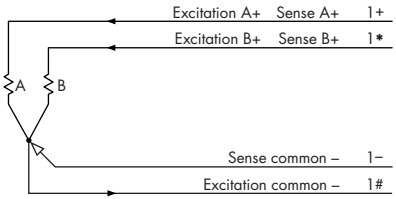
✲	#
+	–
1	

Resistance A: `1R(3W)`

FIGURE 110 Independent 3-wire resistance input on one channel

Example commands for reading inputs of the type shown in Figure 110:			
`3R(3W)`	`4R(4W)`	`12R(3W)`	`1..5R(3W)`

Shared-Terminal 3-Wire Resistance Inputs on One Channel



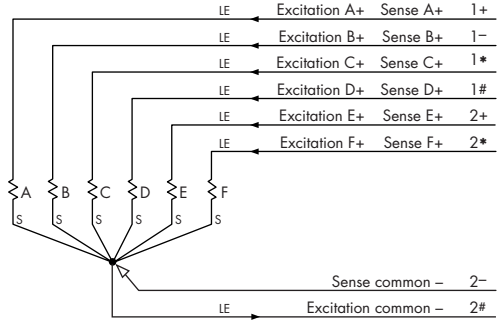
✲	#
+	–
1	

Resistance A: `1+R(3W)`
Resistance B: `1*R(3W)`

FIGURE 111 Shared-terminal 3-wire resistance inputs on one channel

Example commands for reading inputs of the type shown in Figure 111:			
`3+R(3W)`	`4*R(3W)`	`1..4+R(3W)`	`6..8*R(3W)`

Shared-Terminal 3-Wire Resistance Inputs on a Channel Pair



✲	#	✲	#
+	–	+	–
1		2	

Resistance A: `1+R(3W,#)`
Resistance B: `1–R(3W,#)`
Resistance C: `1*R(3W,#)`
Resistance D: `1#R(3W,#)`
Resistance E: `2+R(3W)`
Resistance F: `2*R(3W)`

FIGURE 112 Shared-terminal 3-wire resistance inputs on a channel pair

Example commands for reading inputs of the type shown in Figure 112:	
`3+R(3W,#)`	`9+R(3W,#)`
`3–R(3W,#)`	`9–R(3W,#)`
`3*R(3W,#)`	`9*R(3W,#)`
`3#R(3W,#)`	`9#R(3W,#)`
`4+R(3W)`	`10+R(3W)`
`4*R(3W)`	`10*R(3W)`
Channel pair 3&4	Channel pair 9&10

LEGEND

Kelvin sense point Excitation wire

Wires labelled **LE** should be the same **L**ength and gauge, and run together so that they are in the same **E**nvironmental conditions (temperature and induced noise).

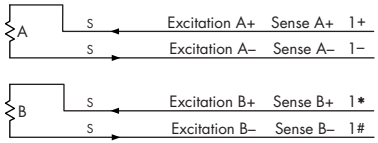
Wires labelled **S** should be as **S**hort as possible because they are included in the measured resistance.

2-Wire Resistance Inputs

Although the 2-wire configurations are the most simple, only use them if the resistance being measured is significantly greater than the resistance of the wires used to connect it to the DT800, because the resistance of the sensor wires is included in the measured resistance. In fact, we only recommend 2-wire configurations where the unknown resistance is greater than about 500 ohms.

The resistances' wires are used for both excitation and sense.

Independent 2-Wire Resistance Inputs on One Channel



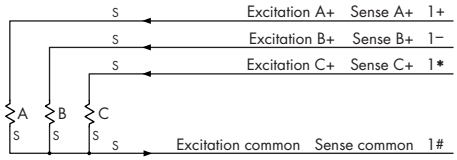
*	#
+	–
1	

Resistance A: **1R**
Resistance B: **1*R**

FIGURE 113 Independent 2-wire resistance inputs on one channel

Example commands for reading inputs of the type shown in Figure 113:			
4R	**5R**	**7R**	**12R**
4*R	**5*R**	**7*R**	**12*R**

Shared-Terminal 2-Wire Resistance Inputs on One Channel



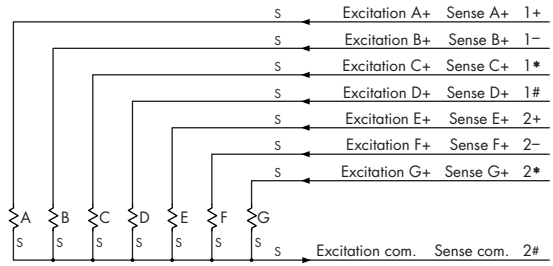
*	#
+	–
1	

Resistance A: **1+R**
Resistance B: **1–R**
Resistance C: **1*R**

FIGURE 114 Shared-terminal 2-wire resistance inputs on one channel

Shared-Terminal 2-Wire Resistance Inputs on a Channel Pair

Maximum number of 2-wire resistance inputs on a DT800: 7 per channel pair x 6 channel pairs = 42.



*	#	*	#
+	–	+	–
1		2	

Resistance A: **1+R(#)**
Resistance B: **1–R(#)**
Resistance C: **1*R(#)**
Resistance D: **1#R(#)**
Resistance E: **2+R**
Resistance F: **2–R**
Resistance G: **2*R**

FIGURE 115 Shared-terminal 2-wire resistance inputs on a channel pair

Example commands for reading inputs of the type shown in Figure 115:			
3+R(#)	**4+R**	**9+R(#)**	**10+R**
3–R(#)	**4–R**	**9–R(#)**	**10–R**
3*R(#)	**4*R**	**9*R(#)**	**10*R**
3#R(#)		**9#R(#)**	
Channel pair 3&4		Channel pair 9&10	

LEGEND

Excitation wire

Wires labelled **S** should be as **S**hort as possible because they are included in the measured resistance.

Bridge Inputs

Channel types:	BGV	voltage-excited bridge (see page 64)
	BGI	current-excited bridge (see page 64)

6-Wire BGV Inputs

Related information:

- the Bridge category in the DT800 Channel Types table (page 64)
- the Resistance and Bridge category in the DT800 Channel Options table (page 71)
- "Voltage Excitation BGV" (page 146)

6-Wire BGV Input — Single Bridge on a Channel Pair

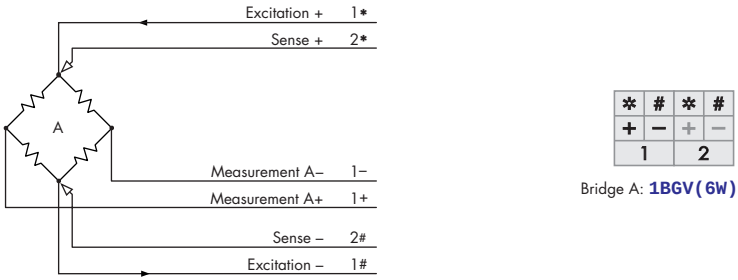


FIGURE 116 One 6-wire BGV input on a channel pair

Example commands for reading inputs of the type shown in Figure 116:			
3BGV(6W)	5BGV(6W)	7BGV(6W)	11BGV(6W)

In Figure 116, you can alternatively connect the measurement wires to the 2+ and 2– terminals (instead of 1+ and 1–). To read this configuration, the command is **2BGV(6W)**.

6-Wire BGV Inputs — Two Bridges on a Channel Pair

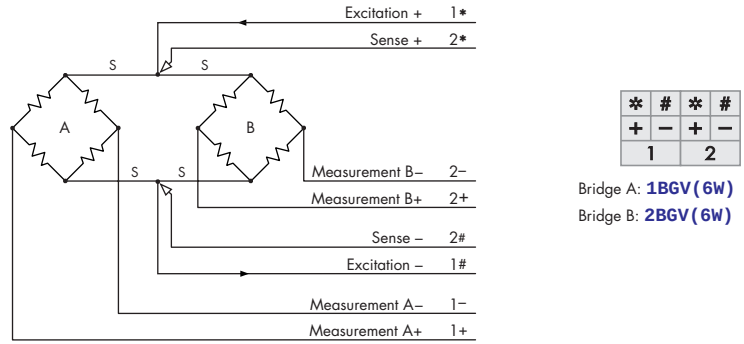


FIGURE 117 Two 6-wire BGV inputs on a channel pair

Example commands for reading inputs of the type shown in Figure 117:			
3BGV(6W)	3*BGV(6W)	11BGV(6W)	11*BGV(6W)

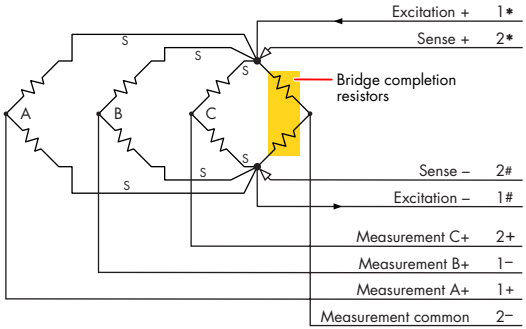
LEGEND

Kelvin sense point Excitation wire

Wires labelled **S** should be as **S**hort as possible because they are included in the measured resistance.

6-Wire BGV Inputs — Three Half-Bridges on a Channel Pair

This configuration requires the **6WC** channel option (page 71).



*	#	*	#
+	–	+	–
1		2	

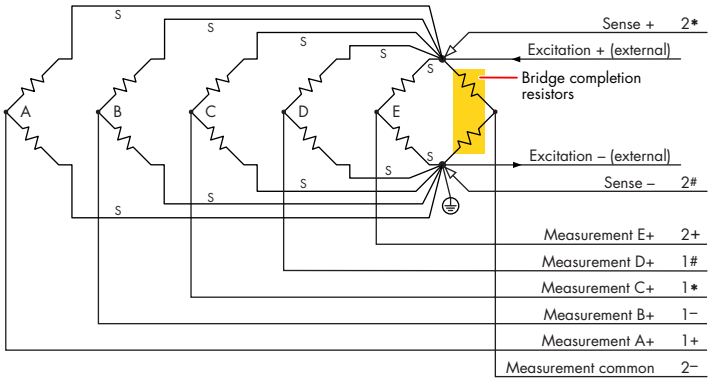
Bridge A: `1+BGV(6WC)`
Bridge B: `1–BGV(6WC)`
Bridge C: `2+BGV(6WC)`

Example commands for reading inputs of the type shown in Figure 118:	
`3+BGV(6WC)`	`9+BGV(6WC)`
`3–BGV(6WC)`	`9–BGV(6WC)`
`4+BGV(6WC)`	`10+BGV(6WC)`
Channel pair 3&4	Channel pair 9&10

FIGURE 118 6-Wire BGV measurement — three half-bridges

6-Wire BGV Inputs — Five Half-Bridges on a Channel Pair, Externally-Powered

This configuration requires the **6WC** channel option (page 71) and the **N** channel option (page 72).



*	#	*	#
+	–	+	–
1		2	

Bridge A: `1+BGV(6WC,N)`
Bridge B: `1–BGV(6WC,N)`
Bridge C: `1*BGV(6WC,N)`
Bridge D: `1#BGV(6WC,N)`
Bridge E: `2+BGV(6WC,N)`

FIGURE 119 6-Wire BGV measurement — five half-bridges, externally-powered

Example commands for reading inputs of the type shown in Figure 119:	
`3+BGV(6WC,N)`	`9+BGV(6WC,N)`
`3–BGV(6WC,N)`	`9–BGV(6WC,N)`
`3*BGV(6WC,N)`	`9*BGV(6WC,N)`
`3#BGV(6WC,N)`	`9#BGV(6WC,N)`
`4+BGV(6WC,N)`	`10+BGV(6WC,N)`
Channel pair 3&4	Channel pair 9&10

LEGEND

Kelvin sense point — Excitation wire — See "Wiring Configurations and the Ac Terminal" on page 149.

Wires labelled **S** should be as **S**hort as possible because they are included in the measured resistance.

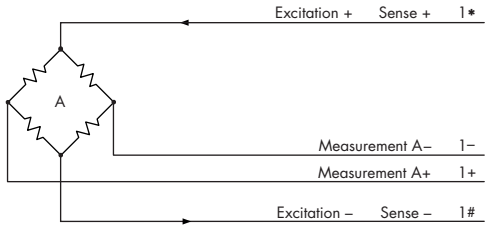
4-Wire BGV Inputs

Use voltage-excited bridge (BGV) configurations only when the bridge is close to the DT800 (BGI configurations are usually preferred).

See also

- the Bridge category in the DT800 Channel Types table (page 64)
- the Resistance and Bridge category in the DT800 Channel Options table (page 71)
- "Voltage Excitation BGV" on page 146.

4-Wire BGV Input on One Channel



*	#
+	–
1	

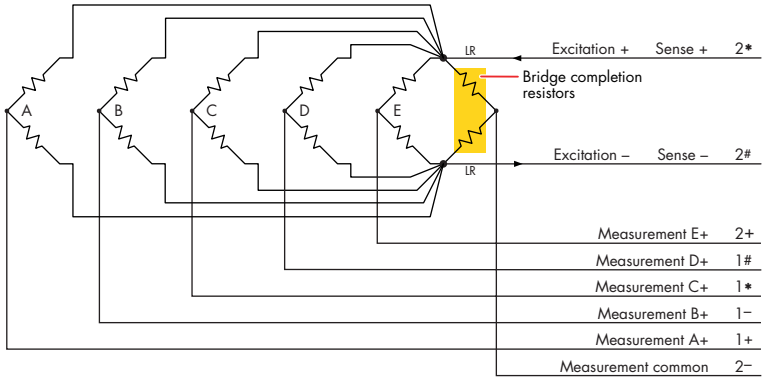
Bridge A: **1BGV(4W)**

FIGURE 120 4-wire BGV input

Example commands for reading inputs of the type shown in Figure 120:			
3BGV(4W)	**4BGV(4W)**	**7BGV(4W)**	**12BGV(4W)**

4-Wire BGV Inputs — Five Half-Bridges on a Channel Pair

The bridge voltage is calculated from measured current and user-specified resistances. This configuration requires the **4WC** channel option (page 71).



*	#	*	#
+	–	+	–
1		2	

Bridge A: **1+BGV(4WC)**
Bridge B: **1–BGV(4WC)**
Bridge C: **1*BGV(4WC)**
Bridge D: **1#BGV(4WC)**
Bridge E: **2+BGV(4WC)**

FIGURE 121 4-Wire BGV measurement — five half-bridges

Example commands for reading inputs of the type shown in Figure 121:	
3+BGV(4WC)	**9+BGV(4WC)**
3–BGV(4WC)	**9–BGV(4WC)**
3*BGV(4WC)	**9*BGV(4WC)**
3#BGV(4WC)	**9#BGV(4WC)**
4+BGV(4WC)	**10+BGV(4WC)**
Channel pair 3&4	Channel pair 9&10

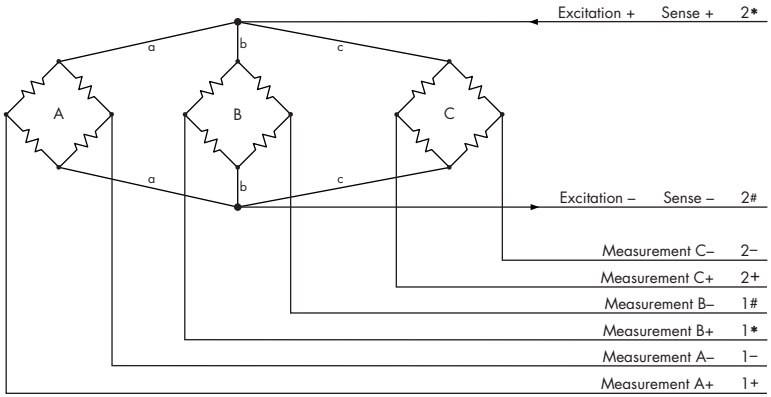
LEGEND

Excitation wire

Wires labelled **LR** should be as **L**ow **R**esistance (large diameter) as possible.

4-Wire BGV Inputs — Three Full Bridges on a Channel Pair

This configuration assumes bridges A, B and C are identical.



*	#	*	#
+	–	+	–
1		2	

Bridge A: **1BGV(4W,#)**
Bridge B: **1*BGV(4W,#)**
Bridge C: **2BGV(4W,#)**

FIGURE 122 4-wire BGV measurement — three full bridges

Example commands for reading inputs of the type shown in Figure 122:	
3BGV(4W,#)	**9BGV(4W,#)**
3*BGV(4W,#)	**9*BGV(4W,#)**
4BGV(4W,#)	**10BGV(4W,#)**
Channel pair 3&4	Channel pair 9&10

LEGEND

Excitation wire

Wires in each wire pair **aa**, **bb**,... should be the same length and gauge, and run together so that they are in the same environmental conditions (temperature and induced noise).

4-Wire BGI Inputs

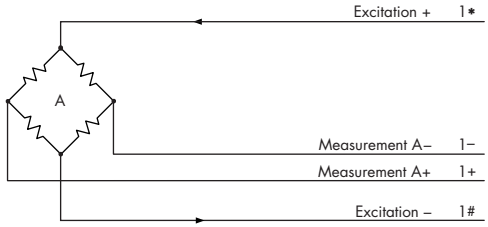
Bridge voltage calculated

We recommend the current-excited bridge (BGI) method for 4-wire bridge measurement, especially for bridges that are distant from the DT800.

See also

- the Bridge category in the DT800 Channel Types table (page 64)
- the Resistance and Bridge category in the DT800 Channel Options table (page 71)
- "Constant-Current Excitation BGI" on page 146.

4-Wire BGI Input on One Channel



*	#
+	–
1	

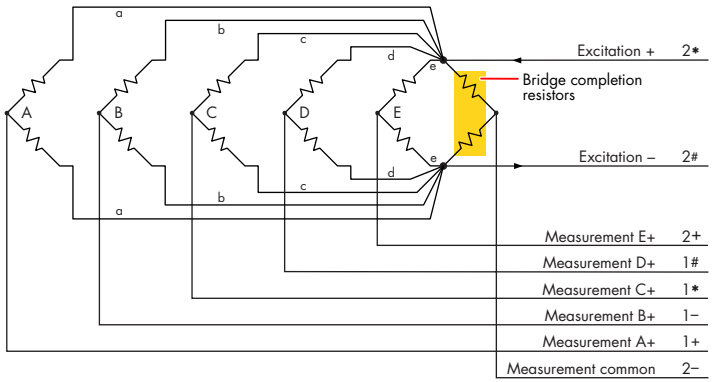
Bridge A: **1BGI**

FIGURE 123 4-wire BGI input

Example commands for reading inputs of the type shown in Figure 123:			
3BGI	**4BGI**	**7BGI**	**12BGI**

4-Wire BGI Inputs — Five Half-Bridges on a Channel Pair

This configuration requires the **4WC** channel option (page 71).



*	#	*	#
+	–	+	–
1		2	

Bridge A: **1+BGI(4WC)**
Bridge B: **1–BGI(4WC)**
Bridge C: **1*BGI(4WC)**
Bridge D: **1#BGI(4WC)**
Bridge E: **2+BGI(4WC)**

FIGURE 124 4-Wire BGI measurement — five half-bridges

Example commands for reading inputs of the type shown in Figure 124:	
3+BGI(4WC)	**9+BGI(4WC)**
3–BGI(4WC)	**9–BGI(4WC)**
3*BGI(4WC)	**9*BGI(4WC)**
3#BGI(4WC)	**9#BGI(4WC)**
4+BGI(4WC)	**10+BGI(4WC)**
Channel pair 3&4	Channel pair 9&10

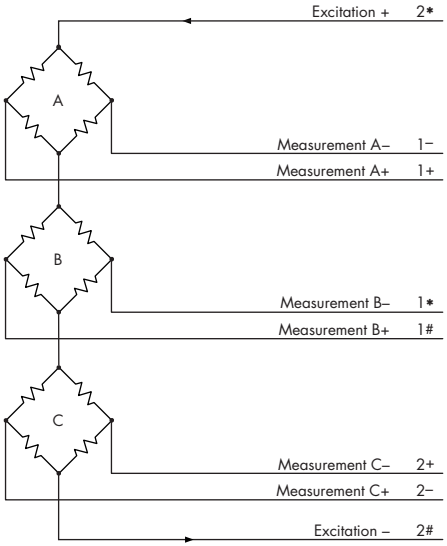
LEGEND

Excitation wire

Wires in each wire pair **aa**, **bb**,... should be the same length and gauge, and run together so that they are in the same environmental conditions (temperature and induced noise).

BGI (33) Inputs — Three Full Bridges on a Channel Pair

This configuration assumes bridges A, B and C are identical.



*	#	*	#
+	–	+	–
1		2	

Bridge A: **1BGI(33,#)**
Bridge B: **1*BGI(33,#)**
Bridge C: **2BGI(33,#)**

FIGURE 125 BGI(33) measurement — three full bridges

Example commands for reading inputs of the type shown in Figure 125:	
3BGI(33,#)	**9BGI(33,#)**
3*BGI(33,#)	**9*BGI(33,#)**
4BGI(33,#)	**10BGI(33,#)**
Channel pair 3&4	Channel pair 9&10

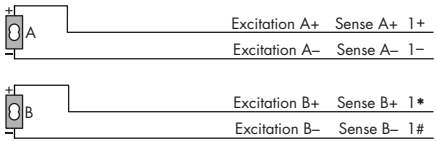
AD590-Series Inputs

IC temperature sensors for long cables (current is proportional to temperature)

Channel types: **AD590**, **AD592**, **TMP17** temperature (see page 65)

2-Wire AD590-Series Inputs

Independent 2-Wire AD590-Series Inputs on One Channel

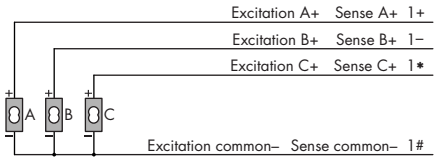


*	#
+	–
1	

Temperature sensor A: **1AD590**
Temperature sensor B: **1*AD590**

FIGURE 126 Independent 2-wire AD590-series inputs on one channel

Shared-Terminal 2-Wire AD590-Series Inputs on One Channel



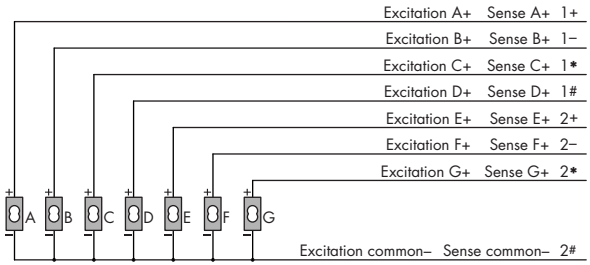
*	#
+	–
1	

Temperature sensor A: **1+AD590**
Temperature sensor B: **1–AD590**
Temperature sensor C: **1*AD590**

FIGURE 127 Shared-terminal 2-wire AD590-series inputs on one channel

Shared-Terminal 2-Wire AD590-Series Inputs on a Channel Pair

Maximum number of shared-terminal 2-wire AD590-type inputs on a DT800: 7 per channel pair x 6 channel pairs = 42.



*	#	*	#
+	–	+	–
1		2	

Temperature sensor A: **1+AD590(#)**
Temperature sensor B: **1–AD590(#)**
Temperature sensor C: **1*AD590(#)**
Temperature sensor D: **1#AD590(#)**
Temperature sensor E: **2+AD590**
Temperature sensor F: **2–AD590**
Temperature sensor G: **2*AD590**

FIGURE 128 Shared-terminal 2-wire AD590-series inputs on a channel pair

LM35-Series Inputs

IC temperature sensors for long cables (voltage is proportional to temperature)

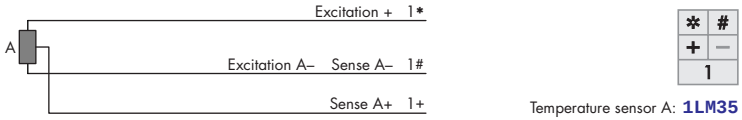
Channel types:	**LM34**, **LM35**, **LM45**, **LM50**, **LM60**, **TMP35**, **TMP36**, **TMP37**	temperature (see page 65)

Note LM34 and LM35: minimum 0 degrees

3-Wire LM35-Series Inputs

Use 3-wire configurations for short cable runs and 4-wire configurations for longer cable runs.

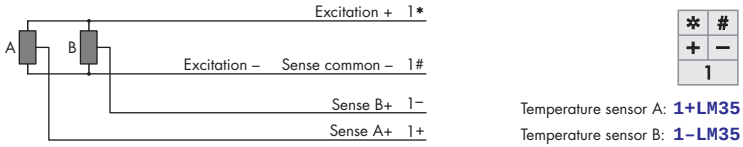
Independent 3-Wire LM35-Series Input on One Channel



Temperature sensor A: **1LM35**

FIGURE 129 Independent 3-wire LM35-series input on one channel

Shared-Terminal 3-Wire LM35-Series Inputs on One Channel

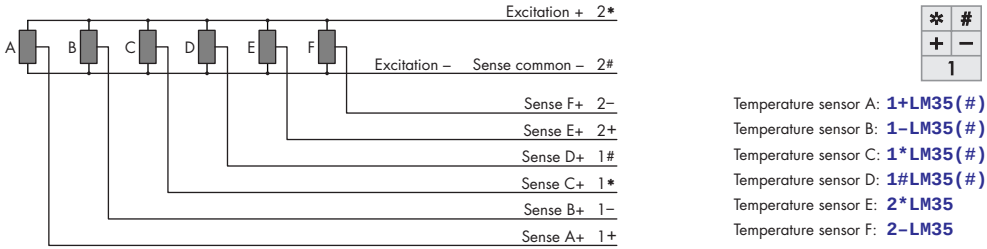


Temperature sensor A: **1+LM35**
Temperature sensor B: **1–LM35**

FIGURE 130 Shared-terminal 3-wire LM35-series inputs on one channel

Shared-Terminal 3-Wire LM35-Series Inputs on a Channel Pair

Maximum number of shared-terminal 3-wire LM35-series inputs on a DT800: 6 per channel pair x 6 channel pairs = 36.



Temperature sensor A: **1+LM35(#)**
Temperature sensor B: **1–LM35(#)**
Temperature sensor C: **1*LM35(#)**
Temperature sensor D: **1#LM35(#)**
Temperature sensor E: **2*LM35**
Temperature sensor F: **2–LM35**

FIGURE 131 Shared-Terminal 3-wire LM35-series inputs on a channel pair

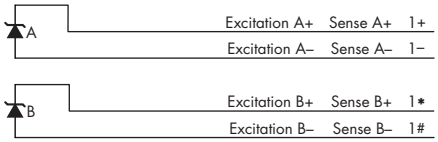
LM135-Series Inputs

IC constant-current supply (voltage is proportional to temperature)

Channel types:	LM135, LM235, LM335	temperature (see page 65)

2-Wire LM135-Series Inputs

Independent 2-Wire LM135-Series Inputs on One Channel

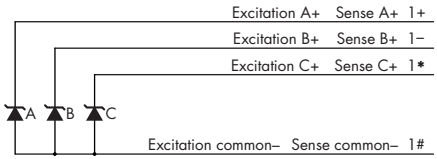


*	#
+	–
1	

Temperature sensor A: `1LM135`
Temperature sensor B: `1*LM135`

FIGURE 132 Independent 2-wire LM135-series inputs on one channel

Shared-Terminal 2-Wire LM135-Series Inputs on One Channel



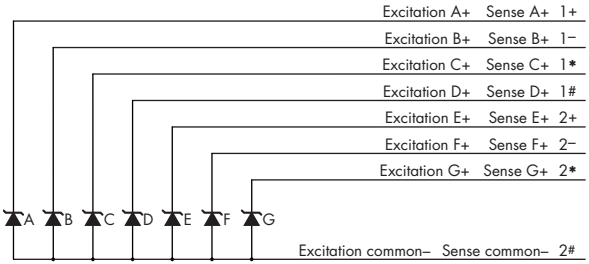
*	#
+	–
1	

Temperature sensor A: `1+LM135`
Temperature sensor B: `1–LM135`
Temperature sensor C: `1*LM135`

FIGURE 133 Shared-terminal 2-wire LM135-series inputs on one channel

Shared-Terminal 2-Wire LM135-Series Inputs on a Channel Pair

Maximum number of shared-terminal 2-wire LM135-Type inputs on a DT800: 7 per channel pair x 6 channel pairs = 42.



*	#	*	#
+	–	+	–
1		2	

Temperature sensor A: `1+LM135(#)`
Temperature sensor B: `1–LM135(#)`
Temperature sensor C: `1*LM135(#)`
Temperature sensor D: `1#LM135(#)`
Temperature sensor E: `2+LM135`
Temperature sensor F: `2–LM135`
Temperature sensor G: `2*LM135`

FIGURE 134 Shared-terminal 2-wire LM135-series inputs on a channel pair

Shielded Inputs

Input with Shield

When the sensor has a high output impedance, or when capacitively-coupled electrical noise from other cables (especially mains power cables) is a problem, we recommend using shielded sensor cable connected to the input channel's # terminal. (But note that this type of shield provides little protection from magnetically-induced noise.)

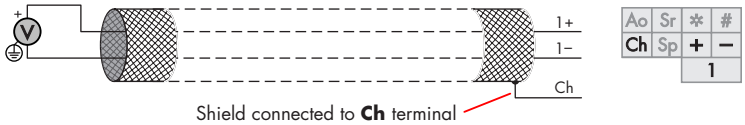


FIGURE 135 Typical input with shield

Input with Guard

When the sensor has a high output impedance and cable capacitance and insulation leakage are significant, we recommend using a guard. See "Guard Terminal" on page 149.

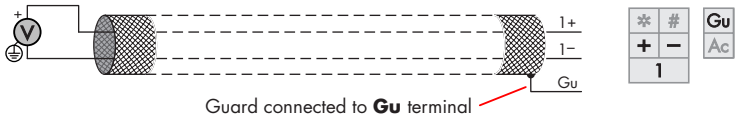


FIGURE 136 Typical input with guard

Input with Shield and Guard

Some difficult applications may require both a shield and a guard.

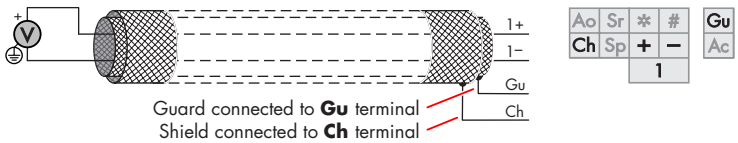
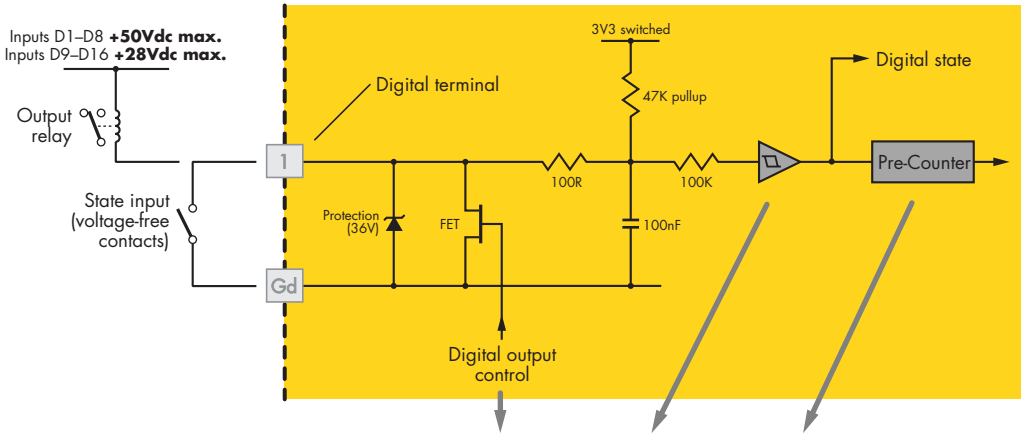


FIGURE 137 Typical input with shield and guard

Wiring Configuration — Digital Channels

Figure 138 is a simplified circuit diagram of a DT800 digital channel to assist you with wiring digital inputs and outputs.



Input	Maximum Frequency
Voltage-free contacts	50Hz
3V CMOS	1kHz

Digital Channel	Digital Output	Threshold	Hardware Pre-Counter	Counter
1–6	Yes (100mA sink)	Logic	4-bit	32-bit
7–8	Yes (100mA sink)	Logic or 10mV	12-bit	32-bit
9–16	No	Logic	None	

Figure 138 DT800 digital channel

Part K — Reference

Command Summaries

Summary — Delete Commands

The following table summarizes all the clear/delete/erase commands supported by the DT800:

	Command	Action	
Job	**DELJOB**	Deletes the current job from the DT800	When a job is deleted, all traces of it (its name, directory structure, program, data and alarms) are erased from the *dataTaker*.
	DELJOB"*JobName*"	Deletes only ***JobName*** from the DT800	
	DELJOB*	Deletes all jobs from the DT800	
Data	**DELDATA**	Deletes the current job's data from the DT800	Job names, directory structures, programs and alarms are not erased.
	DELDATA"*JobName*"	Deletes only ***JobName***'s data from the DT800	
	DELDATA*	Deletes all jobs' data from the DT800	
Alarms	**DELALARMS**	Deletes the current job's alarms from the DT800	Job names, directory structures, programs and data are not erased.
	DELALARMS"*JobName*"	Deletes only ***JobName***'s alarms from the DT800	
	DELALARMS*	Deletes all jobs' alarms from the DT800	
Event Log	**CEVTLOG**	Clears the event log	
PC Card	**CARDCLEAR**	Formats PC Card (deletes all data and alarms, and copies the DT800's current internal directory structure to card)	
Attention LED	**CATTN**	Clears the Attention LED (see page 29)	
User Defaults	**DELUSERINI**	Deletes the working copy of USER.INI from the DT800's SRAM memory	
	DELONRESET	Deletes the working copy of ONRESET.DXC from the DT800's SRAM memory	
	DELFLASHUSERINI	Deletes the backup copy of USER.INI from the DT800's Flash memory	
	DELFLASHONRESET	Deletes the backup copy of ONRESET.DXC from the DT800's Flash memory	
	DELONINSERT	Deletes ONINSERT.DXC from the DT800-specific directory on the card	
	DELONINSERTALL	Deletes ONINSERT.DXC from the root directory on the card	
See also the DT800 Resets table on page 118.			

Table: DT800 Delete Commands — Summary

Summary — Retrieval Commands

The following table summarizes all the information retrieval (unload) commands supported by the DT800 — program, data, alarm and event retrieval.

	Command	Action	
Program	SHOWPROG	Copies the current job's program file to the host port	
	SHOWPROG"*JobName*"	Copies *JobName*'s program file to the host port	
	SHOWPROG*	Copies all currently-defined jobs' program files to the host port	
Data U	U	Returns the current job's data in the order of report schedule A to K	
	U*x*	Returns the current job's data for report schedule *x*	
	U"*JobName*"	Returns data for *JobName* in the order of report schedule A to K	
	U"*JobName*"*x*	Returns data for *JobName* report schedule *x*	
Data U( )	U	Returns the current job's data in the order of report schedule A to K	Type *date* and *time* in formats that match your DT800's current setup (P31 and P39). If a schedule isn't specified, data is always unloaded schedule-by-schedule. **Note** BEGIN and END used here are not the same as the BEGIN and END keywords used to indicate the start and end of a DT800 job.
	U(*from*)	Returns the current job's data starting from BEGIN, *time* or *time*,*date*	
	U(*from*)(*to*)	Returns the current job's data starting from BEGIN, *time* or *time*,*date* and ending with END, *time* or *time*,*date*	
	U*x*	Returns the current job's data for report schedule *x*	
	U*x*(*from*)	Returns the current job's data for schedule *x* starting from BEGIN, *time* or *time*,*date*	
	U*x*(*from*)(*to*)	Returns the current job's data for schedule *x* starting from BEGIN, *time* or *time*,*date* and ending with END, *time* or *time*,*date*	
	U"*JobName*"	Returns data for *JobName* in the order of report schedule A to K	
	U"*JobName*"(*from*)	Returns data for *JobName* starting from BEGIN, *time* or *time*,*date*	
	U"*JobName*"(*from*)(*to*)	Returns data for *JobName* starting from BEGIN, *time* or *time*,*date* and ending with END, *time* or *time*,*date*	
	U"*JobName*"*x*	Returns data for *JobName* report schedule *x*	
	U"*JobName*"*x*(*from*)	Returns data for *JobName* report schedule *x* starting from BEGIN, *time* or *time*,*date*	
	U"*JobName*"*x*(*from*)(*to*)	Returns data for *JobName* report schedule *x* starting from BEGIN, *time* or *time*,*date* and ending with END, *time* or *time*,*date*	
Data U[ ]	U	Returns the current job's data in the order of report schedule A to K	Type *date* and *time* in the DT800's fixed-format style (ISO format; see "Examples — U[ ] Commands" on page 84). If a schedule isn't specified, data is always unloaded schedule-by-schedule.
	U[*from*]	Returns the current job's data starting from *time* or *time*,*date*	
	U[*from*][*to*]	Returns the current job's data starting from *time* or *time*,*date* and ending with *time* or *time*,*date*	
	U*x*	Returns the current job's data for report schedule *x*	
	U*x*[*from*]	Returns the current job's data for schedule *x* starting from *time* or *time*,*date*	
	U*x*[*from*][*to*]	Returns the current job's data for schedule *x* starting from *time* or *time*,*date* and ending with *time* or *time*,*date*	
	U"*JobName*"	Returns data for *JobName* in the order of report schedule A to K	
	U"*JobName*"[*from*]	Returns data for *JobName* starting from *time* or *time*,*date*	
	U"*JobName*"[*from*][*to*]	Returns data for *JobName* starting from *time* or *time*,*date* and ending with *time* or *time*,*date*	
	U"*JobName*"*x*	Returns data for *JobName* report schedule *x*	
	U"*JobName*"*x*[*from*]	Returns data for *JobName* report schedule *x* starting from *time* or *time*,*date*	
	U"*JobName*"*x*[*from*][*to*]	Returns data for *JobName* report schedule *x* starting from *time* or *time*,*date* and ending with *time* or *time*,*date*	

Table: DT800 Retrieval Commands — Summary (sheet 1 of 2)

Alarms **A**	**A**	Returns the current job's alarms in the order of report schedule A to K	
	Ax	Returns the current job's alarms for report schedule ***x***	
	A"*JobName*"	Returns alarms for ***JobName*** in the order of report schedule A to K	
	A"*JobName*"*x*	Returns alarms for ***JobName*** report schedule ***x***	
Alarms **A( )**	**A**	Returns the current job's alarms in the order of report schedule A to K	Type ***date*** and ***time*** in formats that match your DT800's current setup (P31 and P39). If a schedule isn't specified, alarms are always unloaded schedule-by-schedule. **Note** **BEGIN** and **END** used here are not the same as the **BEGIN** and **END** keywords used to indicate the start and end of a DT800 job.
	A(*from*)	Returns the current job's alarms starting from **BEGIN**, ***time*** or ***time,date***	
	A(*from*)(*to*)	Returns the current job's alarms starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***	
	Ax	Returns the current job's alarms for report schedule ***x***	
	Ax(*from*)	Returns the current job's alarms for schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date***	
	Ax(*from*)(*to*)	Returns the current job's alarms for schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***	
	A"*JobName*"	Returns alarms for ***JobName*** in the order of report schedule A to K	
	A"*JobName*"(*from*)	Returns alarms for ***JobName*** starting from **BEGIN**, ***time*** or ***time,date***	
	A"*JobName*"(*from*)(*to*)	Returns alarms for ***JobName*** starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***	
	A"*JobName*"*x*	Returns alarms for ***JobName*** report schedule ***x***	
	A"*JobName*"*x*(*from*)	Returns alarms for ***JobName*** report schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date***	
	A"*JobName*"*x*(*from*)(*to*)	Returns alarms for ***JobName*** report schedule ***x*** starting from **BEGIN**, ***time*** or ***time,date*** and ending with **END**, ***time*** or ***time,date***	
Alarms **A[ ]**	**A**	Returns the current job's alarms in the order of report schedule A to K	Type ***date*** and ***time*** in the DT800's fixed-format style (ISO format; see "Examples — A[ ] Unload Command" on page 106). If a schedule isn't specified, alarms are always unloaded schedule-by-schedule.
	A[*from*]	Returns the current job's alarms starting from ***time*** or ***time,date***	
	A[*from*][*to*]	Returns the current job's alarms starting from ***time*** or ***time,date*** and ending with ***time*** or ***time,date***	
	Ax	Returns the current job's alarms for report schedule ***x***	
	Ax[*from*]	Returns the current job's alarms for schedule ***x*** starting from ***date*** or ***date,time***	
	Ax[*from*][*to*]	Returns the current job's alarms for schedule ***x*** starting from ***date*** or ***date,time*** and ending with ***date*** or ***date,time***	
	A"*JobName*"	Returns alarms for ***JobName*** in the order of report schedule A to K	
	A"*JobName*"[*from*]	Returns alarms for ***JobName*** starting from ***date*** or ***date,time***	
	A"*JobName*"[*from*][*to*]	Returns alarms for ***JobName*** starting from ***date*** or ***date,time*** and ending with ***date*** or ***date,time***	
	A"*JobName*"*x*	Returns alarms for ***JobName*** report schedule ***x***	
	A"*JobName*"*x*[*from*]	Returns alarms for ***JobName*** report schedule ***x*** starting from ***date*** or ***date,time***	
	A"*JobName*"*x*[*from*][*to*]	Returns alarms for ***JobName*** report schedule ***x*** starting from ***date*** or ***date,time*** and ending with ***date*** or ***date,time***	
Event Log	**UEVTLOG**	Returns the contents of the event log	

Table: DT800 Retrieval Commands — Summary (sheet 2 of 2)

Getting Optimal Speed from Your DT800

Although the DT800's sampling engine — its ADC — is capable of taking measurements at 100kHz, the actual speed achievable depends on the following factors:

Speed Factor — ADC Settings

The current settings of the DT800's ADC (calibration, settling time, sampling time,...) can be adjusted using various channel options, parameters and switches. See "Analog-to-Digital Conversion" on page 19. The default ADC settings are suitable for most sensors and we recommend that you don't alter these unless you're an experienced DT800 user.

Speed Factor — Channel Type (Fundamental Samples)

Some channel types require the DT800 to take more fundamental samples to arrive at a reading than others do. See "Number of Fundamental Samples per Reading" on page 63.

For example, to obtain a reading, a channel type that requires four fundamental samples (4FS) per reading takes twice as long as a channel type that requires only two fundamental samples (2FS) per reading. In other words, the maximum scan rate achievable when using a 2FS channel type is double what's achievable for a 4FS channel type[20]. See the Sampling Speeds table on page 51.

Speed Factor — Number of Channels

The more channels you define, the slower each channel is sampled. See the Sampling Speeds table on page 51.

Speed Factor — Data Storage and Return Options

Formatting data for storage or return consumes processor time and consequently reduces sampling speeds.

Speed Factor — Data Formatting

Minimizing the number of significant digits in the stored and returned data frees processor time, which results in increased sampling speed.

Speed versus Accuracy

By default, the DT800 automatically applies zero correction (autozeroing) to all measurement channels — excluding **VNC** (page 63) and internal maintenance channels (page 67) — to remove any input zero drift voltage that might affect the precision of your readings. These drifts can occur

- if the DT800 is exposed to significant temperature change during measurement, or
- through component aging (the DT800 is thoroughly "burnt-in" during manufacture to minimize this effect — see "Characterization" on page 123).

This automatic zero correction is achieved by taking additional zero measurements interleaved with the channel measurement. The result is greatly improved zero stability and common-mode signal rejection. But this comes at the cost of speed as the additional readings take time.

Where speed is of prime importance use the **VNC** channel type. **VNC** is the most fundamental measurement type — it is fast with no zero correction and is most appropriate when measuring fast dynamic signals.

[20] Such scan rate comparisons — based solely on the number of FS (fundamental samples) per reading for individual channel types — are approximate. This is because the DT800 takes FSs in binary multiples; that is, in groups of 1, 2, 4, 8,... FS called **frames**. So a channel type that requires, say, 5FS per reading actually uses a frame of 8FS (the next binary multiple above 5). See "Maximum Scan Rate" on page 63.

Best Speed in Normal Mode

You can adjust one or more of the settings in the table below to increase the DT800's normal-mode sampling speed. However, be aware that such optimization generally compromises the accuracy of the DT800. The settings are listed in the order in which we recommend you apply them.

P11	Mains frequency	Increase P11 to increase speed
P46	Number of samples of each channel per integral number of mains cycles	Reduce P46 to increase speed
/r	Return data	Turn data returns off
GL*n*	Gain lock	Use gain lock **GL*x*** to prevent autoranging
P60	Maximum sampling frequency	Increase P60 to increase speed
Channel Types	Use the most efficient channel type. See "Number of Fundamental Samples per Reading" on page 63.	
P57	Number of frame capture cycles	Reduce P57 to 1 for maximum speed (using gain lock with no excitation has the same effect)
P58	Excitation adjustment tolerance	Increase P58 (reduce tolerance) to increase speed

Best Speed in Burst Mode

You can adjust one or more of the settings in the table below to increase the DT800's burst-mode sampling speed. In burst mode, accuracy is not compromised but the DT800 becomes more susceptible to noise. The settings are listed in the order in which we recommend you apply them.

P48	Default burst sampling frequency	Increase P48 to increase speed
/b	Burst data return	Turn burst data returns off
Channel Types	Use the most efficient channel type. See "Number of Fundamental Samples per Reading" on page 63.	
Channels Per Burst	Reduce the number of channels included in the burst. If it suits your application, use separate bursts of (ideally) just one channel each, rather than one burst modifier containing several channels. See "Bursts and DT800 Resources" on page 53.	
P54	Burst timeout	Reduce P54 to reduce time between bursts

ASCII-Decimal Table

Special characters only

Decimal	ASCII	Control	Description
0	NUL		null
1	SOH	^A	
2	STX	^B	
3	EXT	^C	
4	EOT	^D	
5	ENQ	^E	
6	ACK	^F	acknowledge
7	BEL	^G	bell
8	BS	^H	backspace
9	HT	^I	tab
10	LF	^J	line feed
11	VT	^K	vertical tab
12	FF	^L	form feed
13	CR	^M	carriage return
14	SO	^N	
15	SI	^O	
16	DLE	^P	
17	DC1	^Q	XON
18	DC2	^R	
19	DC3	^S	XOFF
20	DC4	^T	
21	NAK	^U	not acknowledge
22	SYN	^V	
23	ETB	^W	
24	CAN	^X	
25	EM	^Y	
26	SUB	^Z	
27	ESC	^[	escape
28	FS	^|	
29	GS	^]	
30	RS	^^	
31	US	^_	

Decimal	ASCII	Control	Description
32			space
33	!		
34	"		
35	#		
36	$		
37	%		
38	&		
39	'		
40	(		
41	)		
42	*		
43	+		
44	,		comma
45	–		
46	.		period
47	/		
48	0		
49	1		
50	2		
51	3		
52	4		
53	5		
54	6		
55	7		
56	8		
57	9		
58	:		colon
59	;		semicolon
60	<		
61	=		
62	>		
63	?		

Decimal	ASCII	Control	Description
64	@		
91	[		
92	\		
93	]		
94	^		
95	_		underline
96	`		
98	b		"
123	{		
124	|		
125	}		
126	~		
127	DEL		delete

Table: ASCII Characters

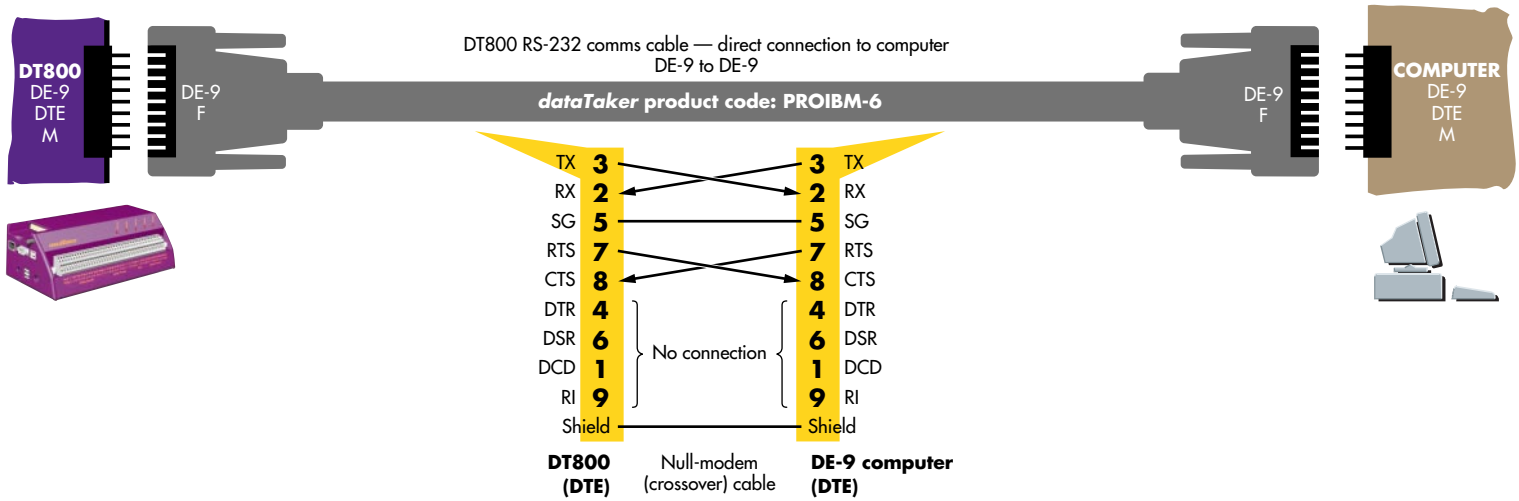
RS-232 Standard

The following table lists the standard RS-232 pinouts for both 9-pin (DE-9) and 25-pin (DB-25) DCE and DTE interfaces as used in Figure 139 (page 191) and Figure 140 (page 192)).

Signal/Function		DTE or DCE (DT800, Computer or Modem)	
		DE-9 Pin	DB-25 Pin
DCD	Data Carrier Detect	1	8
RX	Receive	2	3
TX	Transmit	3	2
DTR	Data Terminal Ready	4	20
SG	Signal Ground	5	7
DSR	Data Set Ready	6	6
RTS	Request To Send	7	4
CTS	Clear To Send	8	5
RI	Ring Indicator	9	22
			The remaining DB-25 pins are not required for RS-232 communication.

Table: RS-232 Pinouts

Cable Details



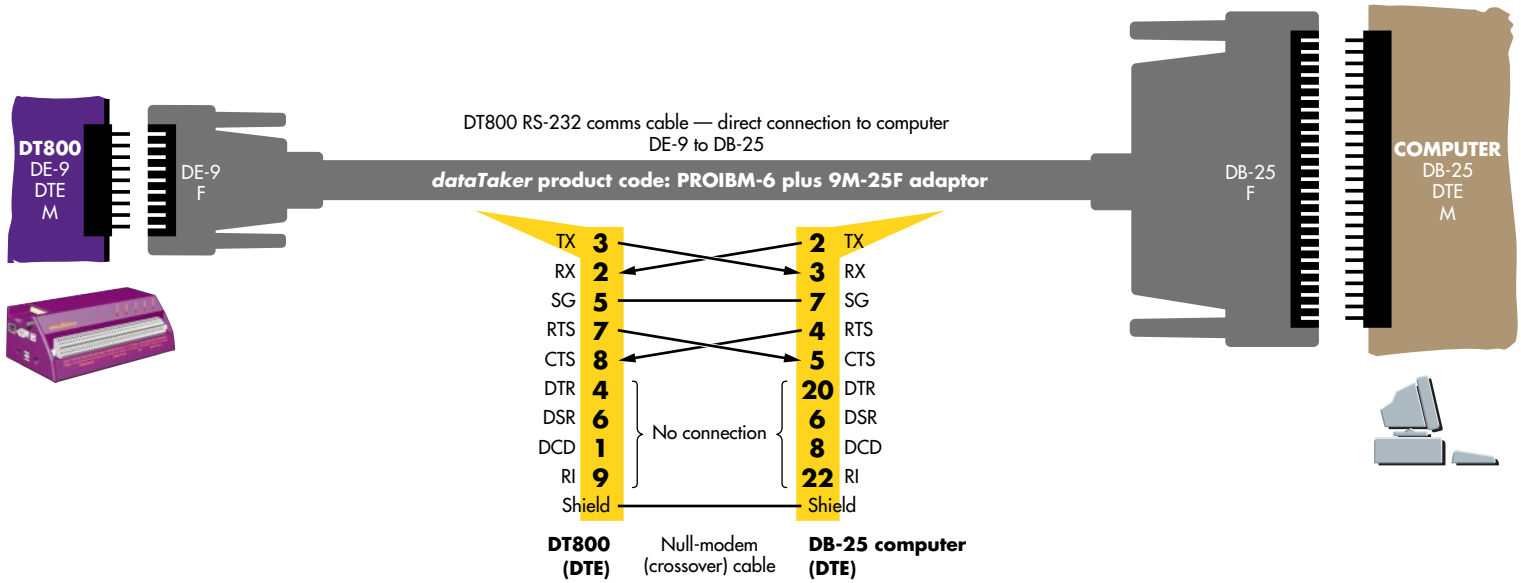


Figure 139 DT800-to-computer RS-232 comms cable — DE-9 computer (upper diagram) and DB-25 computer (lower diagram)

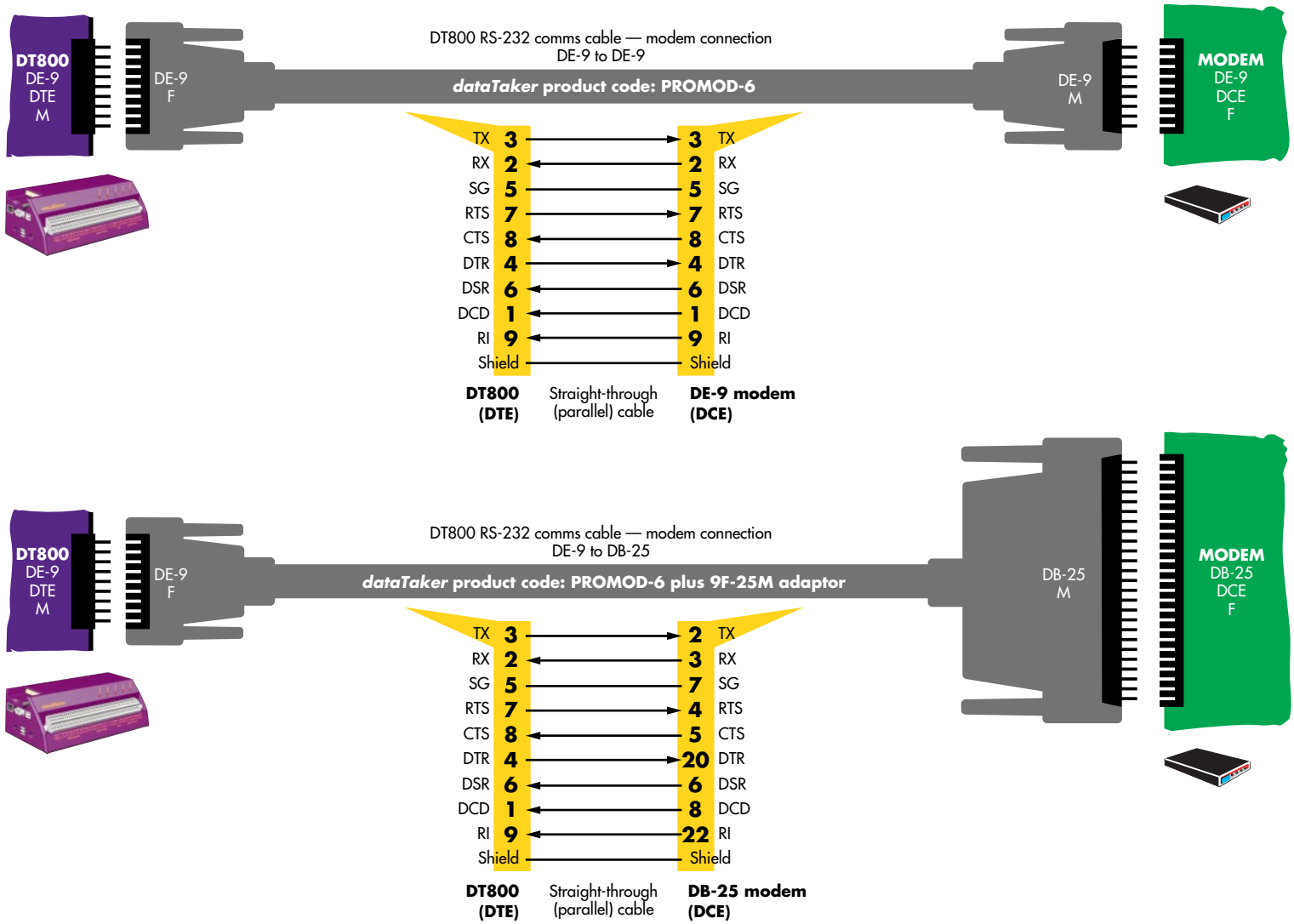


FIGURE 140 DT800-to-modem RS-232 comms cable — DE-9 modem (upper diagram) and DB-25 modem (lower diagram)

Upgrading DT800 Firmware

Operating system upgrade

The DT800's operating system is stored as "firmware" (see page 206) in the DT800's Flash memory (see "Memory" on page 30). This means that you can easily upgrade your DT800's operating system from a host computer running DeLogger (or DeTransfer).

An upgrade takes between 2 and 15 minutes (depends on the method used) and is completely safe. (In case of a problem occurring during an upgrade, the DT800 automatically protects its firmware upgrade "bootstrap" file, which always allows you to restart the process.)

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 DT800's firmware to ensure that no compatibility problems arise.

This procedure returns the DT800 to a completely unprogrammed state. Once the upgrade is complete you will have to restore any settings and programs.

1. Connect to the DT800 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.

   Warning if you are using DeLogger, it may ask if you want to upgrade the DT800 when you connect to the DT800. If this occurs, answer **No** so that you can perform the following steps before the upgrade occurs.

2. Save any previously logged data from internal memory and from any PC card to be used with the new version of firmware.

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 all data for all jobs on the DT800 using the **DELDATA** command.

5. Delete all jobs on the DT800 using the **DELJOB** command.

6. Delete any ONRESET job stored in flash memory using the **DELFLASHONRESET** command.

7. Delete any USER.INI stored in flash memory using the **DELFLASHUSERINI** command.

8. Format any cards to be used with the DT800 using the **FORMAT"A:"** command.

9. Format the internal RAM disk using the **FORMAT"B:"** command.

10. Carry out the firmware upgrade as described in "Firmware Upgrade — Host RS-232 Port" or "Firmware Upgrade — PC Card" below.

    When the upgrade is done, you'll need to reconnect to the DT800 and set up your required settings and programs.

Recommendation — Power Before carrying out a firmware upgrade, we recommend that you charge the DT800's main internal battery for 12 hours. Furthermore, if at all possible, power the DT800 from an external source as well during the upgrade. These two precautions minimize the possibility of a power failure to the DT800 during the upgrade.

Important These instructions are for upgrading a DT800 whose firmware is version 2.00.000000 or later[21]. If its firmware is earlier than this (that is, version 1.xx.xxxxxx), contact your *dataTaker* distributor for specific upgrade instructions.

Two common methods of upgrading a DT800's firmware are described below:

- From a computer connected directly to the Host RS-232 port of the DT800. See "Firmware Upgrade — Host RS-232 Port" below.
- From a PC Card that you've prepared using DeTransfer software: simply insert the card into the DT800 and wait. See "Firmware Upgrade — PC Card" below.

There are also other ways you can upgrade your DT800's firmware (remotely by FTP transfer, for example). If you have a special requirement such as this contact your *dataTaker* representative.

Firmware Upgrade — Host RS-232 Port

Here's the procedure for upgrading the firmware of a DT800 by using DeLogger (version 2 revision 16 or later) on a computer directly connected to the DT800's Host RS-232 port.

Warning If you attempt to make a connection to a DT800, DeLogger checks the firmware version of the DT800 against the latest version that it can see in the C:\Program Files\DeLogger\Firmware\mt 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 DT800 (as described in "Recommended Preparation" above) before allowing either this automatic upgrade to occur, or when 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. Unzip the files in the zip file and place them where the host computer can access them on the computer's hard disk or a network drive.

   Placing them in the directory **C:\Program Files\DeLogger\Firmware\mt** is a good idea because DeLogger looks in this directory by default.

   The firmware upgrade file is named according to its firmware version number and has a .DXF extension — for example, **4000001.DXF**. The size of a typical .DXF file is 500 to 800kB zipped and 1 to 2MB unzipped.

   If you have a choice of upgrade files, always use the latest (the one with the highest number) unless you've been directed to do otherwise.

2. Connect the comms cable supplied with your DT800 between the DT800's Host RS-232 port and either of the computer's COM1 or COM2 serial ports.

3. Power the DT800 as described in "**Recommendation — Power**" above.

4. Start DeLogger and check that you're using version 2 revision 16 or later (from DeLogger's **Help** menu, choose **About DeLogger...**).

   DeLogger is included on the CD supplied with each DT800, or available as a free download from **www.datataker.com**.

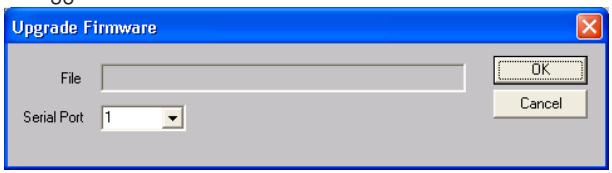
   You can also use DeTransfer version 3.18 or later.

5. In DeLogger, choose **Upgrade Firmware** from the **dataTaker** menu. In DeTransfer, choose **Upgrade Firmware(DT80/800)** from the **File** menu.

[21] Find out the version of your DT800's firmware by connecting to the DT800 and sending the **TEST0** command (that's **TEST** followed by a zero). The DT800 returns a line of information that contains the version number.

6 In the dialog box that opens (Figure 141), select the serial port that will be used to access the DT800 during the upgrade and then click **OK**.

DeLogger



DeTransfer

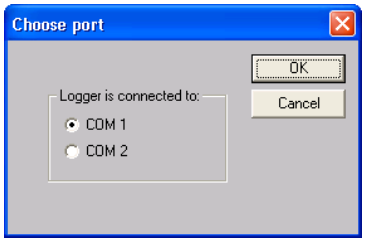


FIGURE 141 Serial port selection dialog boxes

7 In the navigation dialog box that opens, locate the firmware upgrade file (**4000001.dxf**, for example), highlight it (one click) and click **Open** (Figure 142. DeLogger looks in the directory C:\Program Files\DeLogger\Firmware\mt by default. You can select files in other directories using this dialog box if necessary.

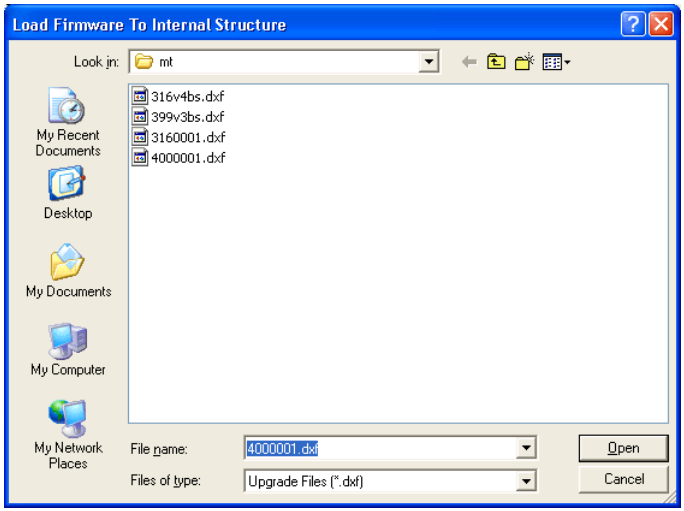


FIGURE 142 Load Firmware... dialog box

8 Follow the on-screen instructions.
During the upgrade, do not remove any cables, or reset or power-down the DT800.

9 Wait until the upgrade is complete (may take up to 15 minutes; depends mainly on the host computer).
Successful completion is indicated by
- the DT800 going through a firm reset, which causes its front-panel LEDs to carry out their standard startup sequence (see "LED Sequence on Startup" on page 28)
- DeLogger (or DeTransfer) presenting you with an "upgrade successful" message.

These indications occur at the same time.

10 To confirm that the DT800 is now running the new firmware, make a normal software connection to the DT800, send `TEST0` and check that the version number returned now ends in the name of the firmware upgrade file you used (`4.00.0010`, for example).

11 While connected to the DT800, configure it with your preferred settings and programs.
The upgraded DT800 is now ready for use.

Firmware Upgrade — PC Card

Here's the procedure for upgrading the firmware of a DT800 by inserting a PC Card (ATA Flash type) into the DT800.

You must firstly prepare the PC Card by using DeTransfer to load the firmware upgrade file onto the card correctly (by splitting it into three component files: **Loader.S**, **Firmware.BIN** and **ONINSERT.DXC**). Your computer must have a PC Card drive, or network access to one. Once the PC Card is prepared, you can use it to upgrade the firmware of one or many DT800s.

1 Carry out step **1** of "Firmware Upgrade — Host RS-232 Port" to obtain the firmware upgrade file.

2 Carry out step **4** of "Firmware Upgrade — Host RS-232 Port" to check that you have a suitable version of DeTransfer.

3 Prepare the PC Card as follows:

a) Insert an ATA Flash PC Card into the computer's PC Card drive.
There needs to be about 1.5MB of free space on the card. Existing firmware upgrade files at the card's root directory will be overwritten, but all other files on the card will be left untouched.

b) From DeTransfer's **File** menu, choose **Prepare Firmware for PCMCIA Upgrade (DT80/800)**.

c) In the Load Firmware to Internal Structure dialog box that opens, locate the firmware upgrade file (**4000001.dxf**, for example), highlight it (one click) and click **Open**.

d) In the Save Loader To File dialog box that opens (Figure 143), navigate to the root directory of the PC Card (that is, the card must be selected/visible in the **Save in** list box as shown in Figure 143), then click **Save**.
The **Loader.S** file is extracted from the firmware upgrade file (.dxf) and saved to the root directory of the card.

Warning Some versions of DeTransfer incorrectly have **Save as type** set to **S Record Files (*.dwn)** at this point. In this case, select **All files (*.*)** from the **Save as type** drop-down list, and ensure that file is called **Loader.S** in the **File name** field.

e) In the Save Firmware To File dialog box that opens (Figure 144), check that the card is still present in the **Save in** list box, then click **Save**.
The **Firmware.BIN** file is extracted from the firmware upgrade file and saved to the root directory of the card.

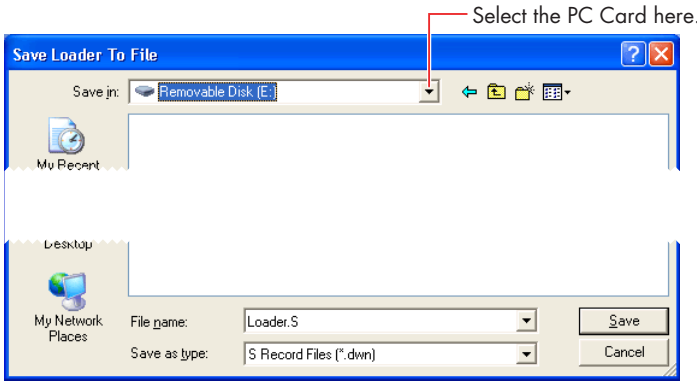


FIGURE 143 Saving the Loader.S component of the .dxf file to the card

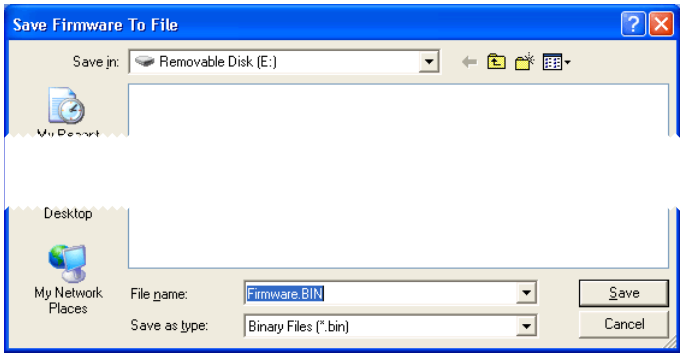


FIGURE 144 Saving the Firmware.BIN component of the .dxf file to the card

f) In the Save ONINSERT.DXC Commands dialog box that opens (Figure 145), check that the card is still present in the **Save in** list box, then click **Save**.
The **ONINSERT.DXC** component of the firmware upgrade file is saved to the root directory of the card.

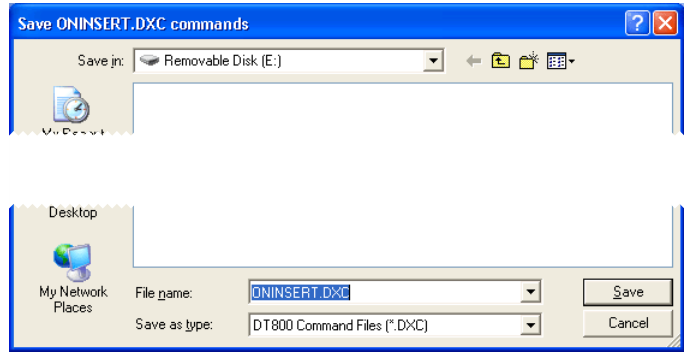


FIGURE 145 Saving the ONINSERT.DXC component of the .dxf file to the card

g) Remove the card from the drive.
The card is ready. It will now automatically upgrade the firmware of any DT800 it is inserted into.

4 Use the prepared card to upgrade a DT800's firmware as follows:

a) Power the DT800 as described in "**Recommendation — Power**" on page 193.

b) Insert the prepared card into the DT800.
You don't need to halt or reset the DT800 first. Any logged data is transferred to the card (this is normal card insertion procedure; see "Retrieving Logged Data — Memory Card Transfer" on page 81), then the firmware upgrade process starts automatically. During the upgrade, do not remove any cables, or reset or power-down the DT800.

c) Wait until the upgrade is complete (will take 2 to 3 minutes).
Successful completion is indicated by
- the DT800 going through a firm reset, which causes its front-panel LEDs to carry out their standard startup sequence (see "LED Sequence on Startup" on page 28)
- DeTransfer presenting you with an "upgrade successful" message.

These indications occur at the same time.

d) Remove the card from the DT800.
Warning Every time you insert this card into a DT800, it will automatically upgrade the DT800's firmware.

5 If you want the DT800 to continue operating, start a job using the **RUNJOB"*JobName*"** command (see page 16).

Confirmation of Card Upgrade

When a DT800 resets at the end of a firmware upgrade from a card, it writes status information to the card. This information includes the DT800's serial number and firmware version.

In this way, the card accumulates a history of each DT800 it was inserted into along with all data and firmware versions for each DT800. Later, using a computer with a PC Card drive, you can retrieve the upgrade history from the card as follows:

a) Using a text editor such as Windows' Notepad, open the EVENT.LOG file (in the card's EVENTS directory).

b) Look for lines that contain a DT800 serial number (such as `080315`) along with the message `"Reset x.xx.xxxxxx"` (where `x.xx.xxxxxx` is the firmware version number).

Upgrade Mode

In case you ever need to use **manual recovery** to put the DT800 into **upgrade mode** (described in separate documentation), here is the procedure:

1 Open the DT800's case (described on page 31).

2 Place the top half of the DT800 upside-down in front of you as in Figure 146.

FIGURE 146 Ready for jumper insertion

3 Install a jumper to short out the bootstrap pin header located next to the AA 3.6V Lithium battery on the power supply PCB (Figure 147).
Note that some older versions of the DT800 do not have this pin header, in which case you must use a jumper wire to connect pin **4A** to pin **6A** on the right-hand DIN connector as shown in Figure 148.
Use of the pin header is preferred because it avoids accidentally connecting the wrong signals on the DIN connector.

4 While watching the front-panel LEDs, press the DT800's hardware reset button once (see Figure 12 on page 25).

5 Wait for the front-panel LEDs to flash, then remove the jumper (or jumper wire).

6 Double-check that you've removed the jumper wire. (It's easy to forget this.)

7 Close the DT800's case (described on page 38).
The DT800 is now in upgrade mode, which is covered in other *dataTaker* documentation.

FIGURE 147 Location of bootstrap pin header

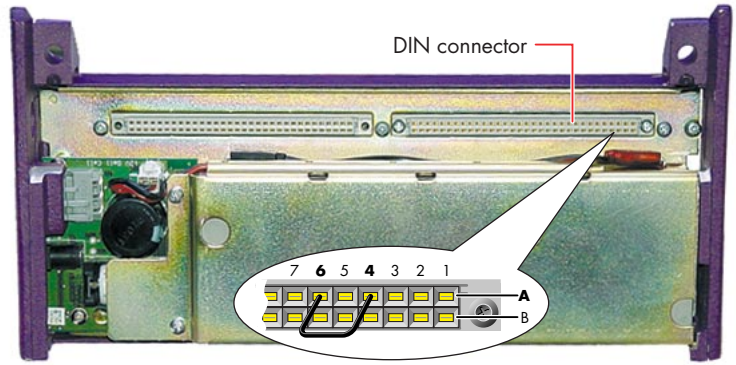


FIGURE 148 Inserting an upgrade jumper — older versions only

Error Messages

The DT800 returns a message when it detects an error in a command, an error in an input channel, 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 a data retrieve operation, error reporting is disabled until the unload is complete (see "Fixed-Format Mode /H" on page 21).

Errors that are a result of reading a channel will cause a value of 99999.9 to be returned or logged as the reading. This value is not modifiable by format channel options.

Errors that occur as a result of reading an alarm channel are reported in the same way as data channels, and the alarm condition automatically becomes true.

The table below lists all of the DT800 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 – Input buffer full • Command too long (maximum 250 characters) • Successive commands input too quickly (with no flow control)		✓	✓	✓	
E3 – Channel option error • Illegal channel option used • Mutually exclusive options used	✓				
E4 – Clear data memory • Attempt to enter new schedule while the store contains data, or LOGON is enabled		✓	✓		
E5 – Data memory full • Internal data storage memory is full • Overwrite switch not enabled (/O)		✓	✓		
E6 – Data memory empty • No data in internal or card memory		✓			
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 • Missing switch command character • Illegal switch command character	✓				
E10 – Command error • Unrecognized keyword	✓				
E11 – Input(s) out of range • One or more analog inputs is over range • Check common-mode voltage				✓	✓

Table: DT800 Error Messages (sheet 1 of 6)

Error Number and Description • Cause/Action	Error Category				
	Syntax	Operation	Memory	Reading	Hardware
E12 – Channel list error • Channel number outside the legal range • Options invalid for channel type • Incomplete channel sequence • Invalid channel type • Shared channels illegally specified • Polynomials or spans specified for day or time • Polynomials or spans index out of range	✓				
E13 – Digital failure • Digital input — output circuit has failed • Return *dataTaker* for service				✓	✓
E14 – Communications error • Baud rate, parity or stop bit errors • Framing errors due to noise on Host RS-232 port		✓			
E15 – Assignment error • Channel number too large • Output channel or system variable out of range • Counter preset to value greater than maximum count — that is, **1C(25)=30**	✓				
E16 – Linearization error(s) • Thermocouple outside range • RTD or thermistor outside linearization range				✓	✓
E17 – Clear card data • Inserted card has data in data area		✓	✓		
E18 – STATUS command error • STATUS incorrectly entered • STATUSn outside the range 1 to 14	✓				
E19 – Card write-protected • Move card write-protect switch to unprotected	✓		✓		
E20 – Illegal character(s) • Invalid characters in the command					
E21 – Illegal separator(s) • Commands not separated by spaces or return	✓				
E22 – Statistical option error • Statistical option not in each multiple report	✓				
E23 – Scan schedule error • Schedule ID not A, B, C, D, E, F, G, H, I, J, K, X or S • Scan time interval too large (>65535) • Scan interval type invalid • Event or counter channels invalid	✓				
E24 – Unload command error • Schedule ID is not one of A, B, C, D, E, F, G, H, I, J, K, Z or X (not S or immediate)	✓				
E25 – Channel table full • Internal acquisition and alarm table filled (maximum 500 entries) • Additional channels cannot be declared		✓	✓		
E26 – Halt command error • Schedule ID not A, B, C, D, E, F, G, H, I, J, K (not S or immediate)	✓				

Table: DT800 Error Messages (sheet 2 of 6)

Error Number and Description • Cause/Action	Error Category				
	Syntax	Operation	Memory	Reading	Hardware
E27 – TEST command error • TEST incorrectly entered • TESTn where n is outside the range 0 to 40	✓				
E28 – Go command error • Schedule ID not A, B, C, D, E, F, G, H, I, J, K	✓				
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 PCB not found • Faulty circuit board, circuit board connector, or circuit board power supply • Contact your *dataTaker* representative.					✓
E31 – File system error • Can't find expected file		✓	✓		✓
E32 – Job not found • The named job cannot be found.	✓				
E33 – File system not initialized • Unable to initialize the DT800 file system (SRAM or other failure) • Contact your *dataTaker* representative.			✓		✓
E34 – Burst complete • Indicates successful completion of burst (not an error)	Message only; not an error				
E35 – Card faulty • Memory card may have an electrical fault			✓		✓
E36 – Directory not found • The named directory cannot be found.	✓	✓			
E37 – No current job • There is no current job in the DT800.	✓				
E38 – File not found • Named file cannot be found • Drive letter or path incorrect • File does not exist	✓				
E39 – Channel list fixed /F • Changes to program are not allowed		✓			
E40 – No data found • No logged data to unload in specified time interval	✓				
E41 – Program area full • Program area on memory card full		✓	✓		
E42 – No card inserted • No memory card inserted into card socket • Memory card not fully inserted • Memory card battery discharged • Memory card failure		✓	✓		✓
E43 – Serial sensor chip failure • RS-485 interface hardware has failed • Return *dataTaker* for service					✓
E44 – Job locked or has data – program ignored until END • Unlock job first • Clear job's data/alarms first		✓			

Table: DT800 Error Messages (sheet 3 of 6)

Error Number and Description • Cause/Action	Error Category				
	Syntax	Operation	Memory	Reading	Hardware
E46 – Flash checksum error • Flash has failed checksum test • DT800 may behave strangely • Return *dataTaker* for service			✓		✓
E47 – User string error • Incorrect declaration $="text" or n$="text"	✓				
E48 – Channel list fixed /F – program ignored until END • Program changes not allowed		✓			
E50 – Job locked or has data • Unlock job and delete its data and alarms first		✓			
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 alarms text is full (total is 16384 characters, shared with expressions text) • Cannot specify additional alarm strings		✓	✓		
E53 – No statistical samples • No statistical sample taken so cannot calculate statistical function	✓			✓	
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) • Reduce number of expressions		✓	✓		
E57 – Unable to create job • No internal memory available			✓		
E58 – Job name error • JobName is too long (max. eight characters, no spaces) • JobName contains non-alphanumeric characters (#, *, [, &,... not allowed)	✓				
E60 – SRAM failure • RAM number n has failed self test • May cause strange behaviour and data loss • Return *dataTaker* for service			✓		✓
E61 – PC Card memory failure • Replace memory card battery • Replace memory card (memory card cannot be serviced, except for battery replacement)			✓		✓
E62 – Flash (startup) program password error • Flash program is password-protected • Password mismatch	✓	✓			
E63 – Flash program command error • Syntax error	✓	✓			
E64 – Invalid channel in BURST • See "Burst Mode Sampling" (page 53)	✓				
E65 – ALARM undefined • Alarm n does not exist	✓				

Table: DT800 Error Messages (sheet 4 of 6)

Error Number and Description • Cause/Action	Error Category				
	Syntax	Operation	Memory	Reading	Hardware
E70 – Bad sensor port number • Serial sensor: can only be one port at present	✓				
E71 – Output buffer overflow • Serial sensor: attempting to construct an output prompt longer than 250 characters		✓			
E72 – Scan buffer overflow • Serial sensor: attempting to scan for an input string longer than 50 characters		✓			
E73 – \nnn too big • Serial sensor: special character code not between 1 and 255	✓				
E74 – CTS detect timeout • Serial sensor: CTS timed out		✓			
E75 – Transmit timeout • Serial sensor: transmit confirm timed out		✓			
E76 – Invalid output format • Serial sensor: bad %..type construction	✓				
E77 – Invalid output type • Serial sensor: unsupported %..type (that is, not %f, etc.)	✓				
E78 – Expecting %x[args] • Serial sensor: bad %..type construction	✓				
E79 – Expecting %x[nCV] • Serial sensor: expecting [nCV] after %..type	✓				
E80 – nCV out of range • Serial sensor: n of nCV must be in the range 1 to 500	✓				
E81 – Expecting] after %x[args • Serial sensor: too many [nCV] after %..type	✓				
E82 – Invalid %c width • Serial sensor: width must be 1 for %c	✓				
E83 – Invalid scan %x format • Serial sensor: bad %..type construction	✓				
E84 – Expecting scan %x[args] • Serial sensor: bad %..type construction	✓				
E85 – Integer too big • Serial sensor: must be in the range 0 to 65535	✓				
E86 – Invalid CTS format • Serial sensor: bad \C construction	✓				
E87 – Invalid RTS format • Serial sensor: bad \R construction	✓				
E88 – Invalid \W format • Serial sensor: bad \W construction	✓				
E89 – Receive time out • Serial sensor: expected characters were not received within timeout period		✓		✓	
E90 – } expected to end timeout • Serial sensor: not expecting closing } in input block	✓				
E91 – { not expected here • Serial sensor: not expecting opening { in output block	✓				
E92 – Bad end of \nnn • Serial sensor: expecting definition after \	✓				

Wiring p168 Schedules p44 Channel Types p63 Channel Options p71 Alarms p96 Parameters p107 Switches p111 Error Messages p197

Error Number and Description • Cause/Action	Error Category Syntax	Operation	Memory	Reading	Hardware
`E93 – Bad break definition` • Serial sensor: no specification of number of character periods	✓				
`E94 – Null control string` • Serial sensor: out of buffer space to allocate to control string • Reset DT800	✓				
`E95 – Bad match definition` • Serial sensor: bad \M construction	✓				
`E96 – n$ out of range` • Serial sensor: n of n$ must be in range 1 to 10	✓				
`E97 – Expecting %S[n$]` • Serial sensor: expecting [n$] after %S	✓				
`E98 – Flash writing error` • Flash memory faulty • Return DT800 for service					✓
`E99 – Flash reading error` • Flash memory faulty • Return DT800 for service					✓
`E100 – Drive to format not specified` • Specify the drive to be formatted (for example, A: or B:).	✓				
`E101 – Drive letter not valid` • Can only format drive A: or B:	✓				
`E102 – ':' expected after drive letter` • Must specify **:** after the drive letter (for example, A:)	✓				
`E103 – Extra stuff after ':' not needed` • Do not specify any directories or files when only a drive letter is required.	✓				
`E104 – Drive format failed` • Unable to format the specified drive. It may be write-protected or damaged.			✓		
`E105 – Command invalid between BEGIN..END` • The RUNJOBONRESET command is not allowed as part of a job definition.	✓				
`E106 – Pn(s) in USER.INI out of range` • An out-of-range parameter has been specified in the USER.INI file. It must be changed for correct operation.	✓				
`E107 – Counter is already used as a trigger` • The specified counter is already used as a trigger and cannot be used again.		✓			
`E108 – System too busy – try again` • The systems needs to sleep for a short period to complete an operation, but is unable to do so (because the system is too busy). This can happen if P15=2 or an Ethernet cable is plugged in.		✓			
`E109 – File IO error: detailed description` • An error occurred while reading or writing to a file on one of the drives (A:, B: or C:). The detailed description will contain exact details of the error type. For example, a write-protected file cannot be written to.			✓		
`E110 – Internal File IO error: detailed description` • An internal error occurred while reading or writing to a file on one of the drives. This error should not normally occur.		✓			✓
`E111 – Channel table full! Extra channels ignored.` • The maximum number of channels across all schedules for one job that can be defined is 500. Any more channels will be ignored.			✓		
`E112 – Insufficient space for job on PC card` • Card does not have enough free space. • Create more space on the memory card then try again, or use another memory card.			✓		

Table: DT800 Error Messages (sheet 6 of 6)

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. DT800s 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

µA
microamp

µs
microsecond

µStrain
microstrain

µV
microvolt

acquisition
See “data acquisition system” on page 205.

actuator
A device that converts a voltage or current input into a mechanical output.

ADC
Analog-to-Digital Converter. Part of the DT800’s input circuitry that converts an analog input voltage to a digital number (in other words, it converts a smoothly-varying signal to a quantized digital value). The DT800 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” on page 204.

Ah
Ampere-hours

analog
Analog channels are designed for information 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. Appendere (Latin) means “to hang”.

ASCII
American Standard Code for Information Interchange. A coding system designed for standardizing data transmission to achieve hardware and software compatibility. It assigns an 8-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. See “ASCII-Decimal Table” beginning on page 189.

asterisk (*)
Also called **splat** or **star**

asynchronous
Not synchronized, 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**. See “Speed versus Accuracy” on page 188.

base date and time
The DT800’s base date is 01/01/89. This is "day zero”.

If you’ve set the DT800’s clock/calendar, it counts time from midnight (00:00am) on 01/01/89, or from midnight on the current day — see **P39** on page 109. If you haven’t set the clock/calendar, any time value included with your data is the interval since the DT800 was powered up.

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 • group terms within expressions.
[ ]	Square brackets	Used to • enclose channels and commands to be conditionally executed within alarms and IFs — for example, **[1CV=1CV+1 HB]** • enclose channel variable to be used within a serial prompt or parse command — for example, **%f[17CV]** • indicate simple grouping (that is, that the contents of the square brackets are to be considered as a single entity and not separate items) — for example, **[CR]** indicates that **CR** is the single control character **carriage return** and not two uppercase characters **C** and **R**.
{ }	Braces	Used to • signify user-definable optional settings. For example, **V{*n*}** signifies that the voltage excitation channel option V is user-definable: you can specify your own value by replacing **{*n*}** with a value like **2000** so that **V{*n*}** becomes **V2000**, or you can omit **{*n*}** to have the DT800 use its default (that is, **V{*n*}** becomes **V**). • signify output actions in the serial sensor command (see page 161).
{[ ]}	Sequence	Used to signify action commands/processes in **ALARM** and **IF** statements. See "Alarm Action Text" on page 99 and "Conditional Processing — IF… Command" on page 60.

Operators:

< >	Angle brackets	Used for • testing alarm conditions • conditional expressions — see "Conditional Calculations" on page 94.

bps

bits per second

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" on page 146.

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 DT800's default is to suffix each scan's data with a carriage return (see page 108). 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 40 (page 44).

channel ID

A channel's number and type. See Figure 40 (page 44).

channel list

A list of channel definitions, each separated by a space. See "Channel List" (page 45).

channel settling time

The time allowed for the input signal to the ADC to stabilize before it is measured.

clock

The DT800 has two clocks:

- its real-time clock/calendar, which you can set to your actual time
- a clock that controls the speed of the DT800's ADC, and therefore controls burst speed

CMRR

Common-Mode Rejection Ratio. A measure of the influence of common-mode voltage (unwanted) on the output of the DT800's instrumentation amplifier (see "common-mode voltage" 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.

$$\text{CMRR} = 20 \log\left(\frac{V_{CM}}{V_{out} \times A_V}\right)$$

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 file

A DT800 program. See "program" on page 208.

command line

One or more DT800 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 DT800 commands (separated by spaces).

A DT800 schedule command is a particular instance of a command line.

In word processing, a return character signifies the end of a paragraph; for the DT800, a return character not only defines the end of a command line but is also the signal to the DT800 to process the command line.

common-mode voltage

An unwanted AC and/or DC voltage that offsets both inputs to the DT800'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 DT800'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**.

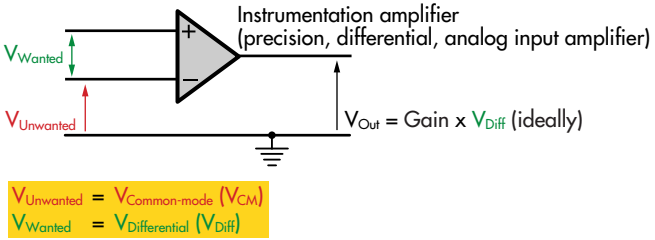
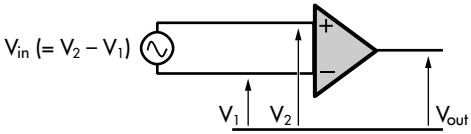


FIGURE 149 Common-mode voltage V_{CM} and Differential voltage (V_{Diff})

When the DT800 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" on page 204.
- 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" on page 205.

Common-mode voltage is calculated as the average of the voltages between the measurement system's ground and the two input terminals:



$$V_{CM} = \frac{V_1 + V_2}{2}$$

FIGURE 150 Common-mode voltage V_{CM} and Differential voltage (V_{Diff})

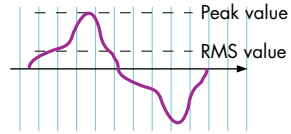
CR

See "carriage return" on page 204.

crest factor

The peak-to-RMS voltage ratio of an AC signal (Figure 151).

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.



$$\text{Crest factor} = \frac{\text{Peak value}}{\text{RMS value}}$$

FIGURE 151 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" on page 207.

data logging system

A data acquisition system with its own on-board data storage and manipulation facilities. See also "logging" on page 207.

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

DCE

Data Communications Equipment. The controlling device in an RS-232 communications link. A DCE device (a modem, for example) establishes, maintains and terminates the RS-232 conversation, enabling a DTE device (such as a computer or a DT800 data *dataTaker*) to communicate over phone lines or data circuits. A DCE device connects to the RS-232 interface of a DTE device. See "DTE" on page 206.

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 DT800 automatically reverts after a reset.

differential voltage

The difference between the voltages on the two inputs of the DT800's instrumentation amplifier (the *dataTaker*'s precision, differential, analog input amplifier). See "common-mode voltage"on page 205.

digital

Digital channels are designed for information that has a finite number of values. For the DataTaker, computers and other electronic equipment this number of values is two; for example, the states of a switch (on/off) or pulses from a digital counter (high/low or 1/0). Compare with **analog**.

direct commands

Commands that perform direct tasks within the DT800 the moment they are sent (for example, switch, parameter, unload, alarm, job and delete commands).

directory
Another name for **folder**

DSD
DataTaker S-record download format. The format of the DT800's firmware upgrade file. See also "S-Record" on page 210.

DTE
Data Terminal Equipment. The information source and/or destination in an RS-232 communications link. The DT800's Host RS-232 port and Serial Channel are DTE devices, as is every PC's RS-232 port (serial port).

Usually, an RS-232 communications channel has a DTE device at one end (typically a computer — or more precisely, the computer's UART chip) and a DCE device at the other end (for example, a modem) — see "DCE" on page 205. But two DTE devices can also be connected directly together for successful RS-232 communication. This is the primary way that the DT800 (a DTE device) communicates with its host computer (also a DTE device). A null-modem cable is required between the two DTE devices — see "null-modem cable" on page 208 and "DT800 Direct (Local) RS-232 Connection" on page 129.

echo
A communications option for commands you send to the DT800. When echo is turned on (by the echo switch **/E**), commands you send to the DT800 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 DT800 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" below). Retains its contents even when power is unavailable. See also "Memory" on page 30.

enable
Turn on or activate

EPROM
Erasable Programmable Read-Only Memory. A special type of PROM that can be erased by exposing it to ultraviolet light. Retains its contents even when power is unavailable.

Ethernet
A LAN (local area network) technology developed by Xerox, Intel and DEC, and subsequently adopted by the IEEE (Institute of Electrical and Electronics Engineers) as a standard. The DT800 supports 10BaseT Ethernet, which uses a twisted-pair cable with 100 meters maximum length between nodes and operates at a data transfer rate of 10Mbps. Ethernet is the most common type of LAN used with the TCP/IP protocol.

firmware
Software stored inside the *dataTaker* (its operating system, for example). The DT800's firmware is semi-permanent, and you can upgrade it with a simple file transfer (instead of having to change hardware inside the *dataTaker*).

Flash
A special type of EEPROM that can be erased and reprogrammed in blocks (instead of one byte at a time — compare with "EEPROM" above). Flash memory is therefore much faster to erase and re-write. Retains its contents even when power is unavailable. The DT800's firmware is stored in Flash memory. See also "Memory" on page 30 and "Upgrading DT800 Firmware" on page 193.

flow control
The process of controlling the flow of information between communications devices. For example, if data is being sent too quickly from a DT800 to its host computer, the computer tells the DT800 to temporarily stop sending data; then when the computer has caught up, it tells the DT800 to resume sending data. See "Flow Control" on page 127. Hardware handshaking (hardware flow control; RTS/CTS) and software handshaking (software flow control; XON/XOFF) are alternative mechanisms of flow control.

The DT800's **DO...** command performs another type of flow control; program flow control. See "Unconditional Processing — DO... Command" on page 61.

folder
Another name for **directory**

format
A specific organisation of related information

frame
A binary multiple (1, 2, 4, 8,...) of fundamental samples. See "Maximum Scan Rate" on page 63.

FTP
File Transfer Protocol. A TCP/IP program, service and protocol for copying files from one computer to another.

fundamental sample
Normal mode and fast mode For any one channel, the average of the background measurements the DT800 takes during a mains cycle. See "Normal Mode Sampling" (page 51), and Figure 51 (page 51).

Burst mode The basic measurements that the DT800 takes then uses to derive each burst reading. See "Burst Mode Sampling" (page 53), and Figure 55 (page 56).

See also "reading" on page 209.

Gregorian calendar
The reformed Julian calendar now in use. A Gregorian year consists of 365 days, and a leap year of 366 days occurs in every year whose number is exactly divisible by four (except centenary years whose numbers are not exactly divisible by 400, such as 1700, 1800 and 1900).

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 DT800 has been designed for maximum immunity to ground loops — see "DT800 Analog Sub-System" on page 148.

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. See "Input with Guard" on page 183.

hash (#)
Also called **pound** or **number sign** (or **crosshatch**, **hatch**, **number**, **sharp** or **octothorpe**)

heartbeat
The brief flash that the Acquiring LED makes every three seconds while the DT800 is awake. See step 3 on page 28.

host computer
The computer you use for supervising the DT800

host software
The software you run on the host computer to supervise the DT800. See "DT800-Friendly Software" on page 10.

hot plugging
Adding (or removing) devices to the DT800's USB while it is running and having the DT800's operating system *automatically* recognize the change.

hunting
An undesirable oscillation

HWFC
Hardware flow control (RTS/CTS). Also known as **hardware handshaking**. See "flow control" on page 206.

Hz
Hertz. A unit of frequency (or change in state) of one cycle per second. Replaces the term **cycles per second** (cps).

instrumentation amplifier
A precision differential amplifier for amplifying the DT800's analog input signals (wanted) and rejecting any common-mode voltage (unwanted). See Figure 3 (page 13) and Figure 92 (page 148).

IP address
A device's address on an Ethernet network. Every device connected to an Ethernet network must be assigned its own unique IP Number.

ISO
International Organization for Standardization

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 Figure 8 on page 17 and "Jobs" on page 15.

Julian calendar
The calendar established by Julius Caesar in 46BC, which fixed the length of the year at 365 days, with 366 days in every fourth year (leap year), and months similar to today's calendar (Gregorian — see page 206).

kb
kilobit

kB
kilobyte

kbps
kilobits per second

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 DT800 has five LEDs on the front panel, which light to indicate Card Busy, Attention, Acquiring, Logging and Charging. See "LEDs" on page 28 for more details.

log
Verb: to store or record information
Noun: a store or record containing information

logging
Recording or storing data. The DT800 logs data to its internal memory and/or an external memory card. Logging is a separate, user-configurable operation that the DT800 performs in addition to its basic function of data acquisition (taking measurements from sensors connected to its inputs). See also "data logging system" on page 205 and "data acquisition system" on page 205.

lsb
least significant bit

LSB
Least Significant Byte

mΩ
milliohm

mA
milliamp

Mb
megabit

MB
megabyte

Mbps
Megabits per second

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" on page 145), pressure, acceleration and concentration of various compounds in gases and liquids.

ms
milliseconds

msb
most significant bit

MSB
Most Significant Byte

multidrop
In communications, a multidrop configuration allows multiple nodes to be connected in parallel by means of a single twisted-pair cable. This requires tristate outputs on each of the nodes.

multiplexer
A "many-in, one-out" switching network that allows many input signals to time-share one analog input circuit. It sequentially routs multiple channels to a single signal processing system. The DT800 has six input multiplexers that service all of its analog input terminals (see "Channel Pairs" on page 12).

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 DT800 to a PC) — see Figure 139 on page 191. Also known as a **crossover cable**.

number sign (#)
See "hash (#)" on page 207.

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

PC Card
A memory card that conforms to the PCMCIA (Personal Computer Memory Card International Association) standard. The DT800 uses Type I, II or III PCMCIA memory cards (ATA Flash type).

peak-to-peak
The value of an alternating quantity measured from its negative peak to its positive peak.

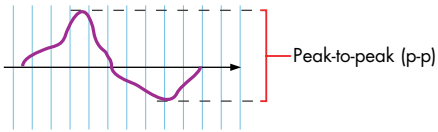


FIGURE 152 Peak-to-peak measurement

period
The time taken for a cyclic event to repeat itself. Reciprocal of frequency:

$$\text{Period} = \frac{1}{\text{Frequency}}$$

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 DT800 using the difference, integration and calculation facilities.

PLC
Programmable Logic Controller. Used to automate monitoring and control of industrial equipment.

plug-and-play
The ability of the DT800's USB to automatically configure any devices connected to it. You plug a device into the USB and immediately play with it without having to configure it yourself.

polling
Requesting information

port
A plug, socket or interface that enables connection to another device for information transfer — in other words, a communications connector. Ports can be physical (like the RS-232 connector on a DT800), or logical (like the FILE port you use when printing to file from Windows programs).

pound (#)
See "hash (#)" on page 207.

PowerPC
Performance Optimization With Enhanced RISC. A RISC-based computer architecture developed jointly by IBM, Apple Computer and Motorola Corporation.

PPP
Point-to-Point Protocol. A protocol that provides powerful serial line connectivity between two computers, or between a computer and a network. Because the DT800 supports PPP, it can use this to communicate with a remote computer over a serial internet link (that is, using a modem connection to the internet).

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, a **BURST** command and an **IF** command.

program
A DT800 program is a group of one or more jobs that you send to the DT800 all at once, so that they are all stored in the DT800. (If you send the jobs one-by-one, each new job replaces the previous one so that there is only ever one job in the DT800.) See "Jobs" beginning on page 15.

PROM
Programmable Read-Only Memory. A memory chip on which you can store information only once (when a PROM has been programmed, it cannot be erased and re-written). Retains its contents even when power is turned off.

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" on page 210.

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 and is the reason why the DT800 uses this type of memory (in particular, see "SRAM" on page 210) to process and store data, alarms, events, the current program, and clock/calendar and other settings. Often simply called **memory**.
See "Memory" on page 30.

RAM disk
An area of RAM configured by a software program to emulate a disk drive.

reading
The final channel value returned by the DT800. To arrive at a single channel reading in normal mode or fast mode, the DT800

- takes suitable background measurements (up to 80 per mains cycle), then
- averages the appropriate background measurements over one mains cycle to produce the fundamental sample value(s) required by the channel type (for example, the **V** channel type requires four fundamental samples — a zero (offset) reading taken on the channel's **+** terminal, a differential reading across the **+** and **–** terminals, another zero (offset) reading on the **–** terminal, then another differential reading across the **+** and **–** terminals), then
- uses the fundamental samples to compute the final reading (for example, for the **V** channel type, the DT800 subtracts the sum of the two offset fundamental samples from the sum of the two differential fundamental samples and divides the result by two to arrive at the final reading).

See also "Number of Fundamental Samples per Reading" (page 63) and "Normal Mode Sampling" (page 51).

real-time
As it happens. The DT800 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 (or, to be more precise, within a split-second).

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.
The DT800's16-bit ADC can resolve to1 part in 65,536 (2^{16}=65,536).

retrieve
To unload or return data and other information from the DT800 to the host computer:

- Unloading through one of the DT800's communications interfaces does not remove data from internal memory or memory card.
- Unloading by temporarily inserting a memory card into the DT800 then removing the card and reading this in a computer does remove data from internal memory.

See "Retrieving Logged Data" on page 81.

return
See "carriage return" on page 204.

RISC
Reduced Instruction Set Computer. A type of microprocessor that recognizes a relatively limited number of instructions (which makes it very fast).

ROM
Read Only Memory. Memory that can be randomly read from but not written to. The DT800 uses electrically-erasable programmable ROMs — see "EEPROM" on page 206 and "Memory" on page 30.

RS-232
Recommended Standard 232: a common communications and interface standard for connecting serial devices. See also "DCE" on page 205.

RTD
Resistance Temperature Detector. A resistive sensor that changes resistance with changes in temperature. See "RTDs" on page 145.

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 stabilize) 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 (see previous entry) 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 DT800 most often. There are several flavors of schedule command — see "Schedules" beginning on page 44.

schedule header
The schedule's ID and trigger — see Figure 40 (page 44).

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 stabilize after the DT800 selects the channel, selects the gain, and applies excitation (if required). See "channel settling time" on page 204.

shared-terminal inputs
See "Shared-Terminal Analog Inputs" on page 13.

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. See "Input with Shield" on page 183.

splat (*)
Also called **asterisk** or **star**

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 "Memory" on page 30.

S-Record
A Motorola file format. A printable ASCII format consisting of an optional header record, and one or more data records followed by an end of file record. Data records may appear in any order. Values are represented as 2 or 4 hexadecimal digit values. See also "DSD" on page 206.

stand-alone
Not connected to a host computer. The DT800 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.

star (*)
Also called **asterisk** or **splat**

state
Of an alarm: the true/false result of an alarm test. The actual states or transitions recognized by the DT800 are
- continuing false
- false-to-true
- continuing true
- true-to-false.

See "Alarm States and Tags" on page 103.

SWFC
Software flow control (XON/XOFF). Also known as **software handshaking**. See "flow control" on page 206.

switch
Full name: **switch command**. A software control. A two-state (ON or OFF) command that changes a DT800 internal setting. For example, sending the switch command `/R` to the DT800 turns ON the return-of-data-to-the-host-computer switch, and sending `/r` turns it OFF. See "Switches" beginning on page 111 for a complete listing.

syntax error
An error in the order, arrangement, case or spelling of the components of a command. For example, sending the command `CardClear` (incorrect case; should be `CARDCLEAR`) to the DT800 causes it to generate the syntax error `E10 — Command error`.

TCP/IP
Transmission Control Protocol / Internet Protocol. A commonly-used family of communication protocols. TCP/IP is the DT800's default Ethernet protocol. See "protocol" on page 209.

thermocouple
A temperature-sensing device constructed from dissimilar metals. See "Thermocouples" on page 143.

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 component that handles asynchronous serial communication. Every computer, internal modem and DT800 has its own UART to manage each of its serial ports.

UDP
User Datagram Protocol. A component of the TCP/IP suite of internet protocols (communication methods).

unshared input
See "Independent Analog Inputs" on page 13.

USB
Universal Serial Bus. An external bus standard that supports data transfer rates of 12mbps, plug-and-play device installation and removal, and hot plugging. A single USB port can be used to connect up to 127 devices.

version number
The version number of the DT800's firmware consists of a major number, a minor number and a build number — see Figure 153.

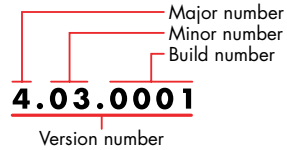


FIGURE 153 Version number components

XON/XOFF
Transmitter on / transmitter off. Control characters used for flow control, 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" on page 203.

Index

Symbols

Numerics

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

S

T

U

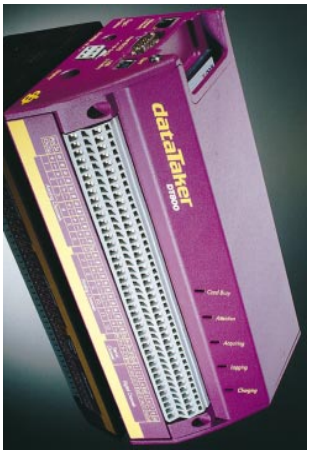
V

W

X

Y

Z


dataTaker
DT600

DATATAKER OFFICES

www.datataker.com

Head Office

Australia

Datataker Pty Ltd
7 Seismic Court
Rowville Melbourne
Victoria 3178

Tel: +61 3 9764 8600
Fax: +61 3 9764 8997
Email: sales@datataker.com.au

Master Distributors

United Kingdom

Grant Instruments (Cambridge) Ltd
Shepreth
Cambridgeshire
SG8 6GB

Tel: +44 (0) 1763 264780
Fax: +44 (0) 1763 262410
Email: sales@datataker.co.uk

United States of America

Computer Aided Solutions
8588 Mayfield Rd, Suite One
Chesterland, OH 44026

Tel: +1 800 9 LOGGER
Tel: +1 440 729 2570
Fax: +1 413 375 6137
Email: sales@datataker.com

China

Earth Products China Limited
Rm 1205-06, Seapower Centre
73-77 Lei Muk Road
Kwai Chung Hong Kong

Tel: +852 2392 8698
Fax: +852 2395 5655
Email: info@epc.com.hk
Web: http://www.epc.com.hk

CE

LLOYD'S REGISTER QUALITY ASSURANCE · ISO9001 · UKAS QUALITY MANAGEMENT 001

Your local dealer