CR5000 Measurement and Control System

Operator's Manual

Issued: 25.01.07

CSL 427

Guarantee

This equipment is guaranteed against defects in materials and workmanship. This guarantee applies for thirty-six months from date of delivery. We will repair or replace products which prove to be defective during the guarantee period provided they are returned to us prepaid. The guarantee will not apply to:

- Equipment which has been modified or altered in any way without the written permission of Campbell Scientific
- Batteries
- Any product which has been subjected to misuse, neglect, acts of God or damage in transit.

Campbell Scientific will return guaranteed equipment by surface carrier prepaid. Campbell Scientific will not reimburse the claimant for costs incurred in removing and/or reinstalling equipment. This guarantee and the Company's obligation thereunder is in lieu of all other guarantees, expressed or implied, including those of suitability and fitness for a particular purpose. Campbell Scientific is not liable for consequential damage.

Please inform us before returning equipment and obtain a Repair Reference Number whether the repair is under guarantee or not. Please state the faults as clearly as possible, and if the product is out of the guarantee period it should be accompanied by a purchase order. Quotations for repairs can be given on request.

When returning equipment, the Repair Reference Number must be clearly marked on the outside of the package.

Note that goods sent air freight are subject to Customs clearance fees which Campbell Scientific will charge to customers. In many cases, these charges are greater than the cost of the repair.

Campbell Scientific Ltd,
Campbell Park, 80 Hathern Road,
Shepshed, Leicestershire, LE12 9GX UK
Tel: +44 (0) 1509 601141
Fax: +44 (0) 1509 601091
Email: support@campbellsci.co.uk
http://www.campbellsci.co.uk

CR5000 MEASUREMENT AND CONTROL SYSTEM
TABLE OF CONTENTS

CR5000 Overview

The CR5000 provides precision measurement capabilities in a rugged, battery-operated package. The system makes measurements at a rate of up to 5,000 samples/second with 16-bit resolution. The CR5000 includes CPU, keyboard display, power supply, and analogue and digital inputs and outputs. The on-board, BASIC-like programming language includes data processing and analysis routines. PC9000 Software provides program generation and editing, data retrieval, and realtime monitoring.

FIGURE OV1-1. CR5000 Measurement and Control System

OV1. Physical Description

Figure OV1-2 shows the CR5000 panel and the associated program instructions. Unless otherwise noted, they are measurement instructions (Section 7).

OV1.1 Measurement Inputs

OV1.1.1 Analogue Inputs

There are 20 differential or 40 single-ended inputs for measuring voltages up to ±5 V. A thermistor installed in the wiring panel can be used to measure the reference temperature for thermocouple measurements, and a heavy copper grounding bar and connectors combine with the case design to reduce temperature gradients for accurate thermocouple measurements. Resolution on the most sensitive range is 0.67 µV

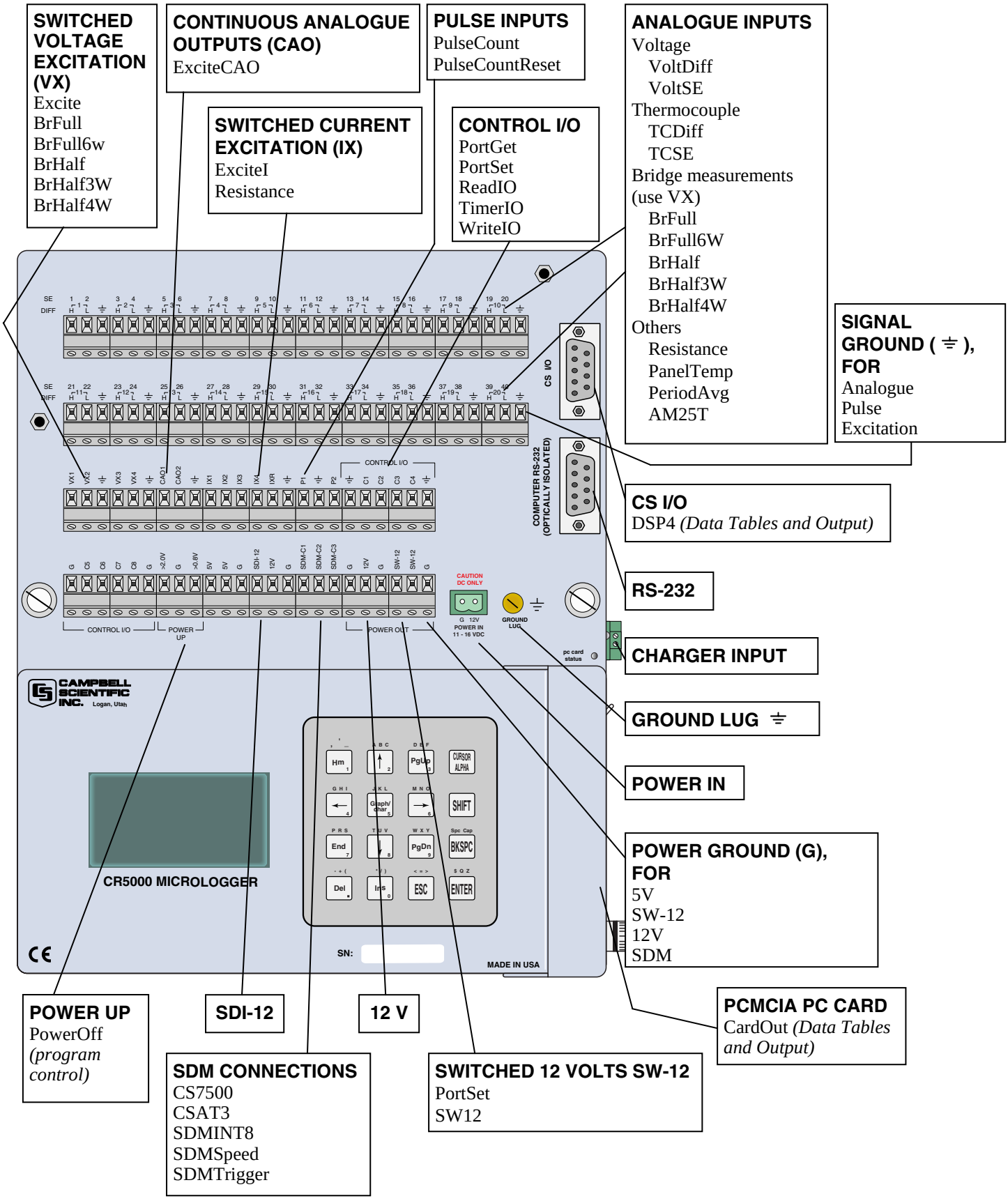


FIGURE OV1-2. CR5000 Panel and Associated Instructions.

OV1.1.2 Signal Grounds (⏚)

The Signal Grounds (⏚) should be used as the reference for Single-ended Analogue inputs, Excitation returns, and sensor shield wires.

Signal returns from the CAO and Pulse channels should use the ⏚ terminals located on the CAO and Pulse terminal strip to minimize current flow through the ⏚ grounds on the analogue terminal strips.

OV1.1.3 Power Grounds (G)

The Power Grounds (G) should be used as the returns for the 5V, SW12, 12V, and C1-C8 outputs. Use of the G grounds for these outputs with potentially large currents will minimize current flow through the analogue section, which can cause Single-ended voltage measurement errors.

OV1.1.4 Ground Lug ⏚

The large ground lug is used to connect a heavy gauge wire to earth ground. A good earth connection is necessary fix the ground potential of the datalogger and to send to earth transients that come in on either the G or ⏚ terminals or are shunted to ground via the spark gaps protecting other inputs.

OV1.1.5 Power In

The G and 12V terminals on the unplugable Power In connector are for connecting power from an external battery to the CR5000. These are the only terminals that can be used to input battery power; the other 12V and SW-12V terminals are out only. Power from this input will not charge internal CR5000 batteries. Power to charge the internal batteries (17-28 VDC or 18 VRMS AC) must be connected to the charger input on the side of the LA battery back.

OV1.1.6 Switched 12 Volts SW-12

The SW-12 terminals provide an unregulated 12 volts that can be switched on and off under program control.

OV1.1.7 Switched Voltage Excitation (VX)

Four switched excitation channels provide precision programmable voltages within the ±5 Volt range for bridge measurements. Each analogue output will provide up to 50 mA between ±5 V.

OV1.1.8 Switched Current Excitation (IX)

Four Switched Current Excitation channels provide precision current excitations programmable within ±2.5 mA for resistance or bridge measurements.

OV1.1.9 Continuous Analogue Outputs (CAO)

Two Continuous Analogue Outputs (CAO) with individual outputs under program control for proportional control (e.g., PID algorithm) and waveform generation. Each analogue output will provide up to 15 mA between ±5 V.

OV1.1.10 Control I/O

There are 8 digital Input/Output channels (0 V low, 5 V high) for frequency measurement, digital control, and triggering.

OV1.1.11 Pulse Inputs

Two Pulse input channels can count pulses from high-level (5 V square wave), switch closure, or low-level A/C signals.

OV1.1.12 Power Up

The CR5000 allows shutting off power under program control. The Power Up inputs allow an external signal to awaken the CR5000 from a powered down state (PowerOff, Section 9). When the CR5000 is in this power off state the ON Off switch is in the on position but the CR5000 is off. If the "<0.5 " input is switched to ground or if the ">2" input has a voltage greater than 2 volts applied, the CR5000 will awake, load and run the “run on power-up” program. If the "< 0.5" input continues to be held at ground while the CR5000 is powered on and goes through its 2-5 second initialization sequence, the CR5000 will not run “run on power-up” program.

OV1.1.13 SDM Connections

The Synchronous Device for Measurement (SDM) connections C1,C2, and C3 along with the adjacent 12 volts and ground terminals are used to connect SDM sensors and peripherals.

OV1.2 Communication and Data Storage

OV1.2.1 PCMCIA PC Card

One slot for a Type I/II/III PCMCIA card. The keyboard display is used to check card status. The card must be powered down before removing it. The card will be reactivated if not removed.

CAUTION Removing a card while it is active can cause garbled data and can actually damage the card. Do not switch off the CR5000 power while the card is present and active.

OV1.2.2 CS I/O

A 9-pin serial I/O port supports CSI peripherals.

OV1.2.3 Computer RS-232

RS-232 Port

OV1.3 Power Supply and AC Adapter

The CR5000 has two base options the low profile without any power supply and the lead acid battery power supply base. The low profile base requires an external DC power source connected to the Power In terminal on the panel.

The battery base has a 7 amp hour battery with built in charging regulator and includes an AC adapter for use where 120 VAC is available (18 VAC RMS output). Charging power can also come from a 17-28 VDC input such as a solar panel. The DCDC18R is available for stepping the voltage up from a nominal 12 volt source (e.g., vehicle power supply) to the DC voltage required for charging the internal battery.

OV2. Memory and Programming Concepts

OV2.1 Memory

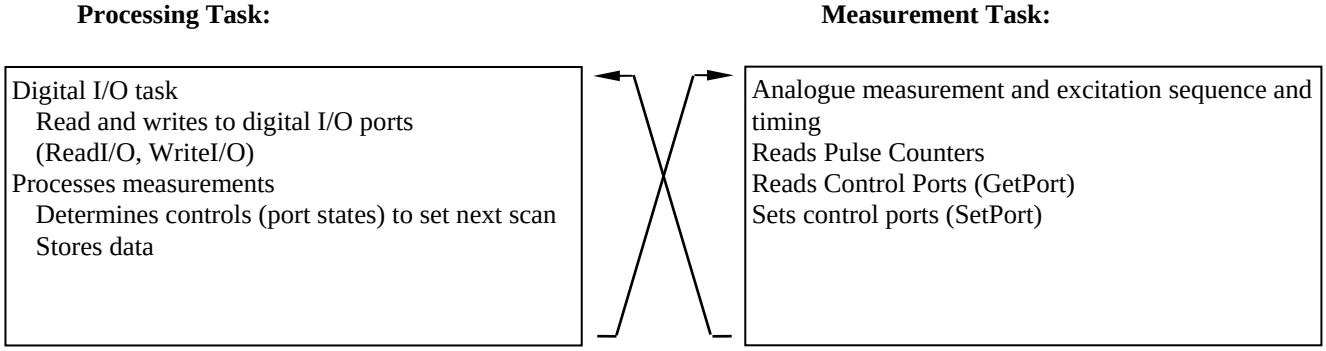
The CR5000 has 2MB SRAM and 1MB Flash EEPROM. The operating system and user programs are stored in the flash EEPROM. The memory that is not used by the operating system and program is available for data storage. The size of available memory may be seen in the status file. Additional data storage is available by using a PCMCIA card in the built in card slot.

OV2.2 Measurements, Processing, Data Storage

The CR5000 divides a program into two tasks. The **measurement task** manipulates the measurement and control hardware on a rigidly timed sequence. The **processing task** processes and stores the resulting measurements and makes the decisions to actuate controls.

The measurement task stores raw Analogue to Digital Converter (ADC) data directly into memory. As soon as the data from a scan is in memory, the processing task starts. There are at least two buffers allocated for this raw ADC data (more under program control), thus the buffer from one scan can be processed while the measurement task is filling another.

When a program is compiled, the measurement tasks are separated from the processing tasks. When the program runs, the measurement tasks are performed at a precise rate, ensuring that the measurement timing is exact and invariant.



The CR5000 can store individual measurements or it may use its extensive processing capabilities to calculate averages, maxima, minima, histogams, FFTs, etc., on periodic or conditional intervals. Data are stored in tables such as listed in Table OV2-1. The values to output are selected when running the program generator or when writing a datalogger program directly.

Table OV2-1. Typical Data Table

TOA4 TIMESTAMP TS	StnName RECORD RN	Temp RefTemp_Avg degC Avg	TC_Avg(1) degC Avg	TC_Avg(2) degC Avg	TC_Avg(3) degC Avg	TC_Avg(4) degC Avg	TC_Avg(5) degC Avg	TC_Avg(6) degC Avg
1995-02-16 15:15:04.61	278822	31.08	24.23	25.12	26.8	24.14	24.47	23.76
1995-02-16 15:15:04.62	278823	31.07	24.23	25.13	26.82	24.15	24.45	23.8
1995-02-16 15:15:04.63	278824	31.07	24.2	25.09	26.8	24.11	24.45	23.75
1995-02-16 15:15:04.64	278825	31.07	24.21	25.1	26.77	24.13	24.39	23.76

OV3. PC9000 Application Software

PC9000 is a Windows™ application for use with the CR5000. The software supports CR5000 program generation, real-time display of datalogger measurements, graphing, and retrieval of data files.

OV3.1 Hardware and Software Requirements

The following computer resources are necessary:

- IBM PC, Portable or Desktop
- 8 Meg of Ram
- VGA Monitor
- Windows 95 or newer
- 30 Meg of Hard Drive Space for software
- 40 Meg of Hard Drive Space for data
- RS232 Serial Port

The following computer resources are recommended:

- 16 Meg of Ram
- 33 MHz 486 or faster
- Mouse

OV3.2 PC9000 Installation

To install the PC9000 Software:

- Start Microsoft Windows
- Insert diskette 1 (marked 1 of 2) in a disk drive.
- From the Program Manager, select **F**ile menu and choose **R**un
- Type (disk drive):\setup and press Enter e.g. a:\setup<Enter>
- The setup routine will prompt for disk 2.

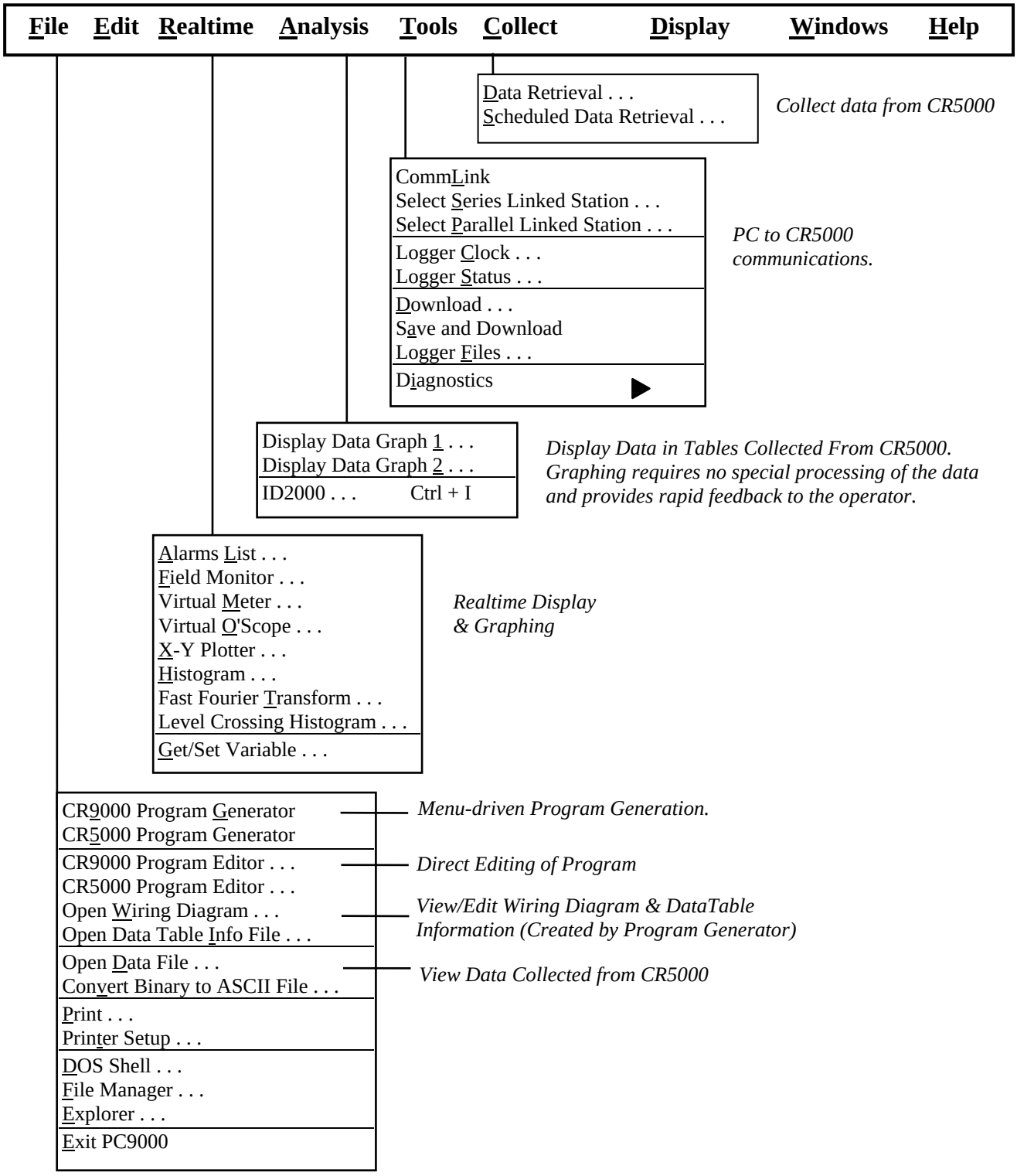
You may use the default directory of PC9000 or install the software in a different directory. The directory will be created for you.

To abort the installation, type Ctrl-C or Break at any time.

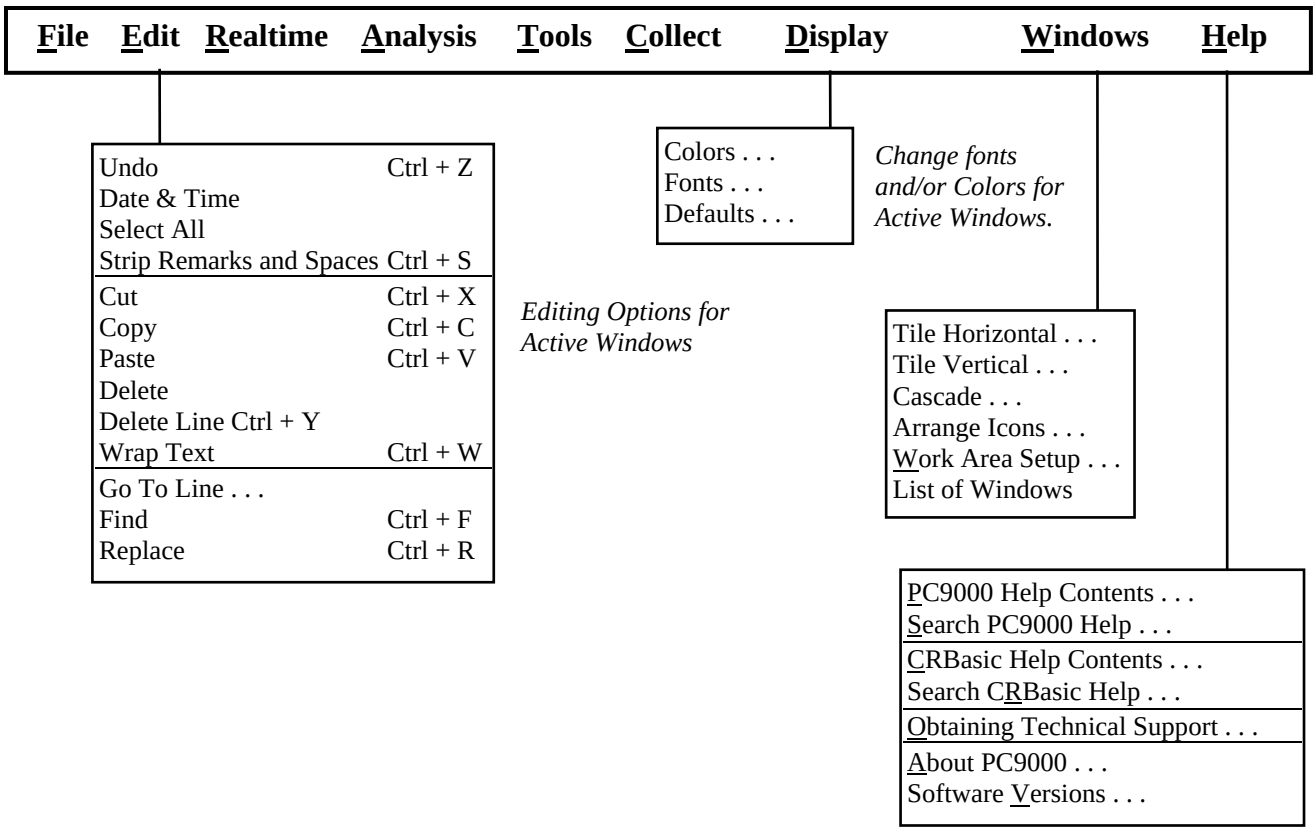
OV3.3 PC9000 Software Overview

This overview points out the main PC9000 functions and where to find them. PC9000 has extensive on-line help to guide the user in its operation, run PC9000 to get the details. A CR5000 is not necessary to try out the programming and real time display options; a demo uses canned data for viewing. Without a CR5000, there are no communications with the datalogger; operations such as downloading programs and retrieving data will not function.

Figures OV3-1 and OV3-2 show the main PC9000 menus. The primary functions of PC9000 are accessed from the File, Comm, Realtime, and Analysis selections on the main menu (Figure OV3-1).



OV3-1. PC9000 Primary Functions



OV3-2. PC9000 Editing, Help, and User Preferences

OV3.3.1 File

Program Generator
Guides the user through a series of menus to configure the measurement types: thermocouple, voltage, bridge, pulse counting, frequency, and others. Creates a CR5000 program, wiring diagram, output table, description, and configuration file.

Program Editor
Create programs directly or edit those created by the program generator or retrieved from the CR5000. Provides context-sensitive help for the CR5000's BASIC-like language.

OV3.3.2 Edit

REALTIME

Virtual Meter
Updates up to five displays simultaneously. Choices include analogue meter, horizontal and vertical bars, independent scaling/offset, multiple alarms, and rapid on-site calibration of sensors

Virtual Oscilloscope
Displays up to six channels. Time base variable from milliseconds to hours.

X-Y Plotter
Allows comparison of any two measurements in real time.

OV3.3.3 Analysis

Data Graphing
Displays up to 16 fields simultaneously as strip charts or two multi-charts with up to 8 traces each. Includes 2D/3D bars, line, log/linear, area, and scatter. Line statistics available for max/min, best fit, mean, and standard deviation. Handles files of unlimited size. Historical graphing requires no special processing of the data and provides rapid feedback to the operator.

OV3.3.4 Tools

Control and Communications
Supports PC to CR5000 communications: clock read/set, status read, program download, and program retrieval.

OV3.3.5 Collect

Collect data from CR5000 data tables

OV3.3.6 Display

Configure the font and colour scheme in an active window.

OV3.3.7 Windows

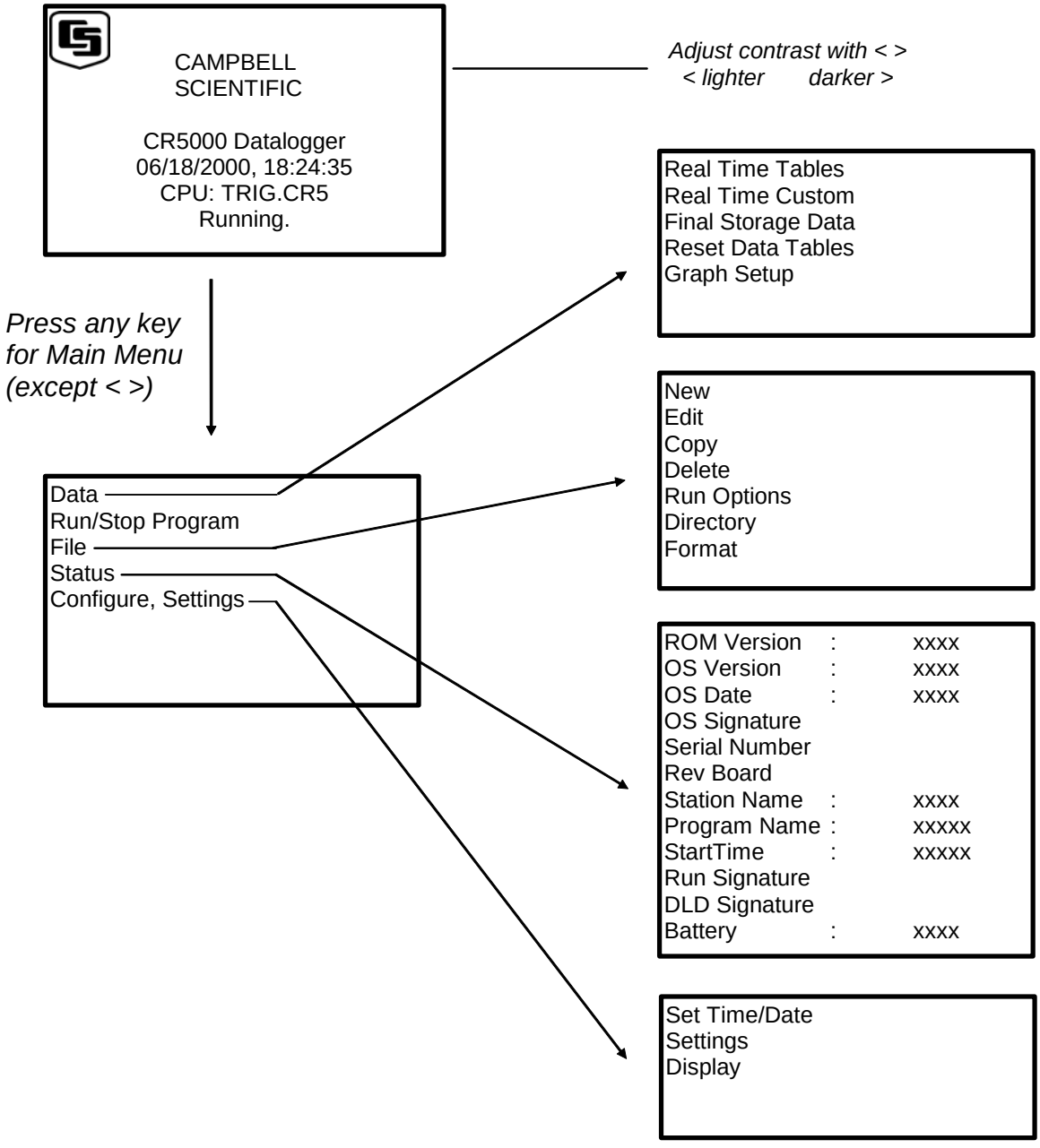
Size and arrange windows.

OV3.3.8 Help

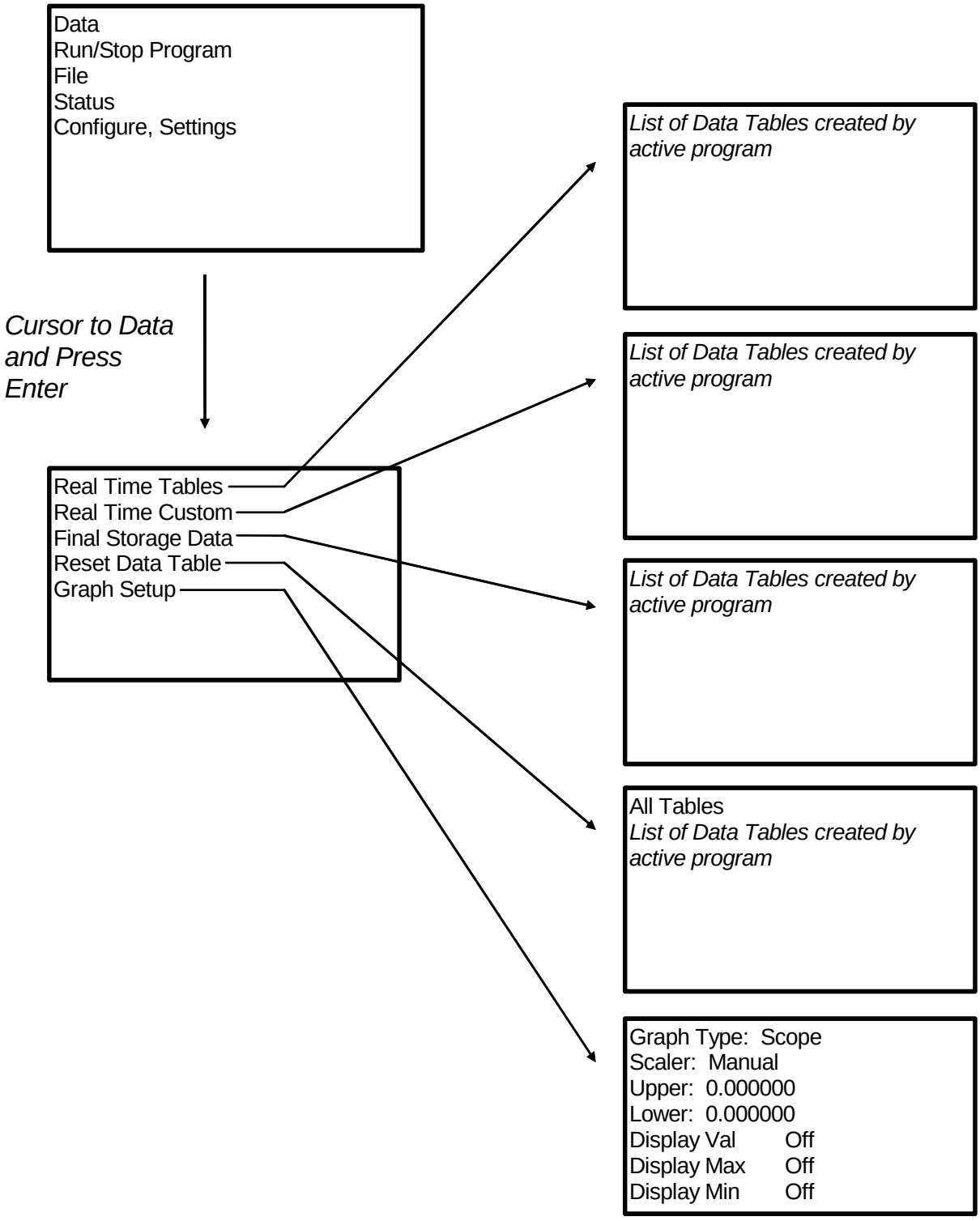
On-line help for PC9000 software.

OV4. Keyboard Display

Power Up Screen

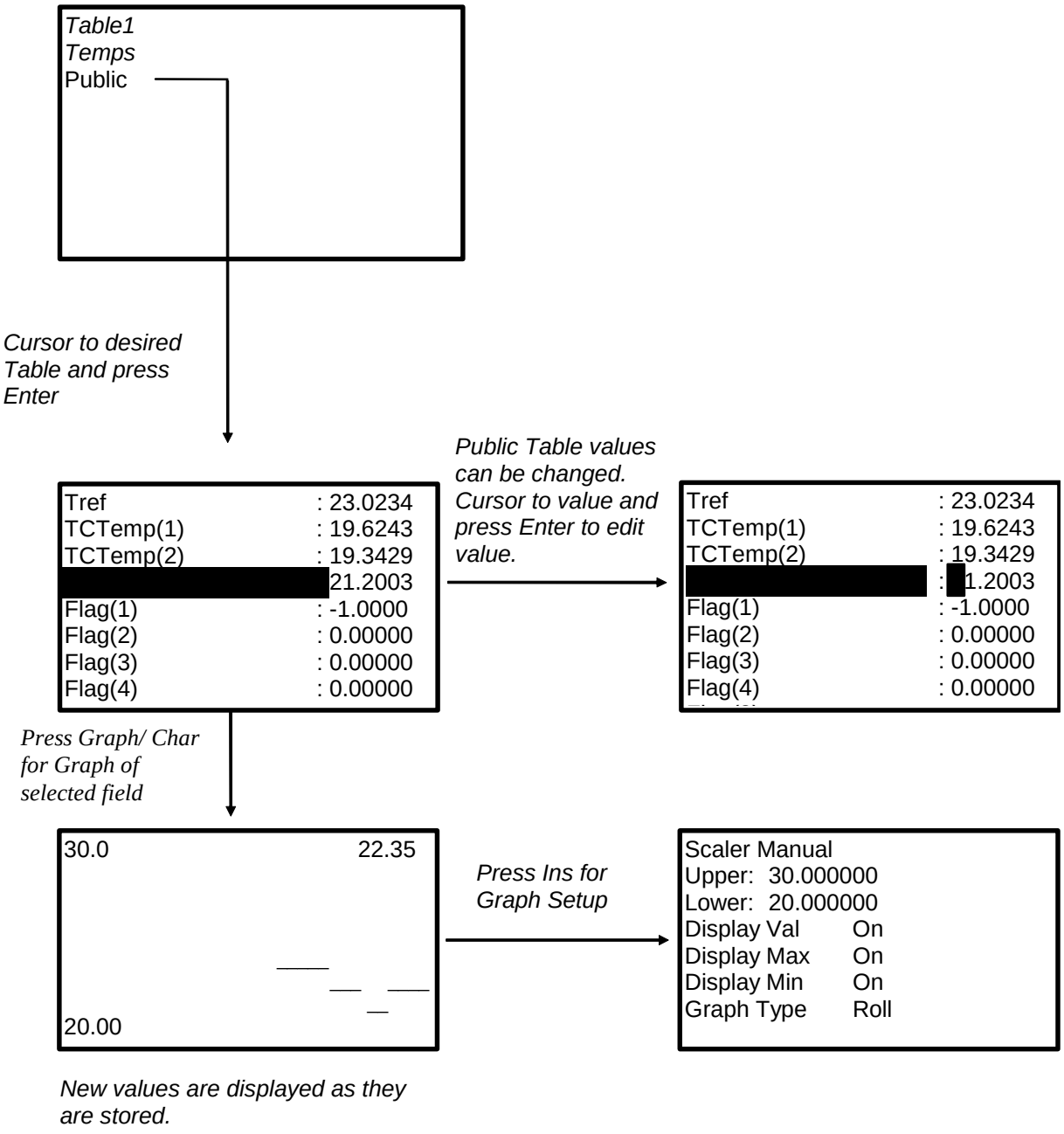


OV4.1 Data Display



OV4.1.1 Real Time Tables

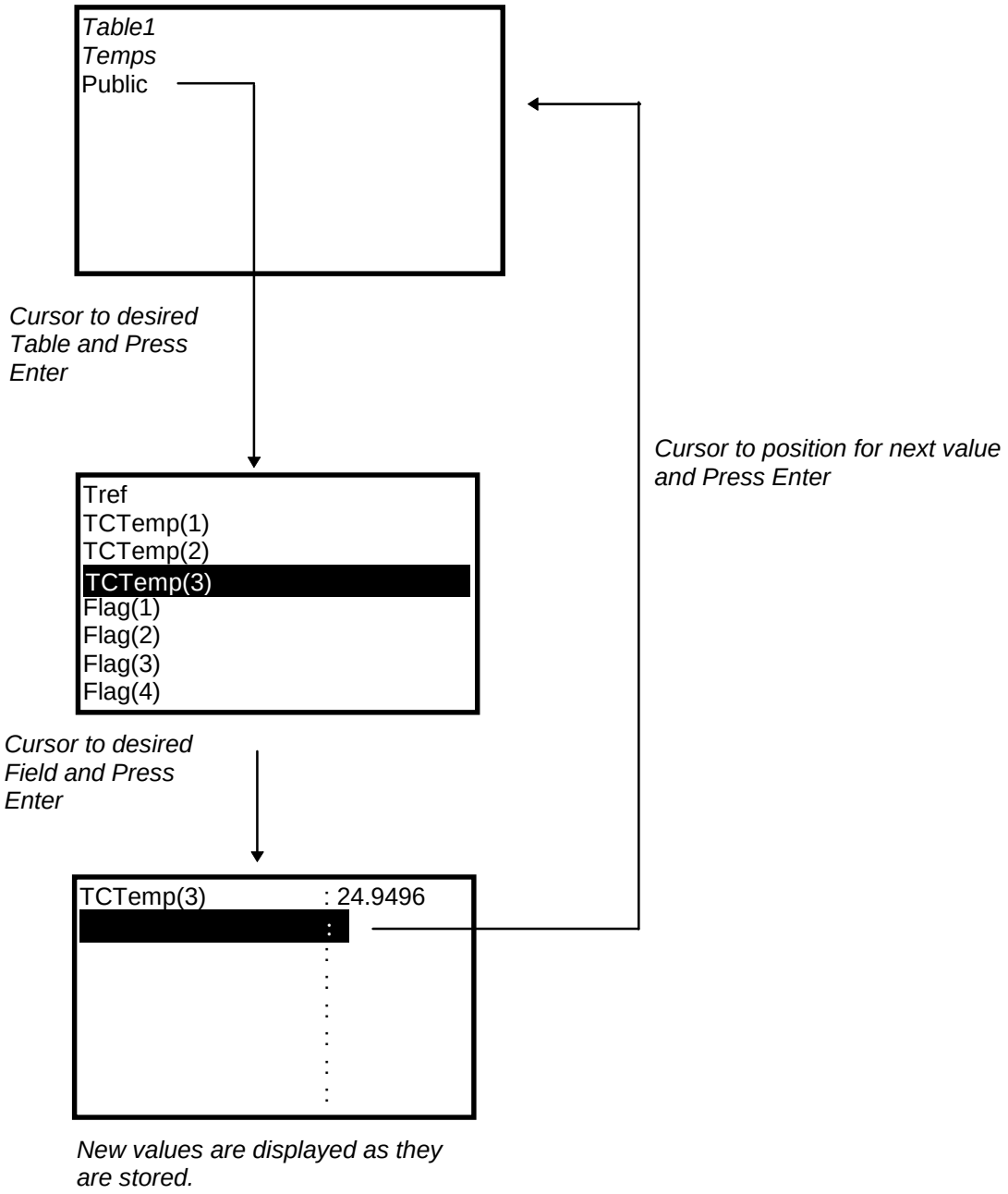
List of Data Tables created by active program. For Example,



New values are displayed as they are stored.

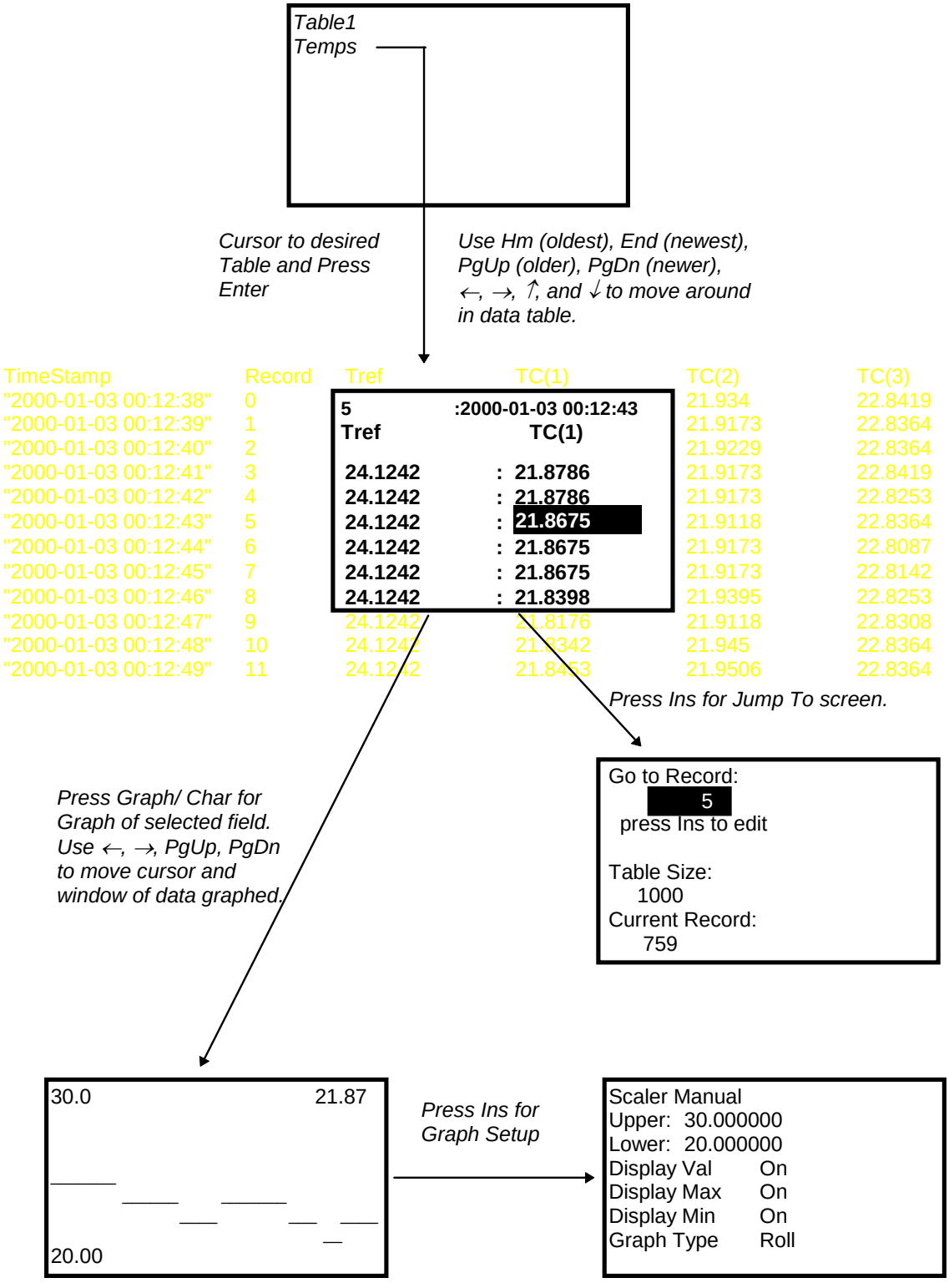
OV4.1.2 Setting up Real Time Custom Display

List of Data Tables created by active program. For Example,



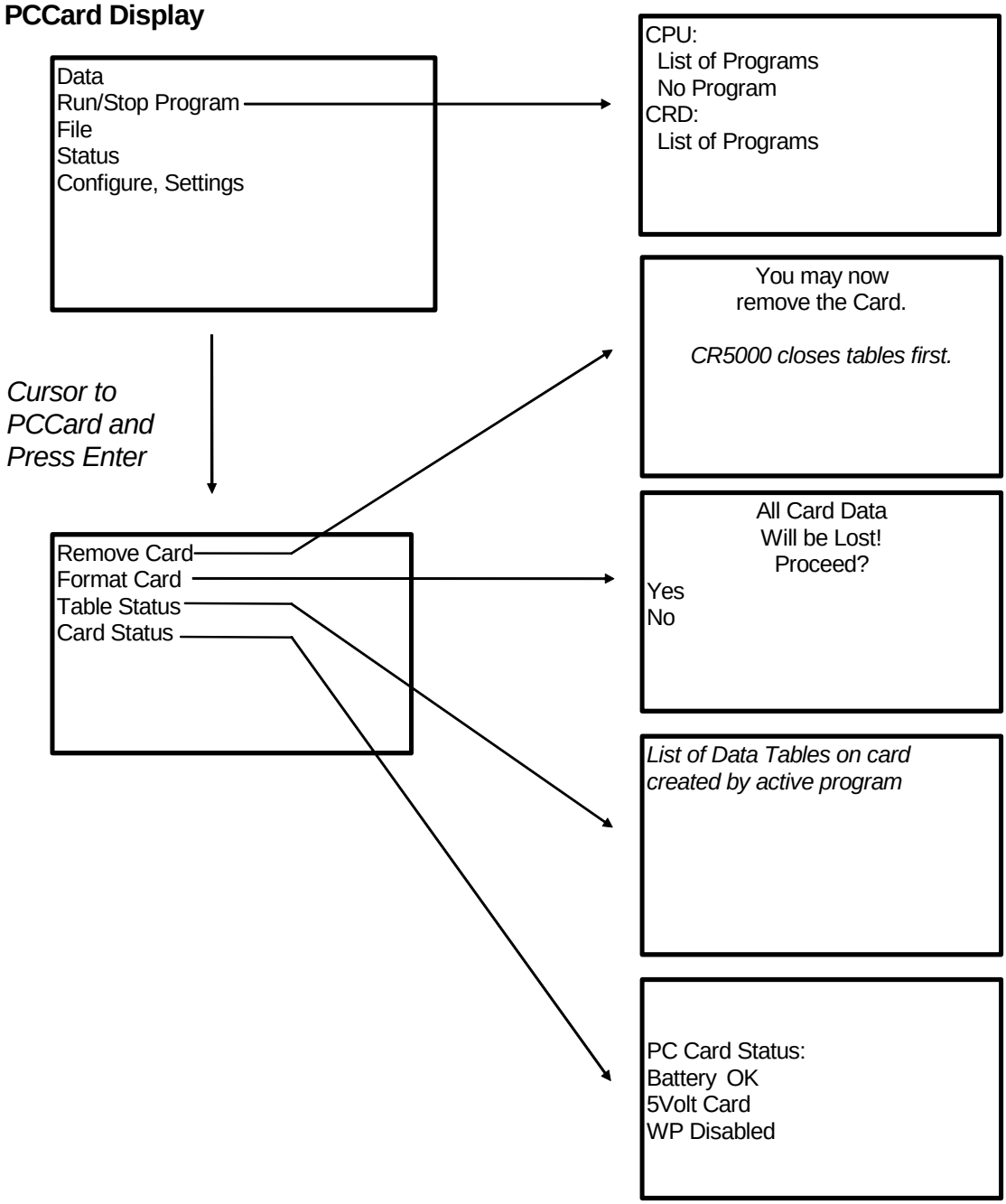
OV4.1.3 Final Storage Tables

List of Data Tables created by active program. For Example:

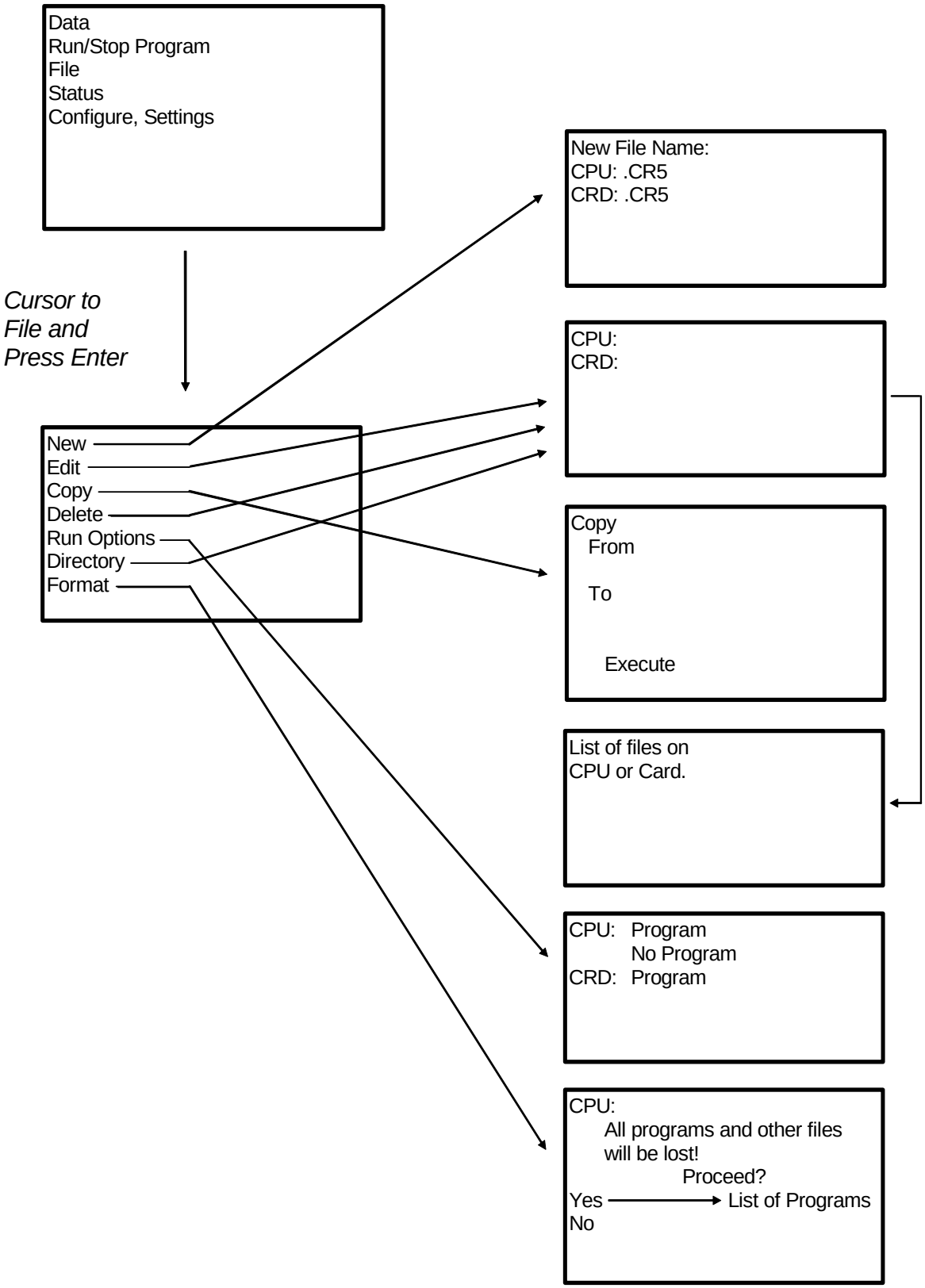


OV4.2 Run/Stop Program

PCCard Display

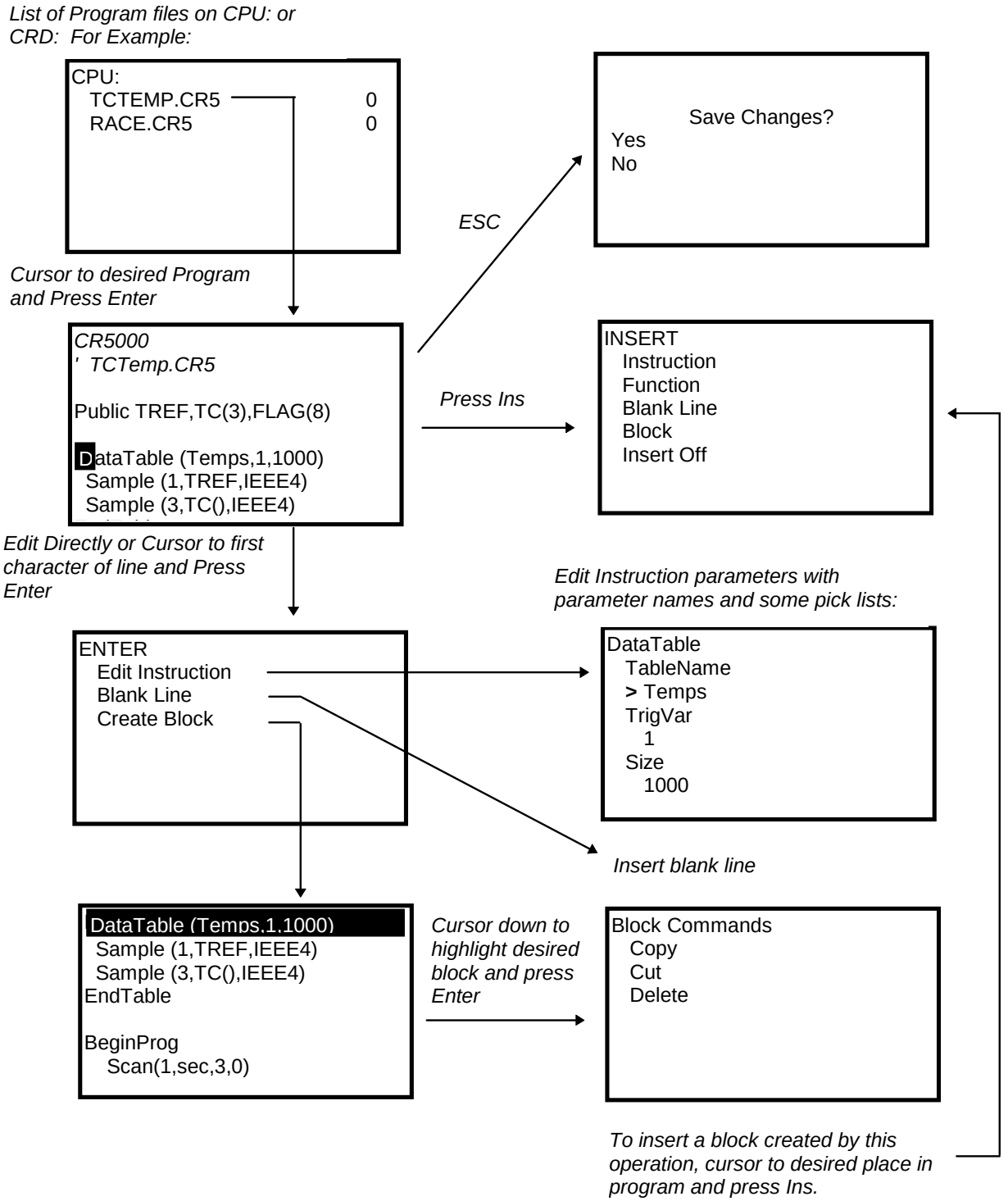


OV4.3 File Display

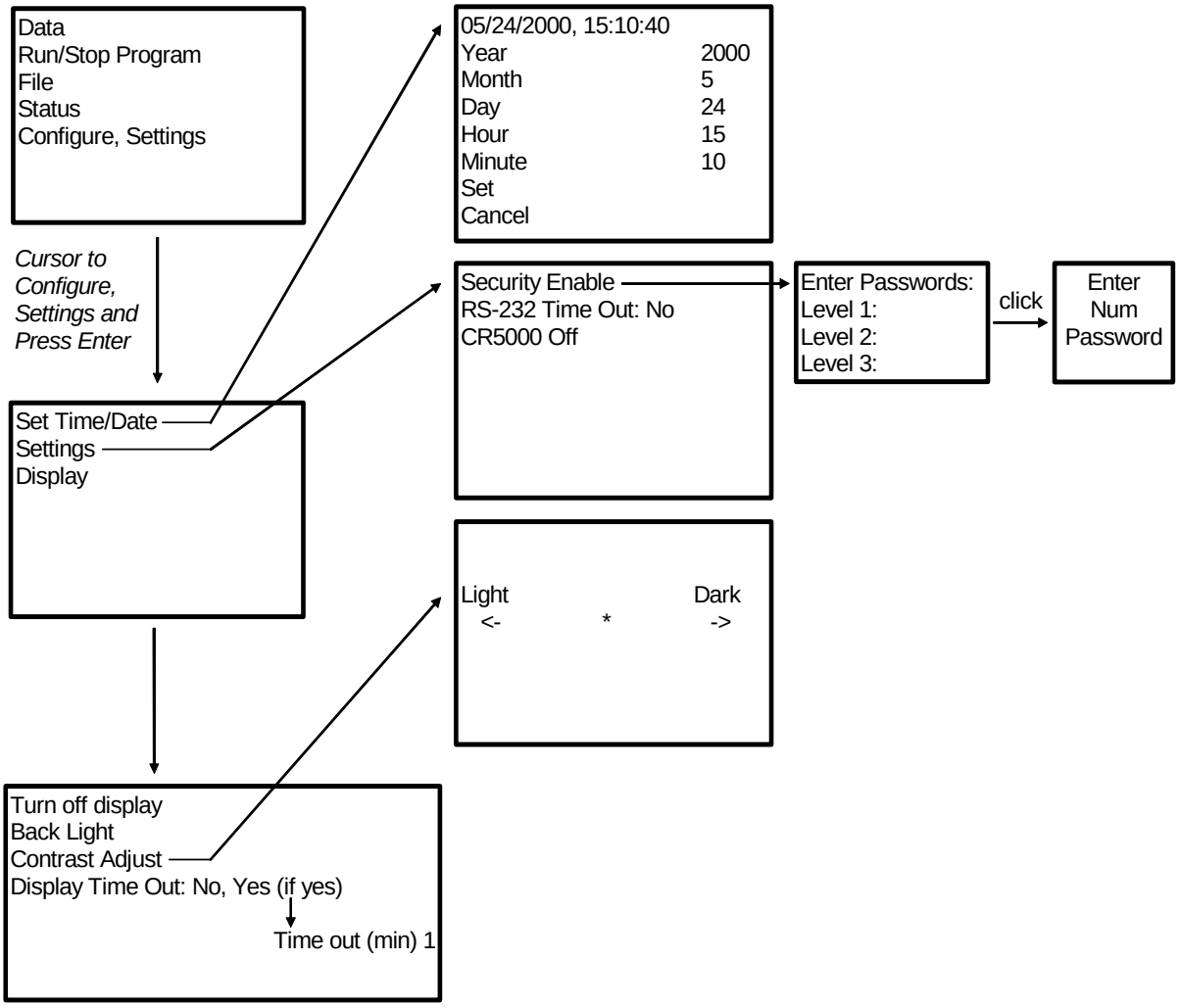


OV4.3.1 File: Edit

The Program Editor in PC9000 is recommended for writing and editing datalogger programs. Changes in the field can be made with the keyboard display.



OV4.4 Configure Display



OV5. Specifications

Electrical specifications are valid over a -25° to +50°C range unless otherwise specified; testing over -40° to +85°C available as an option, excludes batteries. Non-condensing environment required. Yearly calibrations are recommended to maintain electrical specifications.

PROGRAM EXECUTION RATE

The CR5000 can measure one channel and store the result in 500 µs; all 40 SE* channels can be measured in 8 ms (5 kHz aggregate rate).

ANALOG INPUTS

DESCRIPTION: 20 DF* or 40 SE, individually configured. Channel expansion provided through AM16/32, AM416, and AM25T Multiplexers.

RANGES, RESOLUTION, AND TYPICAL INPUT NOISE: Basic Resolution (Basic Res) is the A/D resolution of a single conversion. **Resolution of DFM* with input reversal is half the Basic Res.** Noise values are for DFM with input reversal; noise is greater with SEM.*

Input Rng (mV)	Basic Res (µV)	0 Int. (µV RMS)	250 µs Int. (µV RMS)	20/16.7 ms Int. (µV RMS)
±5000	167	70	60	30
±1000	33.3	30	12	6
±200	6.67	8	2.4	1.2
±50	1.67	3.0	0.8	0.3
±20	0.67	1.8	0.5	0.2

ACCURACY†:

±(0.05% of Reading + Offset) 0° to 40°C
±(0.075% of Reading + Offset) -25° to 50°C
±(0.10% of Reading + Offset) -40° to 85°C

Offset for DFM w/input reversal = Basic Res +1 µV

Offset for DFM w/o input reversal = 2Basic Res + 2 µV

Offset for SEM = 2Basic Res + 10 µV

MINIMUM TIME BETWEEN MEASUREMENTS:

Zero Integration: 125 µs
250 µs Integration: 475 µs
16.7 ms Integration: 19.9 ms
20 ms Integration: 23.2 ms

COMMON MODE RANGE: ±5 V

DC COMMON MODE REJECTION: >100 dB with input reversal (>80 dB without input reversal)

NORMAL MODE REJECTION: 70 dB @ 60 Hz when using 60 Hz rejection

SUSTAINED INPUT VOLTAGE WITHOUT DAMAGE: ±16 Vdc

INPUT CURRENT: ±2 nA typ., ±10 nA max. @ 50°C

INPUT RESISTANCE: 20 GΩ typical

ACCURACY OF INTERNAL THERMOCOUPLE REFERENCE JUNCTION:

±0.25°C, 0° to 40°C
±0.5°C, -25° to 50°C
±0.7°C, -40° to 85°C

ANALOG OUTPUTS

DESCRIPTION: 4 switched voltage; 4 switched current; 2 continuous voltage; switched outputs active only during measurements, one at a time.

RANGE: Voltage (current) outputs programmable between ±5 V (±2.5 mA)

RESOLUTION: 1.2 mV (0.6 µA) for voltage (current) outputs

ACCURACY: ±10 mV (±10 µA) for voltage (current) outputs

CURRENT SOURCING: 50 mA for switched voltage; 15 mA for continuous

CURRENT SINKING: 50 mA for switched voltage; 5 mA for continuous (15 mA w/selectable option)

COMPLIANCE VOLTAGE: ±5 V for switched current excitation

RESISTANCE MEASUREMENTS

Provides voltage ratio measurements of 4- and 6-wire full bridges, and 2-, 3-, 4-wire half bridges. Direct resistance measurements available with current excitation. Dual-polarity excitation is recommended.

VOLTAGE RATIO ACCURACY†: Assumes input and excitation reversal and an excitation voltage of at least 2000 mV.

±(0.04% Reading + Basic Res/4) 0° to 40°C
±(0.05% Reading + Basic Res/4) -25° to 50°C
±(0.06% Reading + Basic Res/4) -40° to 85°C

ACCURACY† WITH CURRENT EXCITATION: Assumes input and excitation reversal, and an excitation current, I_x, of at least 1 mA.

±(0.075% Reading + Basic Res/2I_x) 0° to 40°C
±(0.10% Reading + Basic Res/2I_x) -25° to 50°C
±(0.12% Reading + Basic Res/2I_x) -40° to 85°C

PERIOD AVERAGING MEASUREMENTS

DESCRIPTION: The average period for a single cycle is determined by measuring the duration of a specified number of cycles. Any of the 40 SE analog inputs can be used; signal attenuation and ac coupling may be required.

INPUT FREQUENCY RANGE:

Input Rng (mV)	Signal (peak to peak) Min.	Max.[1]	Min. Pulse W.	Max. Freq
±5000	600 mV	10 V	2.5 µs	200 kHz
±1000	100 mV	2.0 V	5.0 µs	100 kHz
±200	4 mV	2.0 V	25 µs	20 kHz

[1]Maximum signals must be centered around datalogger ground.

RESOLUTION: 70 ns/number of cycles measured

ACCURACY: ±(0.03% of Reading + Resolution)

PULSE COUNTERS

DESCRIPTION: Two 16-bit inputs selectable for switch closure, high frequency pulse, or low-level ac.

MAXIMUM COUNT: 4 x 10^9 counts per scan

SWITCH CLOSURE MODE:
Minimum Switch Closed Time: 5 ms
Minimum Switch Open Time: 6 ms
Maximum Bounce Time: 1 ms open without being counted.

HIGH FREQUENCY PULSE MODE:
Maximum Input Frequency: 400 kHz
Maximum Input Voltage: ±20 V
Voltage Thresholds: Count upon transition from below 1.5 V to above 3.5 V at low frequencies. Larger input transitions are required at high frequencies because of 1.2 µs time constant filter.

LOW LEVEL AC MODE:
Internal ac coupling removes dc offsets up to ±0.5 V.

Input Hysteresis: 15 mV
Maximum ac Input Voltage: ±20 V
Minimum ac Input Voltage (sine wave):

(mV RMS)	Range (Hz)
20	1.0 to 1000
200	0.5 to 10,000
1000	0.3 to 16,000

DIGITAL I/O PORTS

DESCRIPTION: 8 ports selectable as binary inputs or control outputs.

OUTPUT VOLTAGES (no load): high 5.0 V ±0.1 V; low < 0.1 V

OUTPUT RESISTANCE: 330 Ω

INPUT STATE: high 3.0 to 5.3 V; low -0.3 to 0.8 V

INPUT RESISTANCE: 100 kΩ

EMI and ESD PROTECTION

The CR5000 is encased in metal and incorporates EMI filtering on all inputs and outputs. Gas discharge tubes provide robust ESD protection on all terminal block inputs and outputs. The following European CE standards apply.

EMC tested and conforms to BS EN61326:1998.

Details of performance criteria applied are available upon request.

Warning: This is a Class A product. In a domestic environment this product may cause radio interference in which case the user may be required to correct the interference at the user's own expense.

CPU AND INTERFACE

PROCESSOR: Hitachi SH7034

MEMORY: Battery-backed SRAM provides 2 Mbytes for data and operating system use with 128 kbytes reserved for program storage. Expanded data storage with PCMCIA type I, type II, or type III card.

DISPLAY: 8-line-by-21 character alphanumeric or 128 x 64 pixel graphic LCD display w/backlight.

SERIAL INTERFACES: Optically isolated RS-232 9-pin interface for computer or modem. CSI/O 9-pin interface for peripherals such as CSI modems.

BAUD RATES: Selectable from 1,200 to 115,200 bps. ASCII protocol is eight data bits, one start bit, one stop bit, no parity.

CLOCK ACCURACY: ±1 minute per month

SYSTEM POWER REQUIREMENTS

VOLTAGE: 11 to 16 Vdc

TYPICAL CURRENT DRAIN: 400 µA software power off; 1.5 mA sleep mode; 4.5 mA at 1 Hz (200 mA at 5 kHz) sample rate.

INTERNAL BATTERIES: 7 Ahr rechargeable base (optional); 1650 mAhr lithium battery for clock and SRAM backup, 10 years of service typical, less at high temperatures.

EXTERNAL BATTERIES: 11 to 16 Vdc; reverse polarity protected.

PHYSICAL SPECIFICATIONS

SIZE: 9.8" x 8.3" x 4.5" (24.7 cm x 21.0 cm x 11.4 cm) Terminal strips extend 0.4" (1.0 cm).

WEIGHT: 4.5 lbs (2.0 kg) with low-profile base; 12.2 lbs (5.5 kg) with rechargeable base

WARRANTY

Three years against defects in materials and workmanship.

*SE(M): Single-Ended (Measurement)

*DF(M): Differential (Measurement)

† Sensor and measurement noise not included.

We recommend that you confirm system configuration and critical specifications with Campbell Scientific before purchase.

Section 1. Installation and Maintenance

1.1 Protection from the Environment

The normal environmental variables of concern are temperature and moisture. The standard CR5000 is designed to operate reliably from -25 to +50°C (-40°C to +85°C, optional) in noncondensing humidity. When humidity tolerances are exceeded, damage to IC chips, microprocessor failure, and/or measurement inaccuracies due to condensation on the various PC board runners may result. Effective humidity control is the responsibility of the user.

The CR5000 is not hermetically sealed. Two half unit packets of DESI PAK desiccant are located by the batteries. A dry package weighs approximately 19 grams and will absorb a maximum of six grams of water at 40% humidity and 11 grams at 80%. Desiccant packets can be dried out by placing the packets in an oven at 120°C for 16 hours (desiccant only, not the CR5000).

Campbell Scientific offers two enclosures for housing a CR5000 and peripherals. The fiberglass enclosures are classified as NEMA 4X (water-tight, dust-tight, corrosion-resistant, indoor and outdoor use). A 1.25" diameter entry/exit port is located at the bottom of the enclosure for routing cables and wires. The enclosure door can be fastened with the hasp for easy access, or with the two supplied screws for more permanent applications. The white plastic inserts at the corners of the enclosure must be removed to insert the screws. Both enclosures are white for reflecting solar radiation, thus reducing the internal enclosure temperature.

The Model ENC 12/14 fiberglass enclosure houses the CR5000 and one or more peripherals. Inside dimensions of the ENC 12/14 are 14"x12"x5.5", outside dimensions are 18"x13.5"x8.13" (with brackets); weight is 11.16 lbs.

The Model ENC 16/18 fiberglass enclosure houses the CR5000 and several peripherals. Inside dimensions of the ENC 16/18 are 18"x16"x8¾", outside dimensions are 18½"x18¾"x10½" (with brackets); weight is 18 lbs.

1.2 Power Requirements

The CR5000 operates at a nominal 12 VDC. Below 11.0 V or above 16 volts the CR5000 does not operate properly.

The CR5000 is diode protected against accidental reversal of the positive and ground leads from the battery. Input voltages in excess of 18 V may damage the CR5000 and/or power supply. A transzorb provides transient protection by limiting voltage at approximately 20 V.

System operating time for the batteries can be determined by dividing the battery capacity (amp-hours) by the average system current drain. The CR5000 typically draws 1.5 mA in the sleep state (with display off), 4.5 mA with a 1 Hz sample rate, and 200 mA with a 5 kHz sample rate.

1.3 CR5000 Power Supplies

The CR5000 may be purchased with either a rechargeable lead acid battery or with a low profile case without a battery.

While the CR5000 has a wide operating temperature range (-40 to +85°C optional), the lead acid battery base is limited to -40 to +60°C. **Exceeding this range will degrade battery capacity and lifetime and could also cause permanent damage.**

1.3.1 CR5000 Lead Acid Battery BASE

Temperature range: -40° to +60°C
Charging voltage: 17 to 24 VDC or 18 V RMS AC

NOTE In normal operation a charging source should be connected to the base at all times. The CR5000 stops measuring at ~11 V. Battery life is shortened when discharged below 10.5 V.

The CR5000 includes a 12 V, 7.0 amp-hour lead acid battery, an AC transformer (18 V RMS AC), and a temperature compensated charging circuit with a charge indicating LED (Light Emitting Diode). An AC transformer or solar panel should be connected to the base at all times. The charging source powers the CR5000 while float charging the lead acid batteries. The internal lead acid battery powers the datalogger if the charging source is interrupted. The lead acid battery specifications are given in Table 1.3-1.

The leads from the charging source connect to a wiring terminal plug on the side of the base. Polarity of the leads to the connector does not matter. A transzorb provides transient protection to the charging circuit. A sustained input voltage in excess of 40V will cause the transzorb to limit voltage.

The red light (LED) on the base is on during charging with 17 to 24 VDC or 18 V RMS AC. The switch turns power to the CR5000 on or off. Battery charging still occurs when the switch is off.

Should the lead acid batteries require replacement, consult Figure 1.3-1 for wiring.

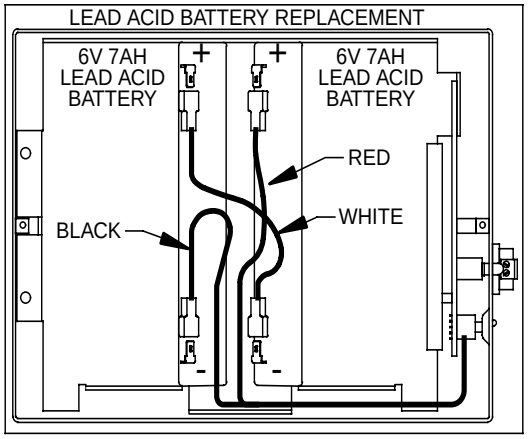


FIGURE 1.3-1. Lead Acid Battery Wiring

Monitor the power supply using datalogger Instruction "Battery". Incorporate this instruction into data acquisition programs to keep track of the state of the power supply. If the system voltage level consistently decreases through time, some element(s) of the charging system has failed. Battery measures the voltage at the CR5000 electronics, not the voltage of the lead acid battery. The measured voltage will normally be about 0.3 V less than the voltage at the internal or external 12 V input. This voltage drop is on account of a Schottkey diode. External power sources must be disconnected from the CR5000 to measure the actual lead acid battery voltage.

TABLE 1.3-1. CR5000 Rechargeable Battery and AC Transformer Specifications

Lead Acid Battery	
Battery Type	Yuasa NP7-6
Float Life @ 25ºC	3 years minimum
Capacity	7.0 amp-hour
Shelf Life, full charge	6 months
Charge Time (AC Source)	40 hr full charge, 20 hr 95% charge
Operating temperature	-40°C to 60°C
AC Transformer	
Input:	120 VAC, 60 Hz
Isolated Output:	18 VAC 1.2 Amp

There are inherent hazards associated with the use of sealed lead acid batteries. Under normal operation, lead acid batteries generate a small amount of hydrogen gas. This gaseous by-product is generally insignificant because the hydrogen dissipates naturally before build-up to an explosive level (4%) occurs. However, if the batteries are shorted or overcharging takes place, hydrogen gas may be generated at a rate sufficient to create a hazard. Campbell Scientific recommends:

1. A CR5000 equipped with standard lead acid batteries should NEVER be used in applications requiring INTRINSICALLY SAFE equipment.

2. A lead acid battery should not be housed in a gas-tight enclosure.

1.3.2 Low Profile CR5000

The low profile CR5000 option is not supplied with a battery base. See Section 1.5 and 1.6 for external power connection considerations.

1.4 Solar Panels

Auxiliary photovoltaic power sources may be used to maintain charge on lead acid batteries.

When selecting a solar panel, a rule-of-thumb is that on a stormy overcast day the panel should provide enough charge to meet the system current drain (assume 10% of average annual global radiation, kW/m^2). Specific site information, if available, could strongly influence the solar panel selection. For example, local effects such as mountain shadows, fog from valley inversion, snow, ice, leaves, birds, etc. shading the panel should be considered.

Guidelines are available from the Solarex Corporation for solar panel selection called "DESIGN AIDS FOR SMALL PV POWER SYSTEMS". It provides a method for calculating solar panel size based on general site location and system power requirements. If you need help in determining your system power requirements contact Campbell Scientific's Marketing Department.

1.5 Direct Battery Connection to the CR5000 Wiring Panel

Any clean, battery backed 11 to 16 VDC supply may be connected to the 12 V and G connector terminals on the front panel. When connecting external power to the CR5000, first, remove the green power connector from the CR5000 front panel. Insert the positive 12 V lead into the right-most terminal of the green connector. Insert the ground lead in the left terminal. Double check polarity before plugging the green connector into the panel.

Diode protection exists so that an external battery can be connected to the green G and 12 V power input connector, without loading or charging the internal batteries. The CR5000 will draw current from the source with the largest voltage. When power is connected through the front panel, switch control on the standard CR5000 power supplies is by-passed (Figure 1.7-1).

1.6 Vehicle Power Supply Connections

1.6.1 CR5000 with Battery Base

The best way to power a CR5000 with battery base from a vehicle’s 12 V power system is to use the DCDC18R to input the power to the CR5000’s charger input (Figure 1.6-1). With this configuration the CR5000’s batteries are charged when the vehicle power is available. When the vehicle’s voltage is too low or off, the CR5000 is powered from its internal batteries.

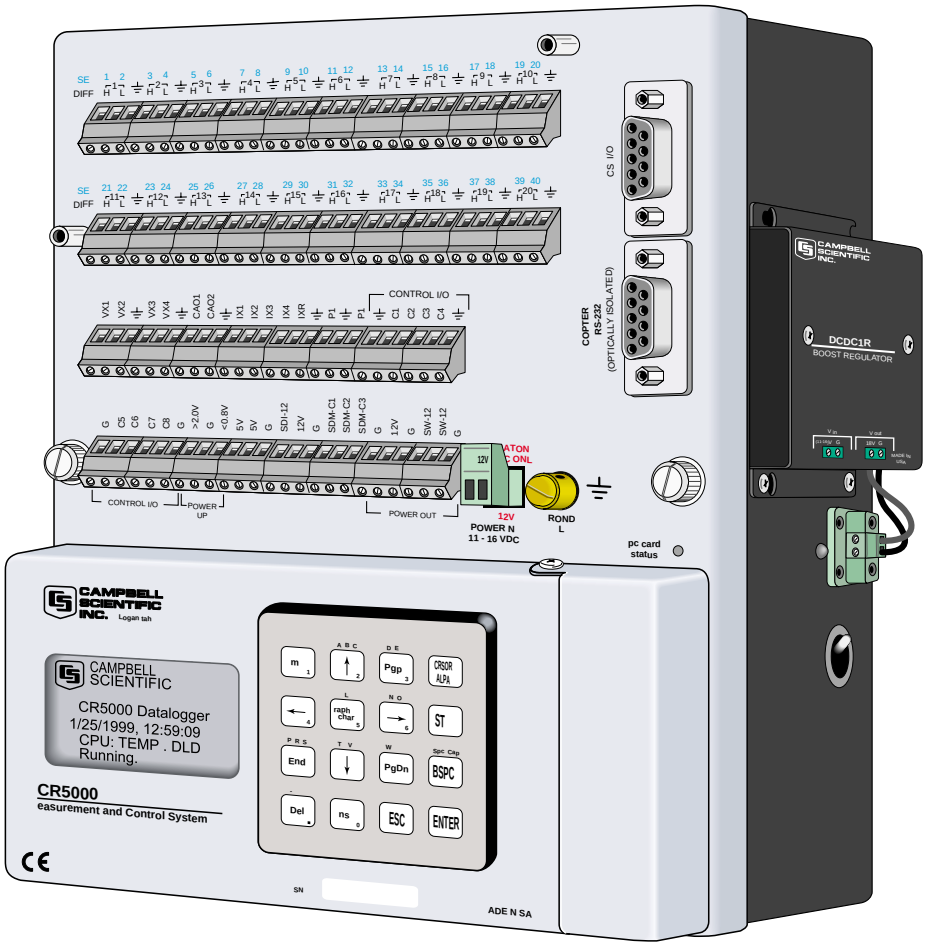


FIGURE 1.6-1. CR5000 with DCDC18R

It is also possible to use the vehicle's 12 V power system as the primary supply for a CR5000 with a battery base (Figure 1.6-2). When a vehicle's starting motor is engaged, the system voltage drops considerably below the 11 volts needed for uninterrupted datalogger function. Diodes in the CR5000 in series with the 12 V Power In connector allow the battery base to supply the needed voltage during motor start. The diodes also prevent the separate power systems of the CR5000 and vehicle from attempting to charge each other. **Because this configuration does not charge the CR5000 batteries, it is not recommended.**

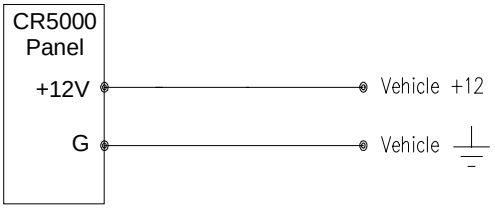


FIGURE 1.6-2. Alternate Connect on to Vehicle Power Supply

1.6.2 CR5000 with Low Profile Base (No Battery)

If a CR5000 without batteries is to be powered from the 12 Volts of a motor vehicle, a second 12 V supply is required. When the starting motor of a vehicle with a 12 V electrical system is engaged, the voltage drops considerably below 11 V, which would cause the CR5000 to stop measurement every time the vehicle is started. The second 12 V supply prevents this malfunction. Figure 1.6-3 shows connecting the two supplies to a CR5000 without a battery base. The diodes allows the vehicle to power the CR5000 without the second supply attempting to power the vehicle.

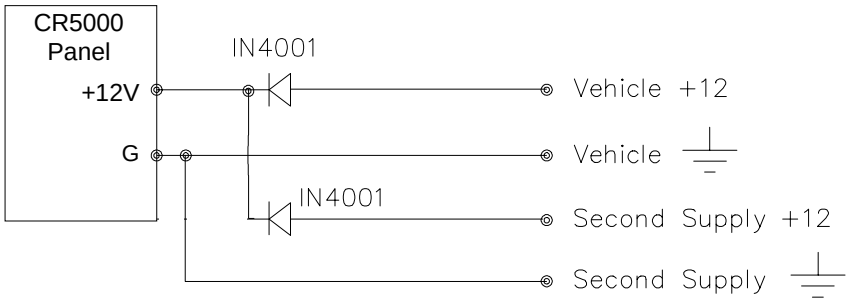


FIGURE 1.6-3. Connecting CR5000 without Battery Base to Vehicle Power Supply

1.7 CR5000 GROUNDING

Grounding of the CR5000 and its peripheral devices and sensors is critical in all applications. Proper grounding will ensure the maximum ESD (electrostatic discharge) protection and higher measurement accuracy.

1.7.1 ESD Protection

An ESD (electrostatic discharge) can originate from several sources. However, the most common, and by far potentially the most destructive, are primary and secondary lightning strikes. Primary lightning strikes hit the datalogger or sensors directly. Secondary strikes induce a voltage in power lines or sensor wires.

The primary devices for protection against ESD are gas-discharge tubes (GDT). All critical inputs and outputs on the CR5000 are protected with GDTs or transient voltage suppression diodes. The GDTs fire at 150 V to allow current to be diverted to the earth ground lug. To be effective, the earth ground lug must be properly connected to earth (chassis) ground. As shown in Figure 1.7-1, the power ground and signal ground are independent lines until joined inside the CR5000.

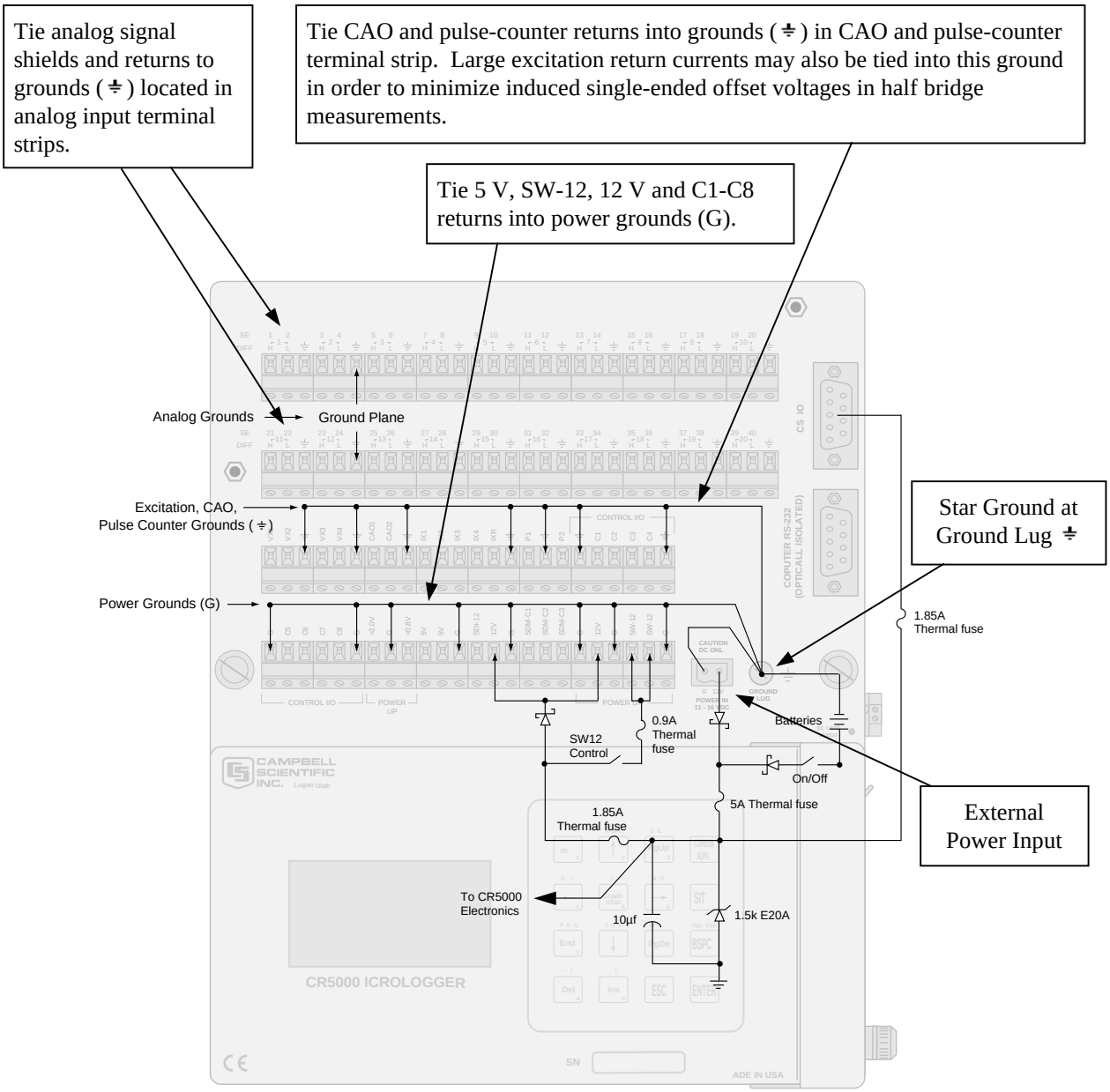


FIGURE 1.7-1. Schematic of CR5000 Grounds

The 9-pin serial I/O ports on the CR5000 are another path for transients to enter and damage the CR5000. Communications devices such a telephone or short-haul modem lines should have spark gap protection. Spark gap protection is often an option with these products, so it should always be requested when ordering. The spark gaps for these devices must be connected to either the CR5000 earth ground lug, the enclosure ground, or to the earth (chassis) ground.

A good earth (chassis) ground will minimize damage to the datalogger and sensors by providing a low resistance path around the system to a point of low potential. Campbell Scientific recommends that all dataloggers be earth (chassis) grounded. All components of the system (dataloggers, sensors, external power supplies, mounts, housings, etc.) should be referenced to one common earth (chassis) ground.

In the field, at a minimum, a proper earth ground will consist of a 6 to 8 foot copper sheathed grounding rod driven into the earth and connected to the CR5000 Ground Lug with a 12 AWG wire. In low conductive substrates, such as sand, very dry soil, ice, or rock, a single ground rod will probably not provide an adequate earth ground. For these situations, consult the literature on lightning protection or contact a qualified lightning protection consultant. An excellent source of information on lightning protection can be located via the web at http://www.polyphaser.com.

In vehicle applications, the earth ground lug should be firmly attached to the vehicle chassis with 12 AWG wire or larger.

In laboratory applications, locating a stable earth ground is not always obvious. In older buildings, new cover plates on old AC sockets may indicate that a safety ground exists when in fact the socket is not grounded. If a safety ground does exist, it is good practice to verify that it carries no current. If the integrity of the AC power ground is in doubt, also ground the system through the buildings, plumbing or another connection to earth ground.

1.7.2 Effect of Grounding on Measurements: Common Mode Range

The common mode range is the voltage range, relative to the CR5000 ground, within which both inputs of a differential measurement must lie in order for the differential measurement to be made correctly. Common mode range for the CR5000 is ±5.0 V. For example, if the high side of a differential input is at 2 V and the low side is at 0.5 V relative to CR5000 ground, a measurement made on the ±5.0 V range would indicate a signal of 1.5 V. However, if the high input changed to 6 V, the common mode range is exceeded and the measurement may be in error.

Common mode range may be exceeded when the CR5000 is measuring the output from a sensor which has its own grounded power supply and the low side of the signal is referenced to the sensors power supply ground. If the CR5000 ground and the sensor ground are at sufficiently different potentials, the signal will exceed the common mode range. To solve this problem, the sensor power ground and the CR5000 ground should be connected, creating one ground for the system.

In a laboratory application, where more than one AC socket may be used to power various sensors, it is not safe to assume that the power grounds are at the same potential. To be safe, the ground of all the AC sockets in use should be tied together with a 12 AWG wire.

1.7.3 Effect of Grounding on Single-Ended Measurements

Low-level single-ended voltage measurements can be problematic because of ground potential fluctuations. The grounding scheme in the CR5000 has been designed to eliminate ground potential fluctuations due to changing return currents from 12 V, SW-12, 5 V, and the control ports. This is accomplished by utilizing separate signal grounds (⏚) and power grounds (G). To take advantage of this design, observe the following grounding rule:

NOTE Always connect a device's ground next to the active terminal associated with that ground.

Examples:

1. Connect 5 Volt, 12 Volt, and control grounds to G terminals.
2. Connect excitation grounds to the closest ⏚ terminal on the excitation terminal block.
3. Connect the low side of single-ended sensors to the nearest ⏚ terminal on the analog input terminal blocks.
4. Connect shield wires to the nearest ⏚ terminal on the analog input terminal blocks.

If offset problems occur because of shield or ground leads with large current flow, tying the problem leads into the ⏚ terminals next to the excitation, CAO, and pulse-counter channels should help. Problem leads can also be tied directly to the ground lug to minimize induced single-ended offset voltages.

1.8 Powering Sensors and Peripherals

The CR5000 is a convenient source of power for sensors and peripherals requiring a continuous or semi-continuous 5 VDC or 12 VDC source. The CR5000 has 2 continuous 12 Volt (12V) supply terminals, 2 switched 12 Volt (SW-12) supply terminals, and 2 continuous 5 Volt (5V) supply terminals. Voltage on the 12V and SW-12 terminals will change with the CR5000 supply voltage. The 5V terminal is regulated and will always remain near 5 Volts (±4%)so long as the CR5000 supply voltage remains above 11 Volts. The 5V terminal is not suitable for resistive bridge sensor excitation. Table 1.8-1 shows the current limits of the 12 Volt and 5 Volt ports. Table 1.8-2 shows current requirements for several CSI peripherals. Other devices normally have current requirements listed in their specifications. Current drain of all peripherals and sensors combined should not exceed current sourcing limits of the CR5000.

Table 1.8-1 Current Sourcing Limits

Terminals	Current Source Limit
SW12	< 900 mA @ 20°C < 729 mA @ 40°C < 630 mA @ 50°C < 567 mA @ 60°C < 400 mA @ 80°C
12V + SW12	< 1.85 A @ 20°C < 1.50 A @ 40°C < 1.30 A @ 50°C < 1.17 A @ 60°C < 0.85 A @ 80°C
5V + CSI/O	< 200 mA

Make certain that the primary source of power for the CR5000 can sustain the current drain for the period of time required. Contact a CSI applications engineer for help in determining a power budget for applications that approach the limits of a given power supply's capabilities. Be particularly cautious about any application using solar panels and cellular telephone or radio, applications requiring long periods of time between site visits, or applications at extreme temperatures.

TABLE 1.8-2. Typical Current Drain for Some CR5000 Peripherals

Peripheral	Typical Current Drain (mA) Quiescent	Active
AM25T	.5	1
COM100	.5	1.8
COM200 Phone Modem	0.0012	140
SDM-INT8	0.4	6.5

1.9 Controlling Power to Sensors and Peripherals

Controlling power to an external device is a common function of the CR5000.

Many devices can conveniently be controlled with the SW-12 (Switched 12 Volt) terminals on the CR5000. Table 1.8-1 shows the current available from SW-12 port.

Applications requiring more control channels or greater power sourcing capacity can usually be satisfied with the use of Campbell Scientific's A21REL-12 Four Channel Relay Driver, A6REL-12 Six Channel Relay Driver, SDM-CD16AC 16 Channel AC/DC Relay Module, or by using the control (C1-C8) ports as described in Section 1.9.1

1.9.1 Use of Digital I/O Ports for Switching Relays

Each of the eight digital I/O ports can be configured as an output port and set low or high (0 V low, 5 V high) using the PortSet or WriteIO instructions. A digital output port is normally used to operate an external relay driver circuit because the port itself has a limited drive capability (2.0 mA minimum at 3.5 V).

Figure 1.9-1 shows a typical relay driver circuit in conjunction with a coil driven relay which may be used to switch external power to some device. In this example, when the control port is set high, 12 V from the datalogger passes through the relay coil, closing the relay which completes the power circuit to a fan, turning the fan on.

In other applications it may be desirable to simply switch power to a device without going through a relay. Figure 1.9-2 illustrates a circuit for switching external power to a device without going through a relay. If the peripheral to be powered draws in excess of 75 mA at room temperature (limit of the 2N2907A medium power transistor), the use of a relay (Figure 1.9-1) would be required.

Other control port activated circuits are possible for applications with greater current/voltage demands than shown in Figures 1.9-1 and 2. For more information contact a Campbell Scientific applications engineer.

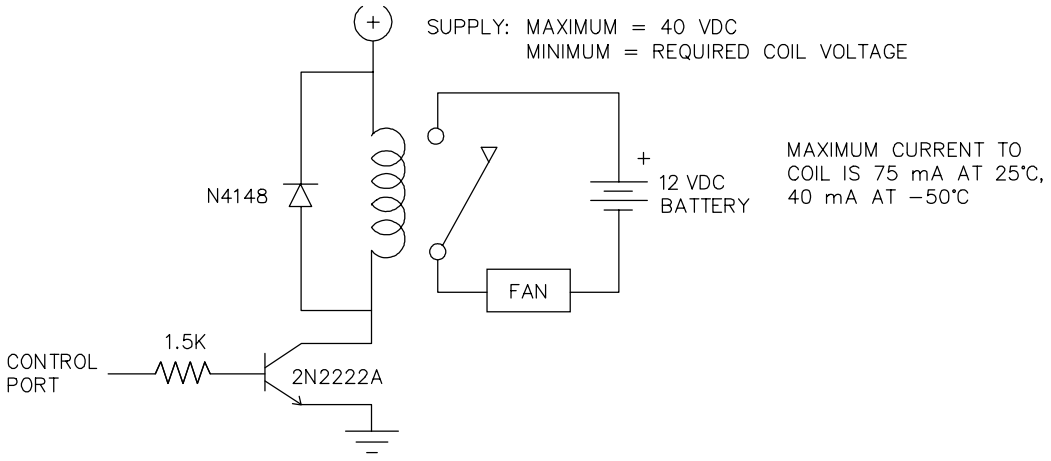


FIGURE 1.9-1. Relay Driver Circuit with Relay

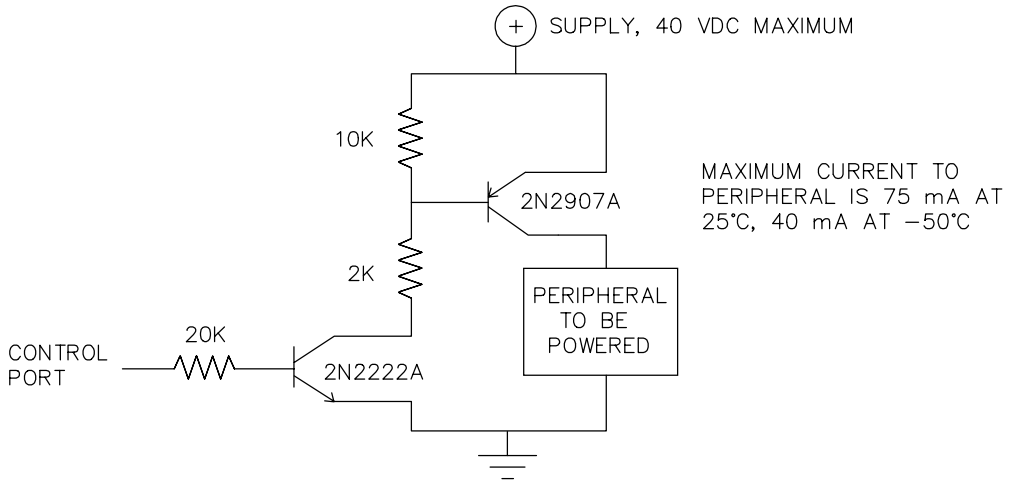


FIGURE 1.9-2. Power Switching without Relay

1.10 Maintenance

The CR5000 power supplies require a minimum of routine maintenance.

When not in use, the rechargeable supply should be stored in a cool, dry environment with the AC charger active.

1.10.1 Desiccant

The CR5000 is shipped with desiccant to reduce humidity. Desiccant should be changed periodically. To prevent corrosion in uncontrolled or condensing atmospheres, the CR5000 must be placed inside a weather tight instrument enclosure with desiccant. Do not completely seal the enclosure if lead acid batteries are present. Hydrogen gas generated by the batteries may build up to an explosive concentration.

1.10.2 Replacing the Internal Battery

CAUTION Misuse of the lithium battery or installing it improperly can cause severe injury. Fire, explosion, and severe burn hazard! Do not recharge, disassemble, heat above 100°C (212°F), solder directly to the cell, incinerate, nor expose contents to water.

The CR5000 contains a lithium battery that operates the clock and SRAM when the CR5000 is not powered. The CR5000 does not draw any power from the lithium battery while it is powered by a 12 VDC supply. In a CR5000 stored at room temperature, the lithium battery should last approximately 10 years (less at temperature extremes). Where the CR5000 is powered most or all of the time the lithium cell should last much longer.

While powered from an external source, the CR5000 measures the voltage of the lithium battery daily. This voltage is displayed in the status table (Section 1.6) A new battery will have approximately 3.6 volts. The CR5000 Status Table has a "Lithium Battery" field. This field is either "True" (battery is good) or "False" (replace battery). If the lithium cell is removed or allowed to discharge below the safe level, the CR5000 will still operate correctly while powered. Without the lithium battery, the clock will reset and data will be lost when power is removed.

A replacement lithium battery can be purchased from Campbell (part number 13497). Table 1.10-1 lists the specifications of the battery.

Table 1.10-1 CR5000 Lithium Battery Specifications

Model	Tadiran	TL-5955 (3.6 V)
Capacity		1650 mAh
self discharge rate		1%/year @ 20°C
Diameter		14.5 mm
Length		33.5 mm
Operating temperature range		-55°C to 85°C

The CR5000 must be partially disassembled to replace the lithium cell.

The battery is replaced as shown in Figures 1.10-4 to 1.10-6. The battery is held in place by a band clamp. It can be removed by gently prying the band from the sides of the battery holder.

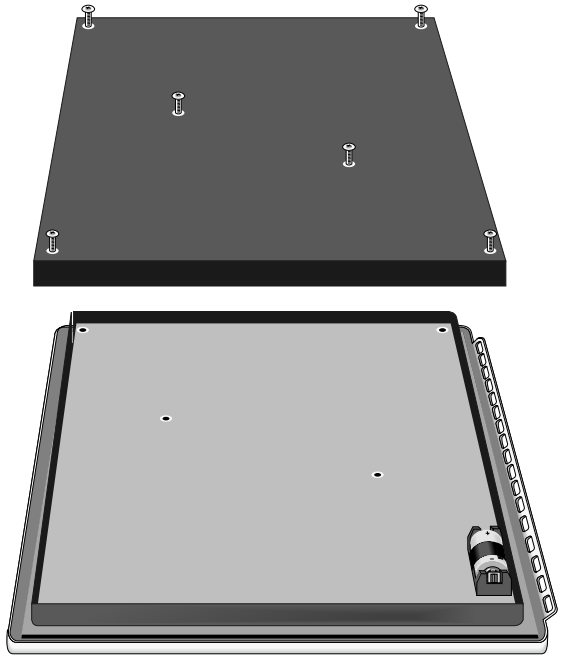


FIGURE 1.10-4. Removal of CR5000 back plate and lithium battery location.

The new cell is placed into the battery holder, observing the polarity markings on the holder. Replace the band clamp, ensuring that both ends snap securely into the battery holder.

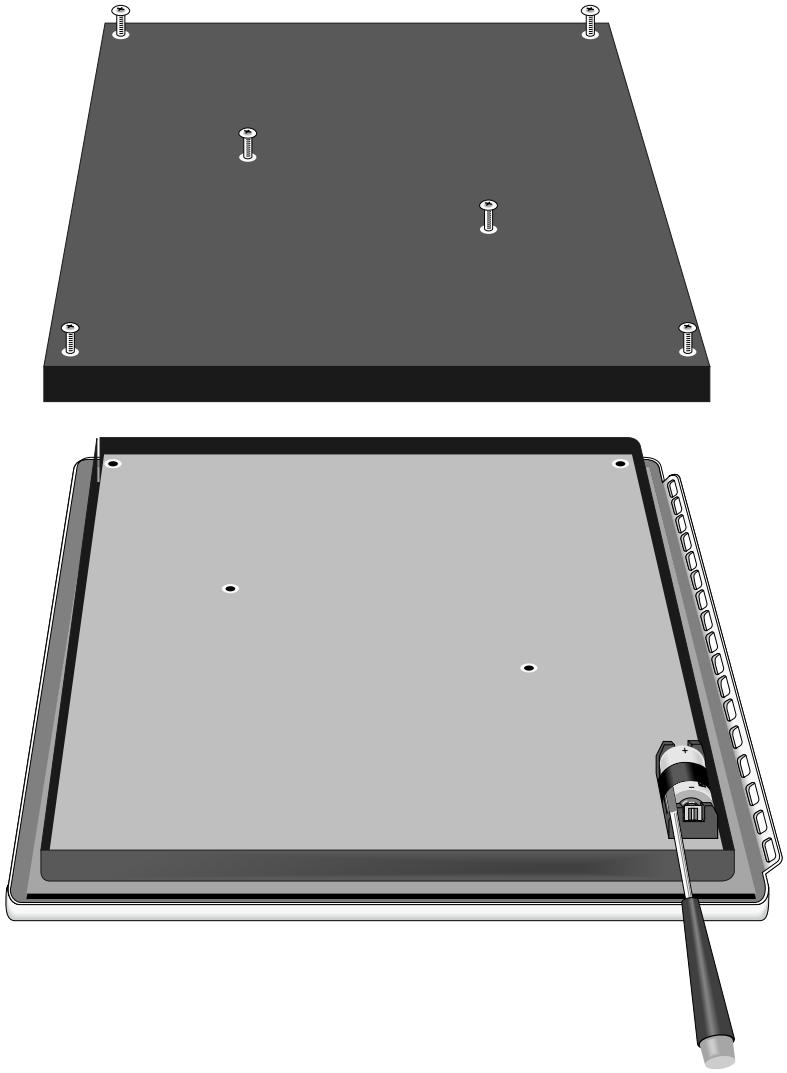


FIGURE 1.10-5. Loosening of band clamp.

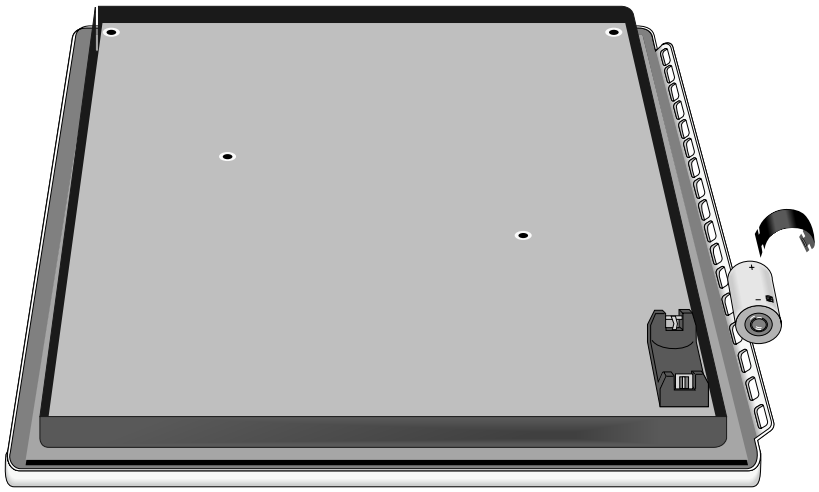


FIGURE 1.11-6. Removal of band clamp and battery.

Section 2. Data Storage and Retrieval

The CR5000 can store individual measurements or it may use its extensive processing capabilities to calculate averages, maxima, minima, histograms, FFTs, etc., on periodic or conditional intervals. Data are stored in tables. For simplicity, the PC9000 program generator allows a maximum of three data tables (in the native language up to 30 data tables can be created). The number of tables and the values to output in each table are selected when running the program generator (Overview) or when writing a datalogger program directly (Sections 4 – 9).

2.1 Data Storage in CR5000

There are two areas for data storage on the CR5000:

Internal Static RAM

PCMCIA PC Card

Internal RAM is used as either the sole storage area for a data table or as a buffer area when data are sent to PC card.

When the CR5000 gets a request for data that is stored on a PC card, the CR5000 only looks for the data in the PC card when the oldest data are requested or if the data are not available in internal RAM.

In the CRBASIC program, the DataTable instruction sets the size of the data table or buffer area. A data table can be stored in a PC card by including the **CardOut** instruction within the data table declaration. A maximum of 30 tables can be created by the program.

2.1.1 Internal Static RAM

Internal RAM is used as either the sole storage area for a data table or as a buffer area when data are sent to a PC card. The only limit on the number of tables is the available memory. Internal RAM is battery backed. Data remain in memory when the CR5000 is powered down under program control. Data in RAM are erased when a different program is loaded and run.

There are 2 Mbytes of SRAM. Some of this is used by the operating system and for program storage. The rest is available for data storage. When a new program is compiled, the CR5000 checks that there is adequate space in SRAM for the data tables; a program that requests more space than is available will not run.

2.1.2 PCMCIA PC Card

The CR5000 has a built in PC card slot for a Type I, Type II, or Type III PCMCIA card. PCMCIA PC Cards allows expanding the CR5000's storage capacity. SRAM and ATA cards are supported. A program can send a maximum of 30 data tables to PC cards.

When a new program is compiled that sends data to the PC card, the CR5000 checks if a card is present and if the card has adequate space for the data tables. If the card has adequate space, the tables will be allocated and the CR5000 will start storing data to them. If there is no card or if there is not enough space, the CR5000 will warn that the card is not being used and will run the program, storing the data in SRAM only. When a card with enough available memory is inserted the CR5000 will create the data tables on the card and store the data that is accumulated in SRAM (Section 2.3.4).

Data stored on cards can be retrieved through the communication link to the CR5000 or by removing the card and inserting it in a PC card slot in a computer. The PCMCIA interface is much faster than the communication link. With large files transferring the PC card is faster than collecting the data over the link.

The CR5000 uses an MS DOS format for the PC cards. Cards can be formatted in a PC or in the CR5000.

2.2 Internal Data Format

TABLE 2.2-1 CR5000 DATA TYPES

Data Type	Size	Range	Resolution
LONG	4 bytes	-2,147,483,648 to +2,147,483,647	1 bit (1)
IEEE4	4 bytes	1.8 E -38 to 1.7 E 38	24 bits (about 7 digits)
FP2	2 bytes	-7999 to +7999	13 bits (about 4 digits)

Data are stored internally in a binary format. Variables and calculations are performed internally in IEEE 4 byte floating point with some operations calculated in double precision. There are two data types used to store data: IEEE4 four byte floating point and Campbell Scientific two byte floating point (FP2). The data format is selected in the instruction that outputs the data. Within the CR5000, time is stored as integer seconds and nanoseconds into the second since midnight, the start of 1990. While IEEE 4 byte floating point is used for variables and internal calculations, FP2 is adequate for most stored data. Campbell Scientific 2 byte floating point provides 3 or 4 significant digits of resolution, and requires half the memory space as IEEE 4 byte floating point (2 bytes per value vs 4).

TABLE 2.2-2. Resolution and Range Limits of FP2 Data

Zero	Minimum Magnitude	Maximum Magnitude
0.000	±0.001	±7999.

The resolution of FP2 is reduced to 3 significant digits when the first (left most) digit is 8 or greater (Table 2.2-2). Thus, it may be necessary to use IEEE4 output or an offset to maintain the desired resolution of a measurement. For example, if water level is to be measured and output to the nearest 0.01 foot, the level must be less than 80 feet for low resolution output to display the 0.01 foot increment. If the water level is expected to range from 50 to 90 feet the data could either be output in high resolution or could be offset by 20 feet (transforming the range to 30 to 70 feet).

TABLE 2.2-3 FP2 Decimal Location	
Absolute Value	**Decimal Location**
0 - 7.999	X.XXX
8 - 79.99	XX.XX
80 - 799.9	XXX.X
800 - 7999.	XXXX.

2.3 Data Collection

Data can be transferred into a computer using PC9000 via a communications link or by transferring a PC card from the PC9000 to the computer. There are three ways to collect data via a link to the CR5000 using the PC9000 software. The **collect** menu is used to collect any or all stored data and is used for most archival purposes.

On each of the RealTime screens, there is a **"write file"** check box. Data stored to the table while the box is checked are also stored in a file on the PC.Logger Files under the Tools menu has the option of retrieving a file from a PC card. This can be used to retrieve a data file.

When the CR5000 is used without a computer in the field, or large data files are collected on a PC card, the **PC card** can be transported to the computer with the data on it.

The format of the data files on the PC card is different than the data file formats created by PC9000 when the collect or write file options are used. Data files retrieved from the Logger Files screen or read directly from the PC card generally need to be converted into another format to be used (Section 2.3.4.2).

2.3.1 The Collect Menu

When Retrieve Data is selected in the Collect menu, PC9000 displays the Collect Data dialog box (Figure 2.3.1). The station name (may be entered by the user when a program is downloaded) is retrieved from the connected CR5000 and shown at the top.

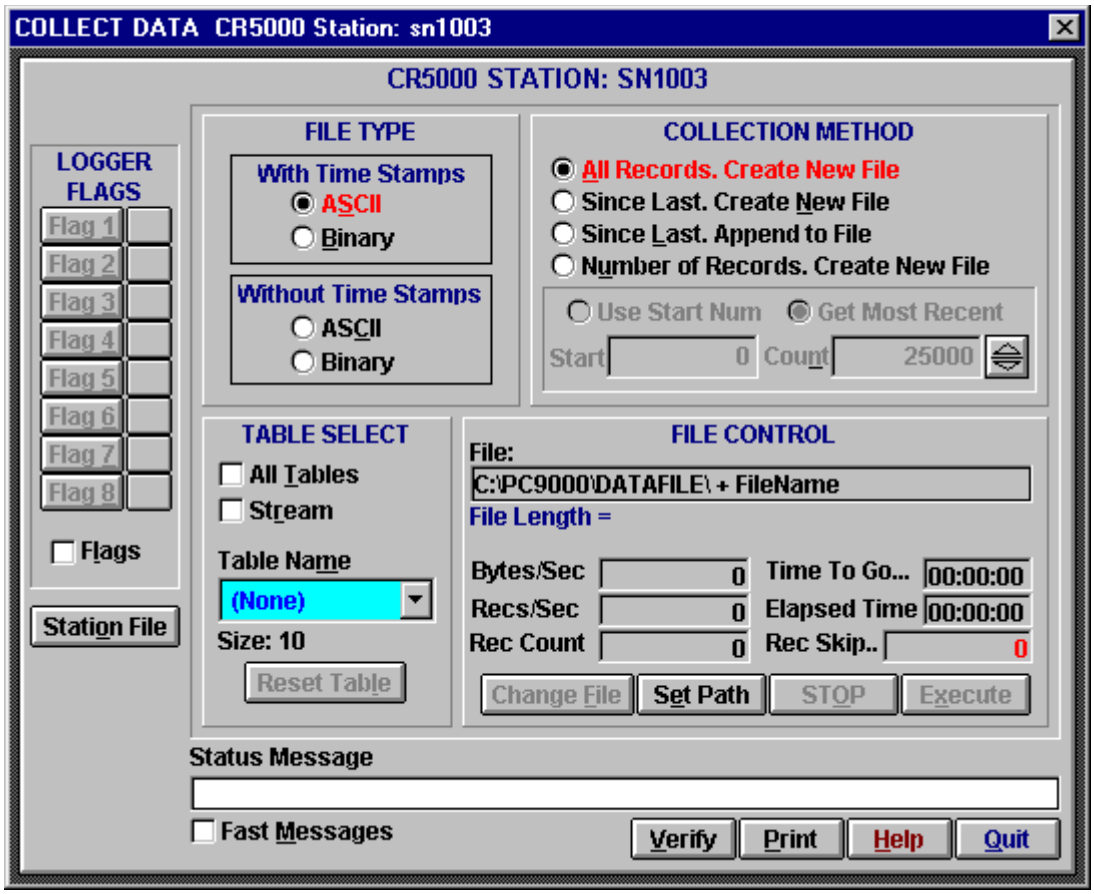


FIGURE 2.3.1. Collect Data Dialog Box

2.3.1.1 File Type

ASCII With Time – Click here to store the data as an ASCII (TOA5, Section 2.4) file. Each record will be date and time stamped.

Binary With Time **–** Click here to store the data as a binary file (TOB1, Section 2.4). Each record will be date and time stamped.

ASCII Without Time – Click here to store the data as an ASCII file (TOA5, Section 2.4). There will be no date and time stamps.

Binary Without Time **–** Click here to store the data as a binary file (TOB1, Section 2.4). There will be no date and time stamps.

2.3.1.2 Collection Method

All Records, Create New File – Collects the entire table stored in the CR5000. PC9000 gets the current record number from the table in the CR5000 and then retrieves the oldest record in the table up to the current record number. The number in the file name is incremented to create the file name in which the data are stored.

Since Last, Create New File – Click here to save new data in a new file. PC9000 searches for the last file with the Root name, gets the last record number from that file, then the current record from the table in the CR5000, and requests all records in between those numbers from the CR5000. The number in the file name is incremented to create the file name in which the data are stored.

Since Last, Append To File – Click here to append retrieved data to the end of the named file. PC9000 searches for the last file with the Root name, gets the last record number from that file, then the current record from the table in the CR5000, and requests all records in between those numbers from the CR5000. The data are appended to the existing file.

Number of Records, Create New File Collects the number of records entered in Num of Recs box. Retrieves that many records back from the current record number. The number in the file name is incremented to create the file name in which the data are stored.

Num of Recs – Enabled when Number of Records, Create New File is checked. Enter the number of records back from the current record number to retrieve.

2.3.1.3 Table Selection

All Tables – When the All Tables box is checked, all data tables except the Public and Status tables are collected when collection is executed. The data from each table are stored in a file with the table name and increment number (see Table naming). This is a convenient method of collecting all data from the CR5000. The first time data are collected, all data is checked and the file type and collection method are selected. PC9000 remembers the settings, and on subsequent collections the operator only needs to click on execute.

Stream – acts the same as Write file for the selected Table Name (Section 2.3.2).

Table Name – When "All tables" is not checked, a single data table can be selected for collection. The Table Name box is used to select the table to be retrieved.

Reset Table – Resetting a data table erases all data in the table and sets the record number back to 0. Unless the table is configured as fill and stop by the CR5000 program, it is not necessary to reset the table because the "Since Last" collection option can be used to get only the new data. If the table is configured as fill and stop, it stops collecting data once full and must be reset before more data can be collected. Use with caution.

2.3.1.4 File Control

The default naming for a file stored to disk is to use the data table name appended with a 2 digit number and the extension **.DAT**. If the table name is longer than 6 characters, it is truncated. For example, the table name EVENTS is stored as EVENTS00.DAT. A table named CYLTEMP is stored as CYLTEM00.DAT.

When the file collection options that create a new file are used, each time a table is collected, the 2 digit number is incremented (e.g., EVENTS00.DAT, EVENTS01.DAT, EVENTS03.DAT ...). PC9000 searches the selected directory and adds 1 to the number of the highest numbered file of the matching name to create the new name.

When the new data are to be appended to the existing file, PC9000 searches the selected directory for the highest numbered file of the matching name, and appends the data to that file.

Change File – Press the Change File button to change the name of the file to be stored on disk. This is not possible when "All Tables" is selected.

Set Path – Press here to select a different disk or directory to write the files to.

EXECUTE – Press here to begin collecting data.

2.3.1.5 Status Messages

TABLE SIZE – Shows the size (in records) of the table highlighted in the Table Name box above.

COLLECTION RANGE – Displays the range of records to be collected. More records than the last number in this range may actually be retrieved.

LOGGER MESSAGE – Displays messages from the CR5000.

2.3.2 RealTime Write File

This feature is provided to allow the user to start and stop collecting data for some event without leaving the real-time window. Check this box to write the current table to a file in the computer. Writing begins with the current record and continues until the Write File box is unchecked or until the window is closed.

This collection method requires that the PC is connected to the CR5000 while the data are collected. Because the beginning and end points of the data file are roughly determined by when the box is checked, this is best suited to collecting data when the user rather than the measurements determine when data should be collected.

The bottom line of the screen will periodically display the current record being written. The name of the file written will be the first six characters of the table name plus 2 digits and an extension of .DAT. If the table name is MAIN then the first file created will be named MAIN00.DAT. The next file will be named MAIN01.DAT and so on. It takes a little time to open the file so be sure it is opened in advance of the event you want to store.

The file is written to every 1 second so it is important the table size be large enough to store sufficient data between writes. During each write operation, the data may not be updated on the screen but this will not effect the stored data.

2.3.3 Logger Files Retrieve

Logger Files under the PC9000 tools menu allows the user to check the programs stored in CPU Flash memory and the files stored on the PCMCIA cards. Any of the files shown in logger files can be copied to the computer by highlighting the file and pressing the retrieve button. Data files in the CR5000 CPU and Flash memory are not shown.

The retrieved data file is stored on the computer in the same form that it was stored on the PC card (TOB2). This format generally needs to be converted to another format for analysis (Section 2.3.4)

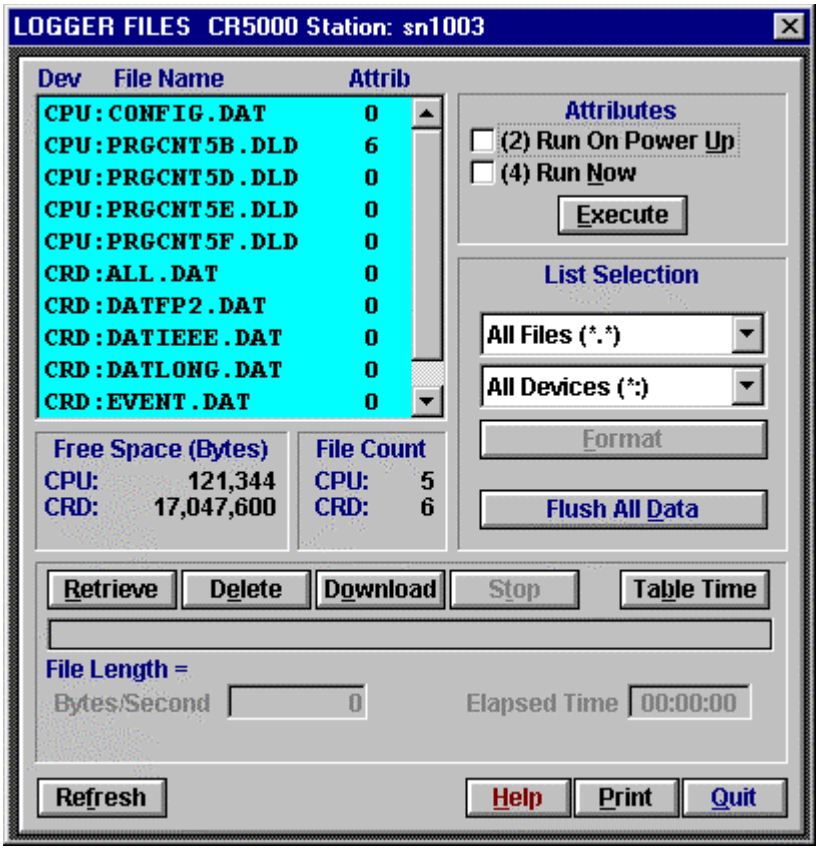


FIGURE 2.3-2. Logger Files Dialog Box

2.3.4 Via PCMCIA PC Card

When the CR5000 is used without a computer in the field, or large data files are collected on a PC card, the **PC card** can be transported to the computer with the data on it.

2.3.4.1 Inserting a PC Card

A card inserted in the PCMCIA slot when no program is running or when a program is running that does not use the PC card does not cause a response

from the CR5000. When a new program is compiled that sends data to the PC card, the CR5000 checks if a card is present and if the card has adequate space for the data tables. If the card has adequate space, the tables will be allocated and the CR5000 will start storing data to them.

When the running program sends data to the card, and a card is inserted, the CR5000 will detect the card and display a message.

A card that has no data tables files with the same names as those created by the program causes the message:

New Card. Start Storing?
No
Yes

If the card has existing data table files that match those created by the program there is the message:

Tables on Card Must be Reset. Proceed?
No
Yes
Same Card

The "No" option [do not start storing data to the card] allows using the CR5000 to check the card and to erase files or to format the card if necessary.

The "Yes" option is to start storing data right away, resetting any tables with the same name that exist on the card. This is the option that is generally used when a blank (but formatted) card is inserted or when a card is reinserted after transferring data to a computer.

The "Same Card" option is only available if the data table header matches the program exactly and the program has already run with a card. This option allows removing the card to read it and then returning it to the CR5000 and having the new data appended to the data already on the card. If you choose to use this option be aware that if the CR5000 overwrites its buffer while the card is out, there will be a gap in the data on the card.

2.3.4.2 Removing Card from CR5000

The PC Card Status LED just above the PC card door is lit when the card is being written to.

CAUTION Removing a card while it is active can cause garbled data and can actually damage the card. Do not switch off the power while the cards are present and active.

To remove a card, use the keyboard display to go to the PcCard menu; move the cursor to "Remove Card: and press Enter. The Status will show "You may now remove the card". Remove the card. The card will be reactivated if not removed.

When the PC card is inserted in a computer, the data files can be copied to another drive or used directly from the PC card just as one would from any other disk. In most cases, however, it will be necessary to convert the file format before using the data.

2.3.4.3 Converting File Format

The CR5000 stores data on the PC card in TOB2 Format. TOB2 is a binary format that incorporates features to improve reliability of the PC Cards. TOB2 allows the accurate determination of each record's time without the space required for individual time stamps.

When TOB2 files are converted to another format, the number of records may be greater or less than the number requested in the data table declaration. There are always at least two additional frames of data allocated. When the file is converted these will result in additional records if no lapses occurred. If more lapses occur than were anticipated, there may be fewer records in the file than were allocated.

PC9000's file converter will convert TOB1 or TOB2 files to other formats. The Convert Data Files option is in the File Menu. The options for TOB2 appear after the name of the file to convert has been selected (Figure 2.3-3.)

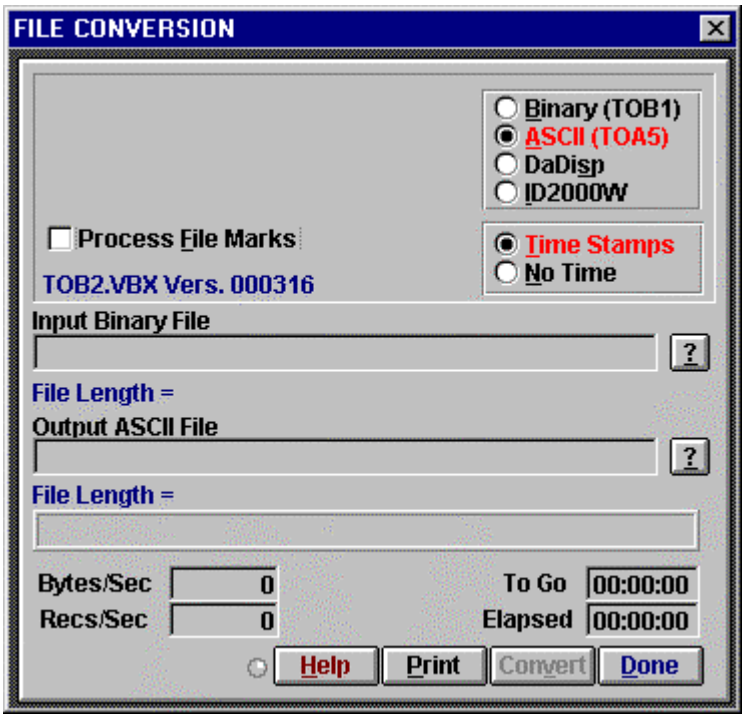


FIGURE 2.3-3 File Conversion Dialog Box

```
"File Format","Station","Logger","Serial No.","OS Ver","DLD File","DLD Sig","Table Name"
"TIMESTAMP","RECORD","Field Name","Field Name","Field Name"
"TS","RN","Field Units","Field Units","Field Units"
"","","Processing","Processing","Processing"
"Field Data Type","Field Data Type","Field Data Type","Field Data Type","Field Data Type"
timestamp,record number,field data,field data,field data,
```

FIGURE 2.4.1 Header Information

2.4 Data Format on Computer

The format of the file stored on disk can be either ASCII or Binary depending on the file type selected in the collect data dialog box. Files collected from a real time window are always stored in ASCII format.

2.4.1. Header Information

Every data file stored on disk has an ASCII header at the beginning. The header gives information on the format, datalogger and program used to collect the data. Figure 2.4.1 is a sample header where the text in the header is a generic name for the information contained in the header. The entries are described following the figure.

File Format
The format of the file on disk. TOA5 is an ASCII format. TOB1 is a Binary format. This information is used by the historical graphing and file conversion functions of PC9000.

Station Name
The station name set in the logger that the data was collected from.

Logger Model
The datalogger model that the data was collected from.

Logger Serial Number
The serial number of the logger that the data was collected from. This is the serial number of the CR5000 CPU.

Operating System Version
The version of the operating system in the logger that the data was collected from.

DLD File
The name of the DLD file that was running when the data were created.

DLD Signature
The signature of the DLD file that created the data.

Table Name
The data table name.

Field Name
The name of the field in the data table. This name is created by the CR5000 by appending underscore (_) and a three character mnemonic for the output processing.

Field Units
The units for the field in the data table. Units are assigned in the program with the units declaration.

Field Processing
The output processing that was used when the field was stored.

Smp = Sample

Max = Maximum

Min = Minimum

Avg = Average

Field Data Type
This header line is only in TOB1 binary format and identifies the data type for each of the fields in the data table.

UINT4 = Unsigned 4 byte integer

IEEE4 = 4 byte floating point

Time Stamp
This field is the date and time stamp for this record. It indicates the time, according to the logger clock, that each record was stored.

Record Number
This field is the record number of this record. The number will increase up to 2E32 and then start over with zero. The record number will also start over at zero if the table is reset.

Field Data
This is the data for each of the fields in the record.

2.4.2 TOA5 ASCII File Format

The following is a sample of a file collected as ASCII with time stamps.

```
"TOA5","Bob's9K","CR5000","1048575","1.00","EXPLDAT.DLD","4339","Temp"
"TIMESTAMP","RECORD","RefTemp_Avg","TC_Avg(1)","TC_Avg(2)","TC_Avg(3)","TC_Avg(4)"
"TS","RN","degC","degC","degC","degC","degC"
"","","Avg","Avg","Avg","Avg","Avg"
"1995-09-19 14:31:43.84",458,29.94,25.6,25.36,25.48,25.4
"1995-09-19 14:31:43.85",459,29.93,25.6,25.36,25.41,25.35
```

The following is an example of how the above data might look when imported into a spread sheet.

TOA5	Bob's9K	CR5000	1048575	1.00	EXPLDAT.DLD	4339	Temp
TIMESTAMP	RECORD	RefTemp_Avg	TC_Avg(1)	TC_Avg(2)	TC_Avg(3)	TC_Avg(4)	
TS	RN	degC	degC	degC	degC	degC	
		Avg	Avg	Avg	Avg	Avg	
1995-09-19 14:31:43.84	458	29.94	25.6	25.36	25.48	25.4	
1995-09-19 14:31:43.85	459	29.93	25.6	25.36	25.41	25.35	

This is the same data table collected as ASCII without time stamps

"TOA5","Bob's9K","CR5000","1048575","1.00","EXPLDAT.DLD","4339","Temp"
"RefTemp_Avg","TC_Avg(1)","TC_Avg(2)","TC_Avg(3)","TC_Avg(4)"
"degC","degC","degC","degC","degC"
"Avg","Avg","Avg","Avg","Avg"
29.94,25.6,25.36,25.48,25.4
29.93,25.6,25.36,25.41,25.35

And again, an example of how the above data might look when imported into a spread sheet.

TOA5	Bob's9K	CR5000	1048575	1.00	EXPLDAT.DLD	4339	Temp
RefTemp_Avg	TC_Avg(1)	TC_Avg(2)	TC_Avg(3)	TC_Avg(4)			
degC	degC	degC	degC	degC			
Avg	Avg	Avg	Avg	Avg			
29.94	25.6	25.36	25.48	25.4			
29.93	25.6	25.36	25.41	25.35			

2.4.2 TOB1 Binary File Format

This is a sample of a file collected as Binary with time stamps.

TOB1,Bob's9K,CR5000,1048575,1.00,EXPLDAT.DLD,4339,Temp
SECONDS,NANOSECONDS,RECORD,RefTemp_Avg,TC_Avg(1),TC_Avg(2),TC_Avg(3),TC_Avg(4)
SECONDS,NANOSECONDS,RN,degC,degC,degC,degC,degC
,,,Avg,Avg,Avg,Avg,Avg
UINT4,UINT4,UINT4,IEEE4,IEEE4,IEEE4,IEEE4,IEEE4
(data lines are binary and not directly readable)

This is an example of binary without time stamps.

TOB1,Bob's9K,CR5000,1048575,1.00,EXPLDAT.DLD,4339,Temp
RefTemp_Avg,TC_Avg(1),TC_Avg(2),TC_Avg(3),TC_Avg(4)
degC,degC,degC,degC,degC
Avg,Avg,Avg,Avg,Avg
IEEE4,IEEE4,IEEE4,IEEE4,IEEE4
(data lines are binary and not directly readable)

2.4.3 TOB2 Binary File Format

The TOB2 binary format has a header similar to the other formats. TOB2 data is stored in fixed size “ frames” that generally contain a number of records. The size of the frames is a function of the record size. The frames are time

stamped, allowing the calculation of time stamps for their records. If there is a lapse in periodic interval records that does not occur on a frame boundary, an additional time stamp is written within the frame and its occurrence noted in the frame boundary. This additional time stamp takes up space that would otherwise hold data.

When TOB2 files are converted to another format, the number of records may be greater or less than the number requested in the data table declaration. There are always at least two additional frames of data allocated. When the file is converted these will result in additional records if no lapses occurred. If more lapses occur than were anticipated, there may be fewer records in the file than were allocated.

Section 3. CR5000 Measurement Details

3.1 Analog Voltage Measurement Sequence

The CR5000 measures analog voltages with either an integrate and hold or a sample and hold analog to digital (A/D) conversion. The A/D conversion is made with a 16 bit successive approximation technique which resolves the signal voltage to approximately one part in 60,000 of the full scale range. The maximum conversion rate is 5000 per second or one measurement every 200 µs (10,000 measurements per second on a single channel).

The timing of CR5000 measurements is precisely controlled. The measurement schedule is determined at compile time and loaded into memory. This schedule sets interrupts that drive the measurement task.

Using two different voltage measurement instructions with the same voltage range takes the same measurement time as using one instruction with two repetitions. (This is not the case in the CR10X, 21X, CR23X and CR7 dataloggers where there is always a setup time for each instruction.)

There are four parameters in the measurement instructions that may vary the sequence and timing of the measurement. These are options to measure and correct the ground offset on single-ended measurements each time measurements are made (**MeasOfs**), reverse the high and low differential inputs (**RevDiff**), to set the time to allow the signal to settle between switching to a channel and making a measurement (**SettlingTime**), and the length of time to integrate a measurement (**Integ**), and to reverse the polarity of excitation voltage (**RevEx**).

3.1.1 Voltage Range

The CR5000 has 5 fixed voltage ranges and autorange. The 16 bit A/D has a resolution of 1 part in 2^{16} (65,536). To allow for some overrange capabilities the A/D is applied to a range approximately 9% greater than the Full Scale Range resulting in the 1 part in 60,000 resolution over the FSR. For example, on the ±20 mV range the full scale range is 40 mV [20 - (-20)] and the resolution is two thirds of a microvolt; 0.04 / 0.000000667 = 60,000. The smaller the voltage range, the better the absolute resolution. In general, a measurement should use the smallest fixed voltage range that will accommodate the full scale output of the sensor being measured. If the voltage exceeds the range, the CR5000 indicates the overrange by returning Not-A-Number (NAN) for the measurement.

For signals that do not fluctuate too rapidly, AutoRange allows the CR5000 to automatically choose the voltage range to use. AutoRange causes the CR5000 to make two measurements. The first measurement determines the range to use. It is made with no integration on the ±5000 mV range. The second measurement is made on the appropriate range using the integration specified in the instruction. Both measurements use the settling time programmed in the instruction. AutoRange optimizes resolution but takes longer than a measurement on a fixed range, because of the two measurements required.

An AutoRange measurement will return Not-A-Number if the voltage exceeds the range picked by the first measurement. To avoid problems with a signal on the edge of a range, AutoRange selects the next larger range when the signal exceeds 90% of a range.

AutoRange is very good for a signal that occasionally exceeds a particular range, for example, a Type J thermocouple that most of the time will be less than 360 °C (±20 mV range) but will occasionally see temperatures as high as 400 °C (±50 mV range, Table 3.4-2). AutoRange should not be used for rapidly fluctuating signals, particularly those whose signal traverses several voltage ranges rapidly because of the possibility that the signal could change ranges between the range check and the actual measurement.

3.1.2 Reversing Excitation or the Differential Input

Reversing the excitation polarity or the differential input are techniques to cancel voltage offsets that are not part of the signal. For example, if there is a +5 µV offset in the measurement circuitry, a 5 mV signal will be measured as 5.005 mV. When the input is reversed, the measurement will be -4.995 mV. Subtracting the second measurement from the first and dividing by 2 gives the correct answer: 5.005-(-4.995)=10, 10/2=5. Most offsets are thermocouple effects caused by temperature gradients in the measurement circuitry or wiring.

Reversing the excitation polarity cancels voltage offsets in the sensor, wiring, and measurement circuitry. One measurement is made with the excitation voltage with the polarity programmed and a second measurement is made with the polarity reversed. The excitation "on time" for each polarity is exactly the same to ensure that ionic sensors do not polarize with repetitive measurements.

Reversing the inputs of a differential measurement cancels offsets in the CR5000 measurement circuitry and improves common-mode rejection. One measurement is made with the high input referenced to the low input and a second with the low referenced to the high.

3.1.3 Measuring Single-Ended Offset

The single-ended offset is a voltage offset on a single-ended input. It is measured by internally switching the input to ground and measuring the voltage. When a single-ended measurement is made this offset is corrected for in the calibration. The offset can either be measured automatically as part of the background calibration or as part of the measurement sequence each time the measurement is made (adding to the time to make the measurement). When the offset is measured in the measurement sequence, the offset is measured once prior to completing all of the instruction reps.

The **MeasOfs** parameter in instructions that make single-ended voltage measurements is used to force the offset measurement. In most cases the background calibration is adequate. Additional accuracy can be gained by making the offset measurement with each measurement instruction when the offset is changing rapidly as it would during when the CR5000 is undergoing rapid temperature swings.

3.1.4 SettlingTime

When the CR5000 switches to a new channel or switches on the excitation for a bridge measurement, there is a finite amount of time required for the signal to reach its true value. Delaying between setting up a measurement (switching to the channel, setting the excitation) and making the measurement allows the signal to settle to the correct value. The default settling times are the minimum required for the CR5000 to settle to within its accuracy specifications. Additional time is necessary when working with high sensor resistances or long lead lengths (higher capacitance). Using a longer settling time increases the time required for each measurement. Section 3.3 goes into more detail on determining an adequate settling time.

When the CR5000 Reverses the differential input or the excitation polarity it delays the same settling time after the reversal as it does before the first measurement. Thus there are two delays per channel when either **RevDiff** or **RevEx** is used. If both **RevDiff** and **RevEx** are selected, there are four measurement segments, positive and negative excitations with the inputs one way and positive and negative excitations with the inputs reversed. The CR5000 switches to the channel:

sets the excitation, delays, **measures**,
reverses the excitation, delays, **measures**,
reverses the excitation, reverses the inputs, delays, **measures**,
reverses the excitation, delays, **measures**.

Thus there are four delays per channel measured. The CR5000 processes the measurement segments into the single value it returns for the measurement.

3.1.5 Integration

Integration is used to reduce the noise included in a measurement. The CR5000 may use a combination of analog and digital integration.

For the fastest measurements, there is a zero integration measurement. This measurement does not integrate. The signal at a precise instant is sampled and this voltage is held for A/D conversion.

With analog integration, the input signal is integrated for a precise period of time. The integrated value is held for the A/D conversion. There are three possible analog integration times 20 ms, 16.67 ms and 250 µs. The 20 ms (1/50 second) and 16.667 ms (1/60 second) are available to integrate out the effects of noise from 50 or 60 Hz AC power sources.

An integration time in microseconds is specified as part of the measurement instruction. An integration time of 0 selects the sample-and-hold, 250 selects the 250 µs integration, 16667 or "_60 Hz" selects the 60 Hz rejection (16667 µs), and 20000 or "_50 Hz" selects 50 Hz rejection (20000 µs).

In addition to the analog integrations, it is possible to average a number of the sample-and-hold or 250 µs integration measurements. The A/D conversions are made as rapidly as possible: every 100 µs for the samples and every 500 µs for the 250 µs integrations. If the integration time specified is 250 µs or a

multiple of 500 µs, the CR5000 will repeat 250 µs integration measurements every 500 µs throughout the integration interval. If the integration time specified is 100 µs or 200 µs, the CR5000 makes one or two samples in the integration interval. The average of these measurements is stored as the result of the measurement.

The random noise level is decreased by the square root of the number of measurements made. For example, the input noise on the ±5000 mV range with one 250 µs integration is 120 µV RMS; entering 2000 µs for the integration (four measurements) will cut this noise in half (120/(√4)=60).

The integration time specified in the measurement instruction is used for each segment of the measurement. Thus, if reversing the differential input or reversing the excitation is specified, there will be two integrations per channel; if both reversals are specified, there will be four integrations.

3.2 Single Ended and Differential Voltage Measurements

A single-ended voltage measurement is made on a single input which is measured relative to ground. A differential measurement measures the difference in voltage between two inputs.

NOTE There are two sets of channel numbers on the analog channels. Differential channels (1-20) have two inputs: high (H) and low (L). Either the high or low side of a differential channel can be used for a single ended measurement. The single-ended channels are numbered 1-40.

Because a single ended measurement is referenced to CR5000 ground, any difference in ground potential between the sensor and the CR5000 will result in an error in the measurement. For example, if the measuring junction of a copper-constantan thermocouple, being used to measure soil temperature, is not insulated and the potential of earth ground is 1 mV greater at the sensor than at the point where the CR5000 is grounded, the measured voltage would be 1 mV greater than the thermocouple output, or approximately 25 ºC high. Another instance where a ground potential difference creates a problem is where external signal conditioning circuitry is powered from the same source as the CR5000. Despite being tied to the same ground, differences in current drain and lead resistance result in different ground potential at the two instruments. For this reason, a differential measurement should be made on an analog output from the external signal conditioner. Differential measurements MUST be used when the inputs are known to be different from ground, such as the output from a full bridge.

Common mode range

In order to make a differential measurement, the inputs must be within the CR5000 common mode range of ±5 V. The common mode range is the voltage range, relative to CR5000 ground, within which both inputs of a differential measurement must lie, in order for the differential measurement to

be made. For example, if the high side of a differential input is at 4 V and the low side is at 3 V relative to CR5000 ground, there is no problem. A measurement made on the ±5000 mV range will return 1000 mV. However, if the high input is at 5.8 V and the low input is at 4.8 V, the measurement can not be made because the high input is outside of the ±5 V common mode range (the CR5000 will indicate the overrange by returning not-a-number (NAN)).

Sensors that have a floating output or are not referenced to ground through a separate connection may need to have one side of the differential input connected to ground to ensure the signal remains within the common mode range.

Problems with exceeding common mode range may be encountered when the CR5000 is used to read the output of external signal conditioning circuitry if a good ground connection does not exist between the external circuitry and the CR5000. When operating where AC power is available, it is not always safe to assume that a good ground connection exists through the AC wiring. If a CR5000 is used to measure the output from a laboratory instrument (both plugged into AC power and referencing ground to outlet ground), it is best to run a ground wire between the CR5000 and the external circuitry. Even with this ground connection, the ground potential of the two instruments may not be at exactly the same level, which is why a differential measurement is desired.

A differential measurement has the option of reversing the inputs to cancel offsets as described above.

NOTE Sustained voltages in excess of ±16 V will damage the CR5000 circuitry.

3.3 Signal Settling Time

Whenever an analog input is switched into the CR5000 measurement circuitry prior to making a measurement, a finite amount of time is required for the signal to stabilize at it's correct value. The rate at which the signal settles is determined by the input settling time constant which is a function of both the source resistance and input capacitance.

The CR5000 delays after switching to a channel to allow the input to settle before initiating the measurement. The default delays used by the CR5000 depend on the integration used and the voltage range. For the sample-and-hold, the default delay is 100 µs on the ±5000, ±1000, ±200, and ±50 mV ranges and 200 µs on the ±20 mV range. The default delay is 200 µs for 250 µs integrations and 3 ms for 50 Hz or 60 Hz rejection integrations. This settling time is the minimum required to allow the input to settle to the resolution specification.

Additional wire capacitance associated with long sensor leads can increase the settling time constant to the point that measurement errors may occur. There are three potential sources of error which must settle before the measurement is made:

1. The signal must rise to its correct value.

2. A small transient caused by switching the analog input into the measurement circuitry must settle.

3. When a resistive bridge measurement is made using a switched excitation channel, a larger transient caused when the excitation is switched must settle.

3.3.1 Minimizing Settling Errors

When long lead lengths are mandatory, the following general practices can be used to minimize or measure settling errors:

1. DO NOT USE WIRE WITH PVC INSULATED CONDUCTORS. PVC has a high dielectric which extends input settling time.

2. Where possible run excitation leads and signal leads in separate shields to minimize transients.

3. When measurement speed is not a prime consideration, additional time can be used to ensure ample settling time. The settling time required can be measured with the CR5000.

3.3.2 Measuring the Necessary Settling Time

The CR5000 can measure the time required for a particular sensor/cable configuration to settle. This is done by allowing the signal to settle the minimum amount of time and then making zero integration measurements every 100 µs. Looking at the series of measurements it is possible to see the settling of the sensor signal. By counting the number of measurements before the signal settles the correct settling time can be determined (100 µs per measurement plus an extra 100 µs if the measurement is made on the 20 mV range where the default settling time is 200 µs).

NOTE This technique for measuring settling time can only be used with instructions that measure only one voltage per repetition. The settling time to use with instructions that make more than one measurement can be determined with an instruction that uses only one measurement. For example, for the BrFull6W determine the settling time with BrFull; for BrHalf3W use BrHalf.

The following example demonstrates measuring the settling time for a differential voltage measurement. If you are not yet familiar with CR5000 programming, you may want to read Section 4 before trying to follow the example.

The series of measurements on the sensor is made with one instruction. Measurement instructions have a repetitions parameter that allow one instruction to repeat the measurement on a number of channels. There is also the capability to repeat measurements on the same channel. When

measurements are repeated on the same channel the settling time is only necessary before the first measurement (as long as the excitation polarity and differential inputs are not reversed).

Before the program to measure the settling time is run, the sensor with the cable that will be used in the installation needs to be connected and the sensor needs to be stabilized. If the sensed value is changing rapidly it will be difficult to separate the settling time from true changes in the value measured. The following program measures the settling time on a full bridge pressure transducer.

```
'CR5000
'Program to measure the settling time on a Pressure Transducer

'Declare the variable array for the 20 Pressure Transducer measurements:
public PT(20)

'Set up the output table for the measurement data:
DataTable (SetlDat,1,5)
  Sample (20,PT(),IEEE4)
EndTable

'The following program repeats the 20 measurements 5 times:
BeginProg
  Scan (1,Sec,3,5)
    'The -1 in the differential channel parameter of BrFull tells
    'the CR5000 to make the 20 measurements on channel 1:
    BrFull (PT(),20,mV50,-1,Vx1,20,5000,False,False,0,0,1.0,0)
    CallTable SetlDat
  NextScan
EndProg
```

The program was run on a Druck water level pressure transducer with 200 feet of cable. The first six measurements of each of the 5 readings are shown in Table 3.3-1. The reading has settled by the second measurement, PT(2), thus a settling time of 200 µs is adequate.

Table 3.3-1. First Six Values of Settling Time Data

PT(1)	PT(2)	PT(3)	PT(4)	PT(5)	PT(6)
0.291257	0.052925	0.053937	0.054274	0.057645	0.056296
0.292269	0.052925	0.053262	0.054948	0.055622	0.056633
0.286538	0.053937	0.054274	0.055959	0.053599	0.054948
0.284515	0.052925	0.052925	0.053599	0.057645	0.055285
0.286875	0.052925	0.052925	0.054611	0.056296	0.057308

3.4 Thermocouple Measurements

A thermocouple consists of two wires, each of a different metal or alloy, which are joined together at each end. If the two junctions are at different temperatures, a voltage proportional to the difference in temperatures is induced in the wires. If the junctions are at the same temperature, there is no voltage. When a thermocouple is used for temperature measurement, the wires are soldered or welded together at the measuring junction. The second

junction, which becomes the reference junction, is formed where the other ends of the wires are connected to the measuring device. (With the connectors at the same temperature, the chemical dissimilarity between the thermocouple wire and the connector does not induce any voltage.) When the temperature of the reference junction is known, the temperature of the measuring junction can be determined by measuring the thermocouple voltage and adding the corresponding temperature difference to the reference temperature.

The CR5000 determines thermocouple temperatures using the following sequence. First the temperature of the reference junction is measured and stored in °C. If the reference junction is the CR5000 analog input terminals, the temperature is measured with the built in thermistor (PanelTemp instruction). The thermocouple measurement instruction measures the thermocouple voltage (TCDiff or TCSE). The thermocouple instruction calculates the voltage that a thermocouple of the type specified would output at the reference junction temperature if its reference junction were at 0 ºC, and adds this voltage to the thermocouple voltage. The temperature of the measuring junction is then calculated from a polynomial approximation of the NIST TC calibrations

3.4.1 Error Analysis

The error in the measurement of a thermocouple temperature is the sum of the errors in the reference junction temperature, the thermocouple output (deviation from standards published in NIST Monograph 175), the thermocouple voltage measurement, and the linearization error (difference between NIST standard and CR5000 polynomial approximations). The discussion of errors which follows is limited to these errors in calibration and measurement and does not include errors in installation or matching the sensor to the environment being measured.

Panel Temperature

A brass bar is just under the CR5000 panel between the two rows of analog input terminals. This bar helps to reduce temperature gradients between the terminals. The panel temperature thermistor is in a depression in the center of this bar.

The thermistor (Betatherm 10K3A1A) has an interchangability specification of 0.1 °C for temperatures between 0 and 70 °C. Below freezing and at higher temperatures this specification is degraded (Figure 3.4.1). Combined with possible errors in completion resistors and the measurement, the accuracy of panel temperature is ±0.25 °C 0 to 40 °C, ±0.5 °C -25 to 50 °C, and ±0.7 °C -40 to 85 °C.

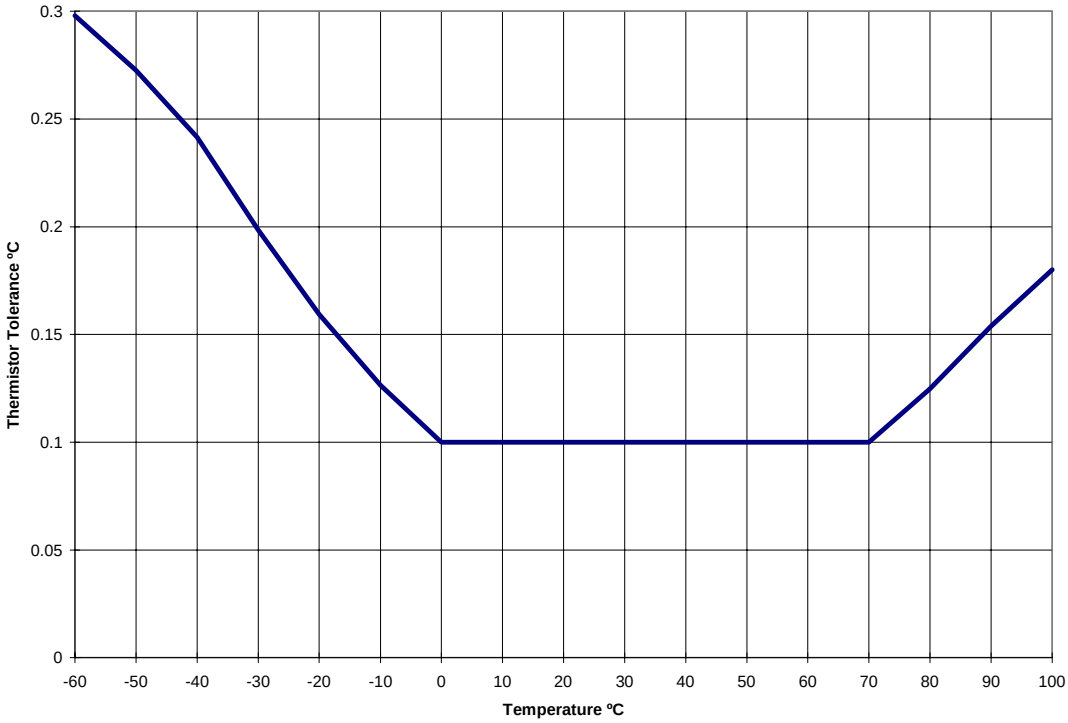


FIGURE 3.4-1. Thermistor Tolerance

The error in the reference temperature measurement is a combination of the error in the thermistor temperature and the difference in temperature between the panel thermistor and the terminals the thermocouple is connected to. The terminal strip cover should always be used when making thermocouple measurements. It insulates the terminals from drafts and rapid fluctuations in temperature as well as conducting heat to reduce temperature gradients. In a typical installation where the CR5000 is in a weather proof enclosure not subject to violent swings in temperature or lopsided solar radiation loading, the temperature difference between the terminals and the thermistor is likely to be less than 0.2 °C.

With an external driving gradient, the temperature gradients on the input panel can be much worse. For example, the CR5000 with the rechargeable battery back was placed in a controlled temperature chamber. Thermocouples in channels at the ends and middle of each analog terminal strip measured the temperature of an insulated aluminum bar outside the chamber. The temperature of this bar was also measured by another datalogger. Differences between the temperature measured by one of the thermocouples and the actual temperature of the bar are due to the temperature difference between the terminals the thermocouple is connected to and the thermistor reference (the figures have been corrected for thermistor errors). Figure 3.4-2 shows the errors when the chamber was changed from -55 to 80 °C in approximately 15 minutes. Figure 3.4-3 shows the results when going from 80 to 25 °C in approximately 45 minutes, a less dramatic change but still greater than would be seen in most natural circumstances. During these rapid changes in temperature, the temperature of panel thermistor will tend to lag behind the terminals because it is buried a bit deeper in the CR5000 and is closer to the

thermal mass of the batteries. Note that the smallest errors are on channels 5 and 16 in the middle of the terminal strips closest to the thermistor.

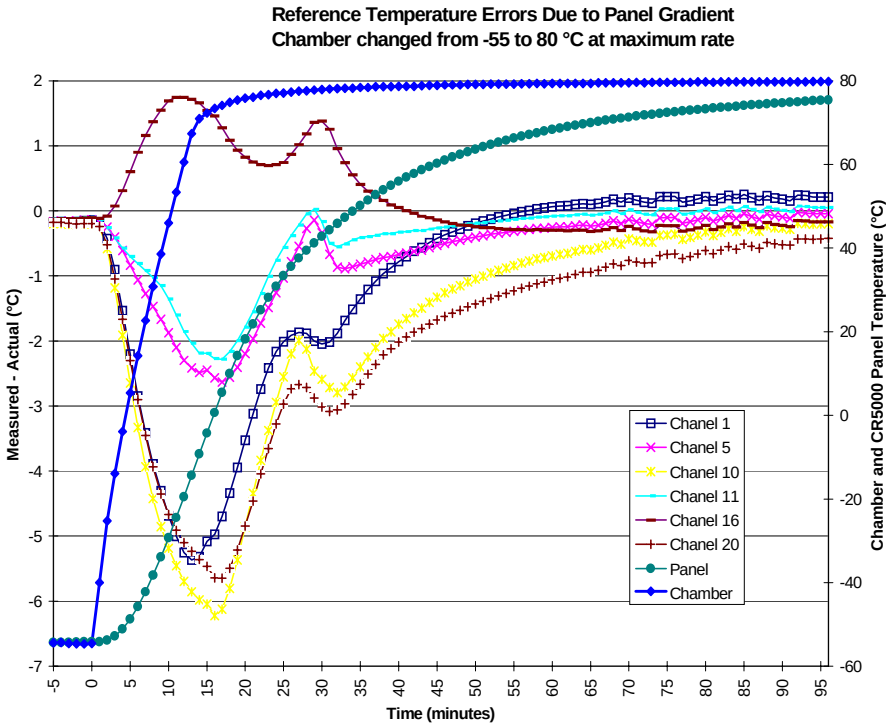


FIGURE 4.3-2. Panel Temperature Gradients During -55 to 80 °C change

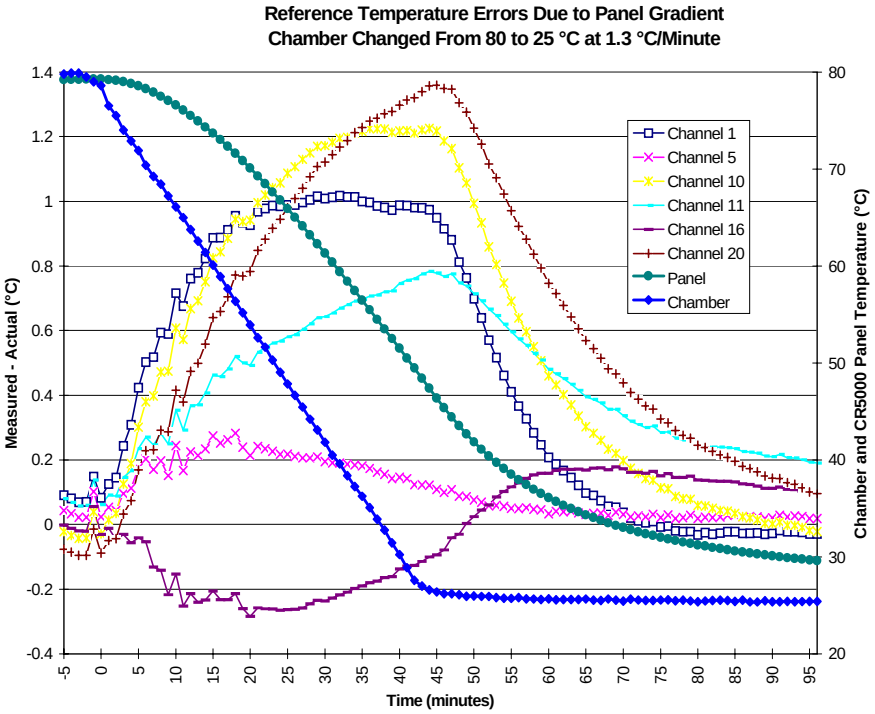


FIGURE 4.3-3. Panel Temperature Gradients During 80 to 25 °C change

Thermocouple Limits of Error

The standard reference which lists thermocouple output voltage as a function of temperature (reference junction at 0ºC) is the National Institute of Standards and Technology Monograph 175 (1993). The American National Standards Institute has established limits of error on thermocouple wire which is accepted as an industry standard (ANSI MC 96.1, 1975). Table 3.4-1 gives the ANSI limits of error for standard and special grade thermocouple wire of the types accommodated by the CR5000.

TABLE 3.4-1. Limits of Error for Thermocouple Wire (Reference Junction at 0ºC)

Thermocouple Type	Temperature Range ºC	Limits of Error (Whichever is greater) Standard	Special
T	-200 to 0	± 1.0ºC or 1.5%	
	0 to 350	± 1.0ºC or 0.75%	± 0.5ºC or 0.4%
J	0 to 750	± 2.2ºC or 0.75%	± 1.1ºC or 0.4%
E	-200 to 0	± 1.7ºC or 1.0%	
	0 to 900	± 1.7ºC or 0.5%	± 1.0ºC or 0.4%
K	-200 to 0	± 2.2ºC or 2.0%	
	0 to 1250	± 2.2ºC or 0.75%	± 1.1ºC or 0.4%
R or S	0 to 1450	± 1.5ºC or 0.25%	± 0.6ºC or 0.1%
B	800 to 1700	± 0.5%	Not Estab.

When both junctions of a thermocouple are at the same temperature there is no voltage produced (law of intermediate metals). A consequence of this is that a thermocouple can not have an offset error; any deviation from a standard (assuming the wires are each homogeneous and no secondary junctions exist) is due to a deviation in slope. In light of this, the fixed temperature limits of error (e.g., ±1.0 °C for type T as opposed to the slope error of 0.75% of the temperature) in the table above are probably greater than one would experience when considering temperatures in the environmental range (i.e., the reference junction, at 0 °C, is relatively close to the temperature being measured, so the absolute error - the product of the temperature difference and the slope error - should be closer to the percentage error than the fixed error). Likewise, because thermocouple calibration error is a slope error, accuracy can be increased when the reference junction temperature is close to the measurement temperature. For the same reason differential temperature measurements, over a small temperature gradient, can be extremely accurate.

In order to quantitatively evaluate thermocouple error when the reference junction is not fixed at 0 ºC, one needs limits of error for the Seebeck coefficient (slope of thermocouple voltage vs. temperature curve) for the various thermocouples. Lacking this information, a reasonable approach is to apply the percentage errors, with perhaps 0.25% added on, to the difference in temperature being measured by the thermocouple.

Accuracy of the Thermocouple Voltage Measurement

The -25 to 50 °C accuracy of a CR5000 differential voltage measurement is specified as ± (0.075% of the measured voltage plus the input offset error of 2 times the basic resolution of the range being used to make the measurement plus 2 µV). The input offset error reduces to the basic resolution if the differential measurement is made utilizing the option to reverse the differential input.

For optimum resolution, the ±20 mV range is used for all but high temperature measurements (Table 3.4-2). The input offset error dominates the voltage measurement error for environmental measurements. A temperature difference of 45 to 65 °C between the measurement and reference junctions is required for a thermocouple to output 2.67 mV, the voltage at which 0.075% of the reading is equal to 2 µV. For example, assume that a type T thermocouple is used to measure a temperature of 45 °C and that the reference temperature is 25 °C. The voltage output by the thermocouple is 830.7 µV. At 45 degrees a type T thermocouple outputs 42.4 µV per ºC. The possible slope error in the voltage measurement is 0.00075x830.7 µV = 0.62 µV or 0.014 ºC (0.62/42.4). The basic resolution on the ±20 mV range is 0.67 µV or 0.01 ºC. The 2 µV offset is an error of 0.047 ºC. Thus, the possible error due to the voltage measurement is 0.081 ºC on a non-reversing differential, or 0.024 ºC with a reversing differential measurement. The value of using a differential measurement with reversing input to improve accuracy is readily apparent.

The error in the temperature due to inaccuracy in the measurement of the thermocouple voltage is worst at temperature extremes, particularly when the temperature and thermocouple type require using the 200 mV range. For example, assume type K (chromel-alumel) thermocouples are used to measure temperatures around 1300 ºC. The TC output is on the order of 52 mV,

requiring the ±200 mV input range. At 1300 ºC, a K thermocouple outputs 34.9 µV per ºC. The possible slope error in the voltage measurement is 0.00075x52 mV = 39 µV or 1.12 ºC (39/34.9). The basic resolution on the 200 mV range is 6.67 µV or 0.19 ºC. Thus, the possible error due to the voltage measurement is 1.56 ºC on a non-reversing differential, or 1.31 ºC with a reversing differential measurement.

TABLE 3.4-2. Voltage Range for maximum Thermocouple resolution

TC Type and temp. range ºC	Temp. range for ±20 mV range	Temp. range for ±50 mV range	Temp. range for ±200 mV range
T -270 to 400	-270 to 400	not used	not used
E -270 to 1000	-270 to 280	-270 to 660	>660
K --270 to 1372	-270 to 610	-270 to 1230	>1230
J -210 to 1200	-210 to 360	-210 to 870	> 870
B 0 to 1820	0 to 1820	not used	not used
R -50 to 1768	-50 to 1680	-50 to 1768	not used
S -50 to 1768	-50 to 1768	not used	not used
N -270 to 1300	-270 to 580	-270 to 1300	not used

When the thermocouple measurement junction is in electrical contact with the object being measured (or has the possibility of making contact) a differential measurement should be made.

Noise on Voltage Measurement

The typical input noise on the ±20 mV range for a differential measurement with no integration and input reversal is 1.8 µV RMS. On a type T thermocouple (approximately 40 µV/ºC) this is 0.05 ºC. Note that this is an RMS value, some individual readings will vary by greater than this. By integrating for 250 µs the noise level is reduced to 0.6 µV RMS.

Thermocouple Polynomial: Voltage to Temperature

NIST Monograph 175 gives high order polynomials for computing the output voltage of a given thermocouple type over a broad range of temperatures. In order to speed processing and accommodate the CR5000's math and storage capabilities, 4 separate 6th order polynomials are used to convert from volts to temperature over the range covered by each thermocouple type. Table 3.4-3 gives error limits for the thermocouple polynomials.

TABLE 3.4-3. Limits of Error on CR5000 Thermocouple Polynomials (Relative to NIST Standards)

TC Type	Range °C			Limits of Error °C
T	**-270**	**to**	**400**	
	-270	to	-200	+ 18 @ -270
	-200	to	-100	± 0.08
	-100	to	100	± 0.001
	100	to	400	± 0.015
J	**-150**	**to**	**760**	± 0.008
	-100	to	300	± 0.002
E	**-240**	**to**	**1000**	
	-240	to	-130	± 0.4
	-130	to	200	± 0.005
	200	to	1000	± 0.02
K	**-50**	**to**	**1372**	
	-50	to	950	± 0.01
	950	to	1372	± 0.04

Reference Junction Compensation: Temperature to Voltage

The polynomials used for reference junction compensation (converting reference temperature to equivalent TC output voltage) do not cover the entire thermocouple range. Substantial errors will result if the reference junction temperature is outside of the linearization range. The ranges covered by these linearizations include the CR5000 environmental operating range, so there is no problem when the CR5000 is used as the reference junction. External reference junction boxes however, must also be within these temperature ranges. Temperature difference measurements made outside of the reference temperature range should be made by obtaining the actual temperatures referenced to a junction within the reference temperature range and subtracting one temperature from the other. Table 3.4-3 gives the reference temperature ranges covered and the limits of error in the linearizations within these ranges.

Two sources of error arise when the reference temperature is out of range. The most significant error is in the calculated compensation voltage, however error is also created in the temperature difference calculated from the thermocouple output. For example, suppose the reference temperature for a measurement on a type T thermocouple is 300 °C. The compensation voltage calculated by the CR5000 corresponds to a temperature of 272.6 °C, a -27.4 °C error. The type T thermocouple with the measuring junction at 290 °C and reference at 300 °C would output -578.7 µV; using the reference temperature of 272.6 °C, the CR5000 calculates a temperature difference of -10.2 °C, a -0.2 °C error. The temperature calculated by the CR5000 would be 262.4 °C, 27.6 °C low.

TABLE 3.4-4. Reference Temperature Compensation Range and Polynomial Error Relative to NIST Standards

TC Type	Range °C	Limits of Error °C
T	-100 to 100	± 0.001
J	-150 to 296	± 0.005
E	-150 to 206	± 0.005
K	-50 to 100	± 0.01

Error Summary

The magnitude of the errors described in the previous sections illustrate that the greatest sources of error in a thermocouple temperature measurement with the CR5000 are likely to be due to the limits of error on the thermocouple wire and in the reference temperature. Errors in the thermocouple and reference temperature linearizations are extremely small, and error in the voltage measurement is negligible.

To illustrate the relative magnitude of these errors in the environmental range, we will take a worst case situation where all errors are maximum and additive. A temperature of 45 °C is measured with a type T (copper-constantan) thermocouple, using the ±20 mV range. The nominal accuracy on this range is 1 µV (0.01% of 10 mV) which at 45 °C changes the temperature by 0.012 °C. The RTD is 25 °C but is indicating 25.1 °C, and the terminal that the thermocouple is connected to is 0.05 °C cooler than the RTD.

TABLE 3.4-5. Example of Errors in Thermocouple Temperature

Source	Error: °C : % of Total Error			
	Single Differential No Integration		Reversing Differential 250 µs Integration	
	ANSI TC Error (1°C)	TC Error 1% Slope	ANSI TC Error (1°C)	TC Error 1% Slope
Reference Temp.	0.15°:11.5%	0.15°:29.8%	0.15°:12.6%	0.15°:38.4%
TC Output	1.0°:76.7%	0.2°:39.8%	1.0°:84%	0.2°:51.2%
Voltage Measurement	0.081°:6.2%	0.081°:16.1%	0.024°:2%	0.024°:6.1%
Noise	0.07°:5.4%	0.07°:13.9%	0.015°:1.2%	0.015°:3.8%
Reference Linearization	0.001°:0.1%	0.001°:0.2%	0.001°:0.1%	0.001°:0.25%
Output Linearization	0.001°:0.1%	0.001°:0.2%	0.001°:0.1%	0.001°:0.25%
Total Error	1.303°:100%	0.503°:100%	1.191°:100%	0.391°:100%

3.4.2 Use of External Reference Junction or Junction Box

An external junction box is often used to facilitate connections and to reduce the expense of thermocouple wire when the temperature measurements are to be made at a distance from the CR5000. In most situations it is preferable to make the box the reference junction in which case its temperature is measured and used as the reference for the thermocouples and copper wires are run from

the box to the CR5000. Alternatively, the junction box can be used to couple extension grade thermocouple wire to the thermocouples, and the CR5000 panel temperature used as the reference. Extension grade thermocouple wire has a smaller temperature range than standard thermocouple wire, but meets the same limits of error within that range. The only situation where it would be necessary to use extension grade wire instead of a external measuring junction is where the junction box temperature is outside the range of reference junction compensation provided by the CR5000. This is only a factor when using type K thermocouples, where the upper limit of the reference compensation linearization is 100 ^{o}C and the upper limit of the extension grade wire is 200 ^{o}C. With the other types of thermocouples the reference compensation range equals or is greater than the extension wire range. In any case, errors can arise if temperature gradients exist within the junction box.

Figure 3.4-1 illustrates a typical junction box. Terminal strips will be a different metal than the thermocouple wire. Thus, if a temperature gradient exists between A and A' or B and B', the junction box will act as another thermocouple in series, creating an error in the voltage measured by the CR5000. This thermoelectric offset voltage is a factor whether or not the junction box is used for the reference. This offset can be minimized by making the thermal conduction between the two points large and the distance small. The best solution in the case where extension grade wire is being connected to thermocouple wire would be to use connectors which clamped the two wires in contact with each other.

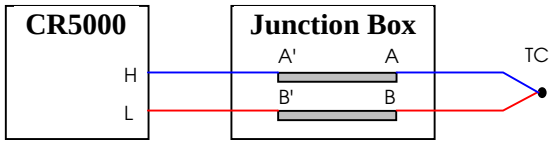


FIGURE 3.4-1. Diagram of Junction Box

An external reference junction box must be constructed so that the entire terminal area is very close to the same temperature. This is necessary so that a valid reference temperature can be measured and to avoid a thermoelectric offset voltage which will be induced if the terminals at which the thermocouple leads are connected (points A and B in Figure 3.4-1) are at different temperatures. The box should contain elements of high thermal conductivity, which will act to rapidly equilibrate any thermal gradients to which the box is subjected. It is not necessary to design a constant temperature box, it is desirable that the box respond slowly to external temperature fluctuations.

Radiation shielding must be provided when a junction box is installed in the field. Care must also be taken that a thermal gradient is not induced by conduction through the incoming wires. The CR5000 can be used to measure the temperature gradients within the junction box.

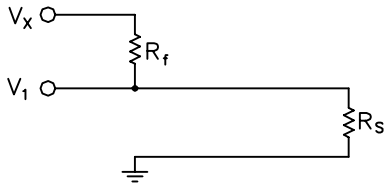
3.5 Bridge Resistance Measurements

There are six bridge measurement instructions included in the standard CR5000 software. Figure 3.5-1 shows the circuits that would typically be measured with these instructions. In the diagrams, the resistors labeled R_s would normally be the sensors and those labeled R_f would normally be fixed resistors. Circuits other than those diagrammed could be measured, provided the excitation and type of measurements were appropriate.

All of the bridge measurements have the option (**RevEx**) to make one set of measurements with the excitation as programmed and another set of measurements with the excitation polarity reversed. The offset error in the two measurements due to thermal emfs can then be accounted for in the processing of the measurement instruction. The excitation channel maintains the excitation voltage or current until the hold for the analog to digital conversion is completed. When more than one measurement per sensor is necessary (four wire half bridge, three wire half bridge, six wire full bridge), excitation is applied separately for each measurement. For example, in the four wire half bridge when the excitation is reversed, the differential measurement of the voltage drop across the sensor is made with the excitation at both polarities and then excitation is again applied and reversed for the measurement of the voltage drop across the fixed resistor.

Calculating the actual resistance of a sensor which is one of the legs of a resistive bridge usually requires additional processing following the bridge measurement instruction. In addition to the schematics of the typical bridge configurations, Figure 3.5-1 lists the calculations necessary to compute the resistance of any single resistor, provided the values of the other resistors in the bridge circuit are known.

BrHalf



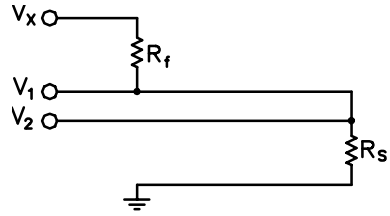
X = result w/mult = 1, offset = 0

$$X = \frac{V_1}{V_x} = \frac{R_s}{R_s + R_f}$$

$$R_s = R_f \frac{X}{1 - X}$$

$$R_f = \frac{R_s(1 - X)}{X}$$

BrHalf3W



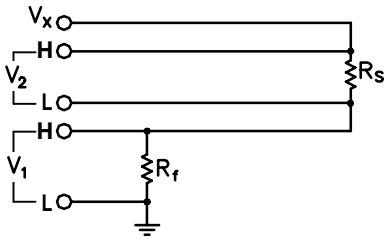
X = result w/mult = 1, offset = 0

$$X = \frac{2V_2 - V_1}{V_X - V_1} = \frac{R_s}{R_f}$$

$$R_s = R_f X$$

$$R_f = R_s / X$$

BrHalf4W



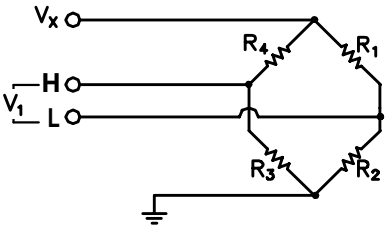
X = result w/mult = 1, offset = 0

$$X = \frac{V_2}{V_1} = \frac{R_s}{R_f}$$

$$R_s = R_f X$$

$$R_f = R_s / X$$

BrFull



X = result w/mult = 1, offset = 0

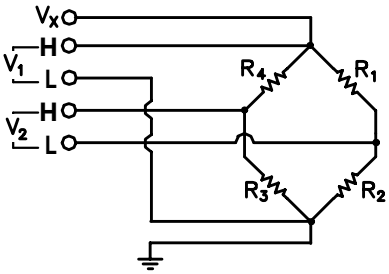
$$X = 1000\frac{V_1}{V_x} = 1000\left(\frac{R_3}{R_3+R_4} - \frac{R_2}{R_1+R_2}\right)$$

$$X_1 = -X/1000 + R_3/(R_3+R_4)$$

$$R_1 = \frac{R_2(1-X_1)}{X_1}$$

$$R_2 = \frac{R_1 X_1}{1-X_1}$$

BrFull6W



X = result w/mult = 1, offset = 0

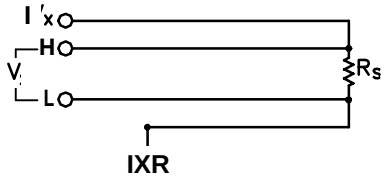
$$X = 1000\frac{V_2}{V_1} = 1000\left(\frac{R_3}{R_3+R_4} - \frac{R_2}{R_1+R_2}\right)$$

$$X_2 = X/1000 + R_2/(R_1+R_2)$$

$$R_3 = \frac{R_4 X_2}{1-X_2}$$

$$R_4 = \frac{R_3(1-X_2)}{X_2}$$

Resistance

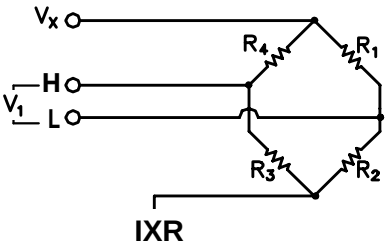


X = result w/mult = 1, offset = 0

$$X = \frac{V}{I_x} = R_s$$

Resistance

used to measure full bridge



X = result w/mult = 1, offset = 0

$$X = \frac{V_1}{I_x} = R_{bridge}\left(\frac{R_3}{R_3+R_4} - \frac{R_2}{R_1+R_2}\right)$$

$$= \frac{R_3(R_1+R_2) - R_2(R_3+R_4)}{R_1+R_2+R_3+R_4}$$

$$R_1 = \frac{-R_2R_4 - X(R_2+R_3+R_4)}{X-R_3}$$

$$R_2 = \frac{R_1R_3 - X(R_1+R_3+R_4)}{X+R_4}$$

$$R_3 = \frac{-R_2R_4 - X(R_1+R_2+R_4)}{X-R_1}$$

$$R_4 = \frac{R_1R_3 - X(R_1+R_2+R_3)}{X+R_2}$$

FIGURE 3.5-1. Circuits Used with Bridge Measurement Instructions

3.6 Measurements Requiring AC Excitation

Some resistive sensors require AC excitation. These include electrolytic tilt sensors, soil moisture blocks, water conductivity sensors and wetness sensing grids. The use of DC excitation with these sensors can result in polarization, which will cause an erroneous measurement, and may shift the calibration of the sensor and/or lead to its rapid decay.

Other sensors like LVDTs (without built in electronics) require an AC excitation because they rely on inductive coupling to provide a signal. DC excitation would provide no output.

Any of the bridge measurements can reverse excitation polarity to provide AC excitation and avoid ion polarization. The frequency of the excitation can be determined by the delay and integration time used with the measurement. The highest frequency possible is 5 kHz, the excitation is switched on and then reversed 100 µs later when the first measurement is held and then is switched off after another 100 µs when the second measurement is held (i.e., reverse the excitation, 100 µs delay, no integration).

Influence of Ground Loop on Measurements

When measuring soil moisture blocks or water conductivity the potential exists for a ground loop which can adversely affect the measurement. This ground loop arises because the soil and water provide an alternate path for the excitation to return to CR5000 ground, and can be represented by the model diagrammed in Figure 3.6-1.

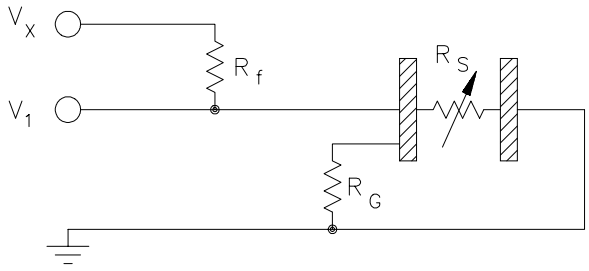


FIGURE 3.6-1. Model of Resistive Sensor with Ground Loop

In Figure 3.6-1, V_x is the excitation voltage, R_f is a fixed resistor, R_s is the sensor resistance, and R_G is the resistance between the excited electrode and CR5000 earth ground. With R_G in the network, the measured signal is:

$$V_1 = V_x \frac{R_s}{\left(R_s + R_f\right) + R_s R_f / R_G} \qquad [3.6\text{-}1]$$

$R_s R_f / R_G$ is the source of error due to the ground loop. When R_G is large the equation reduces to the ideal. The geometry of the electrodes has a great effect on the magnitude of this error. The Delmhorst gypsum block used in the 227 probe has two concentric cylindrical electrodes. The center electrode is used

for excitation; because it is encircled by the ground electrode, the path for a ground loop through the soil is greatly reduced. Moisture blocks which consist of two parallel plate electrodes are particularly susceptible to ground loop problems. Similar considerations apply to the geometry of the electrodes in water conductivity sensors.

The ground electrode of the conductivity or soil moisture probe and the CR5000 earth ground form a galvanic cell, with the water/soil solution acting as the electrolyte. If current was allowed to flow, the resulting oxidation or reduction would soon damage the electrode, just as if DC excitation was used to make the measurement. Campbell Scientific probes are built with series capacitors in the leads to block this DC current. In addition to preventing sensor deterioration, the capacitors block any DC component from affecting the measurement.

3.7 Pulse Count Measurements

Many pulse output type sensors (e.g., anemometers and flow-meters) are calibrated in terms of frequency (counts/second). For these measurements the accuracy is related directly to the accuracy of the time interval over which the pulses are accumulated. Frequency dependent measurements should have the PulseCount instruction programmed to return frequency. If the number of counts is primary interest, PulseCount should be programmed to return counts (i.e., the number of times a door opens, the number of tips of a tipping bucket rain gage).

The interval of the scan loop that PulseCount is in is not the sole determining factor in the calculation of frequency. While normally the counters will be read on the scan interval, if execution is delayed, for example by lengthy output processing, the pulse counters are not read until the scan is synchronized with real time and restarted. The CR5000 actually measures the elapsed time since the last time the counters were read when determining frequency so in the case of an overrun, the correct frequency will still be output.

The resolution of the pulse counters is one count. The resolution of the calculated frequency depends on the scan interval: frequency resolution = 1/scan interval (e.g., a pulse count in a 1 second scan has a frequency resolution of 1 Hz, a 0.5 second scan gives a resolution of 2 Hz, and a 1 ms scan gives a resolution of 1000 Hz). The resultant measurement will bounce around by the resolution. For example, if you are scanning a 2.5 Hz input once a second, in some intervals there will be 2 counts and in some 3 as shown in figure 3.7-1. If the pulse measurement is averaged, the correct value will be the result.

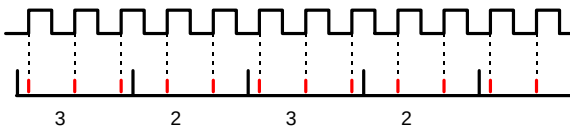


FIGURE 3.7-1. Varying counts within Pulse interval.

The resolution gets much worse with the shorter intervals used with higher speed measurements. As an example, assume that engine RPM is being measured from a signal that outputs 30 pulses per revolution. At 2000 RPM, the signal has a frequency of 100 Hz (2000 RPMx(1 min/60 s)x30=1000). The multiplier to convert from frequency to RPM is 2 RPM/Hz (1 RPM/(30 pulses/60s) = 2). At a 1 second scan interval, the resolution is 2 RPM. However, if the scan interval were 1 ms, the resolution would be 2000 RPM. At the 1 ms scan, if every thing was perfect, each interval there would be 1 count. However, a slight variation in the frequency might cause 2 counts within one interval and none in the next, causing the result to vary between 0 and 4000 RPM!

3.8 Self Calibration

The CR5000 performs a self-calibration of the analog voltage measurements and excitation voltages. The range gains and offsets and the excitation voltage output will vary with temperature. The self calibration allows the CR5000 to maintain its specifications over the temperature range.

Rather than make all of the measurements required to calibrate all range/integration type combinations possible in the CR5000, the calibration only measures the range/integration type combinations that occur in the running CR5000 program. The calibration may occur in three different modes.

1. Compile time calibration. This occurs prior to running the program and calibrates all integration/range combinations needed. For the 250 usc and zero integrations multiple measurements are made and averaged to come up with gain values to use in the measurements. Ten measurements are made on the zero integration ranges and five measurements for the 250 usec integrations. When this calibration is performed the values in the calibration table are completely replaced (i.e., no filtering is used).

2. System background calibration. This automatically takes place in the background while the user program is running. Multiple measurements are not averaged, but a filter is applied to the new gain/offset values obtained. The filter is used so that the calibration values change slowly. The filter combines the newly measured value multiplied by 0.1 with the previous calibration value by 0.9 to arrive at the new calibration value. A piece of the background calibration is added to each fast scan in the user program. The background calibration measurements will be repeated every 4 seconds or the time it takes to complete them, whichever is longer. If there is not enough time to do the background calibration, the CR5000 will display: "Warning when Fast Scan X is running background calibration will be disabled." (X is the number of the fast scan where the first scan entered in the program is 1, the next scan is 2, etc.)

3. Calibration under program control. When the calibrate instruction is included in a program, the calibration is identical to the compile time calibration. The calibration table values are replaced with those calculated. The fast integrations have averaging as in the compile calibrate. When a calibrate instruction is found in any scan the

background calibration will be disabled (even if the scan is not executed). The calibrate instruction is described in Section 7.

The self calibration does not take place if there is not enough time to run it or if the calibrate instruction is in the CR5000 program and never executed. Without the self calibration the drift in accuracy with temperature is about a factor of 10 worse. For example, over the extended temperature range (-40 to 85°C) the accuracy specification is approximately 0.1% of reading. If the self calibration is disabled, the accuracy over the range is approximately 1% of reading. Temperature is the main factor causing a calibration shift and the need for the self calibration. If the temperature of the CR5000 remains the same there will be little calibration drift with the self calibration disabled.

The time constant for the background calibration (at the 4 second rate) is approximately 36 seconds. This allows the CR5000 to remain calibrated during fairly rapid temperature changes. In cases of extreme temperature change, such as bringing a vehicle from equilibrium in a chamber at -30°C out into a hot Arizona day, it may be worthwhile to override the background calibration by running the calibration instruction in the scan with the measurements.

Another case where using the calibration instruction makes sense is where there is not time for the background calibration in the normal scan but the program can periodically stop making measurements and run the calibration instruction in a separate scan.

Section 4. CRBasic - Native Language Programming

The CR5000 is programmed in a language that has some similarities to a structured basic. There are special instructions for making measurements and for creating tables of output data. The results of all measurements are assigned variables (given names). Mathematical operations are written out much as they would be algebraically. This section describes a program, its syntax, structure, and sequence.

4.1 Format Introduction

4.1.1 Mathematical Operations

Mathematical operations are written out much as they would be algebraically. For example, to convert a temperature in Celsius to Fahrenheit one might write:

```
TempF = TempC * 1.8 + 32
```

With the CR5000 there may be 2 or 20 temperature (or other) measurements. Rather than have 20 different names, a *variable array* with one name and 20 elements may be used. A thermocouple temperature might be called TCTemp. With an array of 20 elements the names of the individual temperatures are TCTemp(1), TCTemp(2), TCTemp(3), ... TCTemp(20). The array notation allows compact code to perform operations on all the variables. For example, to convert ten temperatures in a variable array from C to F:

```
For I=1 to 10
        TCTemp(I)=TCTemp(I)*1.8+32
Next I
```

4.1.2 Measurement and Output Processing Instructions

Measurement instructions are procedures that set up the measurement hardware to make a measurement and place the results in a variable or a variable array. Output processing instructions are procedures that store the results of measurements or calculated values. Output processing includes averaging, saving maximum or minimum, standard deviation, FFT, etc.

The instructions for making measurements and outputting data are not found in a standard basic language. The instructions Campbell Scientific has created for these operations are in the form of procedures. The procedure has a keyword name and a series of parameters that contain the information needed to complete the procedure. For example, the instruction for measuring the temperature of the CR5000 input panel is:

PanelTemp *(Dest, Integ)*

PanelTemp is the keyword name of the instruction. The two parameters associated with PanelTemp are: *Dest*ination, the name of the variable in which to put the temperature; and *Integ*ration, the length of time to integrate the measurement. To place the panel temperature in the variable RefTemp (using a 250 microsecond measurement integration time) the code is:

```
PanelTemp(RefTemp, 250)
```

The use of these instructions should become clearer as we go through an introductory example.

4.1.3 Inserting Comments Into Program

Comments can be inserted into a program by preceding the comment with a single quote ('). Comments can be entered either as independent lines or following CR5000 code. When the CR5000 compiler sees the ' it ignores the rest of the line.

```
' The declaration of variables starts here.
Public Start(6)      'Declare the start time array
```

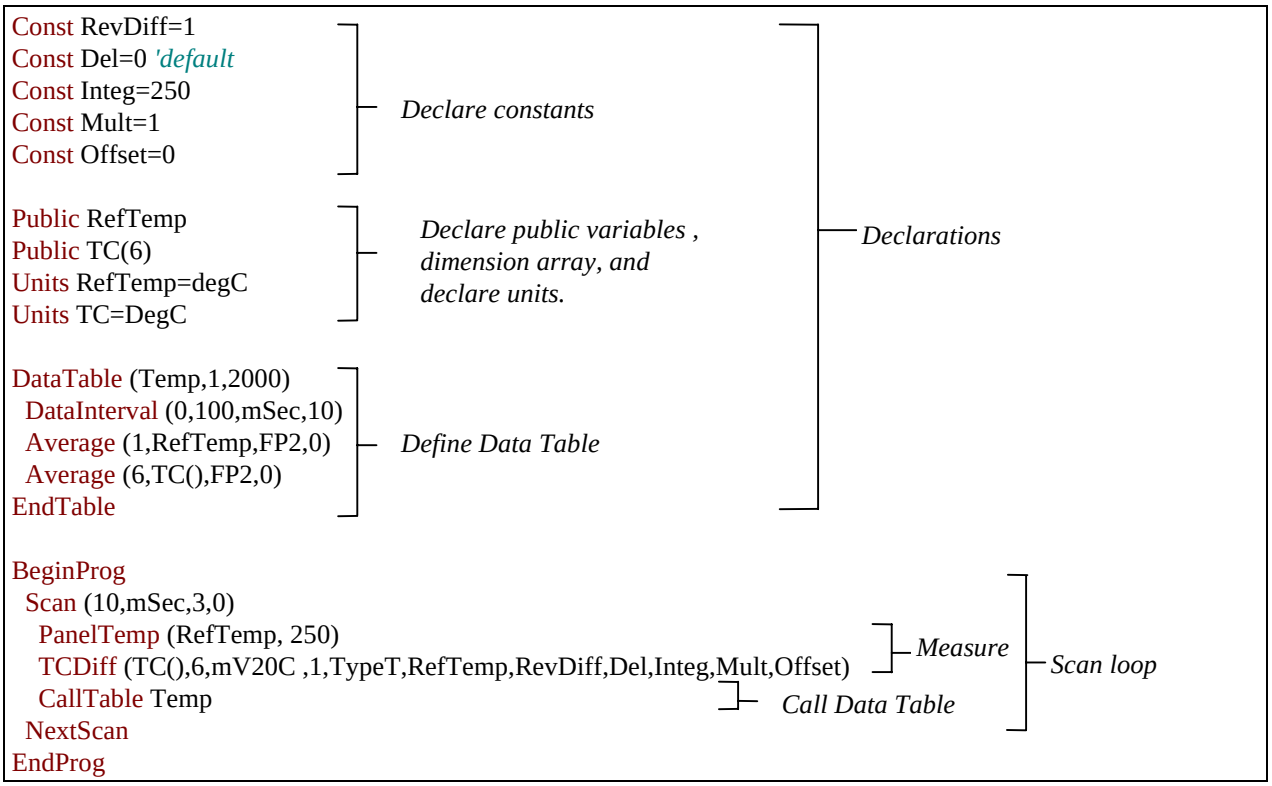
4.2 Programming Sequence

The following table describes the structure of a typical CR5000 program:

Declarations	*Make a list of what to measure and calculate.*
Declare constants	*Within this list, include the fixed constants used,*
Declare Public variables	*indicate the values that the user is able to view while the program is running,*
Dimension variables	*the number of each measurement that will be made,*
Define Aliases	*and specific names for any of the measurements.*
Define data tables.	*Describe, in detail, tables of data that will be saved from the experiment.*
Process/store trigger	*Set when the data should be stored. Are they stored when some condition is met? Are data stored on a fixed interval? Are they stored on a fixed interval only while some condition is met?*
Table size	*Set the size of the table in CR5000 RAM*
Other on-line storage devices	*Should the data also be sent to the PC card?*
Processing of Data	*What data are to be output (current value, average, maximum, minimum, etc.)*

Define Subroutines	*If there is a process or series of calculations that need to be repeated several times in the program, it can be packaged in a subroutine and called when needed rather than repeating all the code each time.*
Program	*The program section defines the action of datalogging*
Set scan interval	*The scan sets the interval for a series of measurements*
Measurements	*Enter the measurements to make*
Processing	*Enter any additional processing with the measurements*
Call Data Table(s)	*The Data Table must be called to process output data*
Initiate controls	*Check measurements and Initiate controls if necessary*
NextScan	*Loop back (and wait if necessary) for the next scan*
End Program	

4.3 Example Program



4.3.1 Data Tables

Data storage follows a fixed structure in the CR5000 in order to optimize the time and space required. Data are stored in tables such as:

TOA4 TIMESTAMP TS	StnName RECORD RN	Temp RefTemp_Avg degC Avg	TC_Avg(1) degC Avg	TC_Avg(2) degC Avg	TC_Avg(3) degC Avg	TC_Avg(4) degC Avg	TC_Avg(5) degC Avg	TC_Avg(6) degC Avg
1995-02-16 15:15:04.61	278822	31.08	24.23	25.12	26.8	24.14	24.47	23.76
1995-02-16 15:15:04.62	278823	31.07	24.23	25.13	26.82	24.15	24.45	23.8
1995-02-16 15:15:04.63	278824	31.07	24.2	25.09	26.8	24.11	24.45	23.75
1995-02-16 15:15:04.64	278825	31.07	24.21	25.1	26.77	24.13	24.39	23.76

The user's program determines the values that are output and their sequence. The CR5000 automatically assigns names to each field in the data table. In the above table, TIMESTAMP, RECORD, RefTemp_Avg, and TC_Avg(1) are fieldnames. The fieldnames are a combination of the variable name (or alias if one exists) and a three letter mnemonic for the processing instruction that outputs the data. Alternatively, the FieldNames instruction can be used to override the default names.

The data table header also has a row that lists units for the output values. The units must be declared for the CR5000 to fill this row out (e.g., Units RefTemp

= degC). The units are strictly for the user's documentation; the CR5000 makes no checks on their accuracy.

The above table is the result of the data table description in the example program:

```
DataTable (Temp,1,2000)
        DataInterval(0,10,msec,10)
        Average(1,RefTemp,fp2,0)
        Average(6,TC(1),fp2,0)
EndTable
```

All data table descriptions begin with **DataTable** and end with **EndTable**. Within the description are instructions that tell what to output and that can modify the conditions under which output occurs.

DataTable(*Name, Trigger, Size*)
DataTable (Temp,1,2000)

The DataTable instruction has three parameters: a user specified name for the table, a trigger condition, and the size to make the table in CR5000 RAM. The trigger condition may be a variable, expression, or constant. The trigger is true if it is not equal to 0. Data are output if the trigger is true and there are no other conditions to be met. No output occurs if the trigger is false (=0). The example creates a table name Temp, outputs any time other conditions are met, and retains 2000 records in RAM.

DataInterval(*TintoInt, Interval, Units, Lapses*)
DataInterval(0,10,msec,10)

DataInterval is an instruction that modifies the conditions under which data are stored. The four parameters are the time into the interval, the interval on which data are stored, the units for time, and the number of lapses or gaps in the interval to keep track of. The example outputs at 0 time into (on) the interval relative to real time, the interval is 10 milliseconds, and the table will keep track of 10 lapses. The DataInterval instruction reduces the memory required for the data table because the time of each record can be calculated from the interval and the time of the most recent record stored. Other output condition modifiers are: Worst Case and FillandStop.

The output processing instructions included in a data table declaration determine the values output in the table. The table must be called by the program in order for the output processing to take place. That is, each time a new measurement is made, the data table is called. When the table is called, the output processing instructions within the table process the current inputs. If the trigger conditions for the are true, the processed values are output to the data table. In the example, several averages are output.

Average(*Reps, Source, DataType, DisableVar*)
Average(1,RefTemp,fp2,0)
Average(6,TC(1),fp2,0)

Average is an output processing instruction that will output the average of a variable over the output interval. The parameters are repetitions - the number of elements in an array to calculate averages for, the Source variable or array to average, the data format to store the result in (Table 4.3-1), and a disable variable that allows excluding readings from the average if conditions are not met. A reading will not be included in the average if the disable variable is not equal to 0; the example has 0 entered for the disable variable so all readings are included in the average.

TABLE 4.3-1 Formats for Output Data

Code	Data Format	Size	Range	Resolution
FP2	Campbell Scientific floating point	2 bytes	±7999	13 bits (about 4 digits)
IEEE4	IEEE four byte floating point	4 bytes	1.8 E -38 to 1.7 E 38	24 bits (about 7 digits)
LONG	4 byte Signed Integer	4 bytes	-2,147,483,648 to +2,147,483,647	1 bit (1)

4.3.2 The Scan -- Measurement Timing and Processing

Once you know what you want, the measurements and calculations have been listed and the output tables defined, the program itself may be relatively short. The executable program begins with BeginProg and ends with EndProg. The measurements, processing, and calls to output tables bracketed by the Scan and NextScan instructions determine the sequence and timing of the datalogging.

```
BeginProg
Scan(1,MSEC,3,0)
        ModuleTemp(RefTemp, 250)
        TCDiff(TC(),6,mV50,4,1,TypeT,RefTemp,RevDiff,Del,Integ,Mult,Offset)
        CallTable Temp
NextScan
EndProg
```

The Scan instruction determines how frequently the measurements within the scan are made:

Scan*(Interval, Units, BufferSize, Count)*

Scan(1,MSEC,3,0)

The Scan instruction has four parameters. The *Interval* is the interval between scans. *Units* are the time units for the interval. The maximum scan interval is one minute. The *BufferSize* is the size (in the number of scans) of a buffer in RAM that holds the raw results of measurements. Using a buffer allows the processing in the scan to at times lag behind the measurements without affecting the measurement timing (see the scan instruction in Section 9 for more details). *Count* is the number of scans to make before proceeding to the instruction following NextScan. A count of 0 means to continue looping

forever (or until ExitScan). In the example the scan is 1 millisecond, three scans are buffered, and the measurements and output continue indefinitely.

4.4 Numerical Entries

In addition to entering regular base 10 numbers there are 3 additional ways to represent numbers in a program: scientific notation, binary, and hexadecimal (Table 4.4-1).

TABLE 4.4-1 Formats for Entering Numbers in CRBasic

Format	Example	Value
Standard	6.832	6.832
Scientific notation	5.67E-8	$5.67X10^{-8}$
Binary:	&B1101	13
Hexadecimal	&HFF	255

The binary format makes it easy to visualize operations where the ones and zeros translate into specific commands. For example, a block of ports can be set with a number, the binary form of which represents the status of the ports (1= high, 0=low). To set ports 1, 3, 4, and 6 high and 2, 5, 7, and 8 low; the number is &B00101101. The least significant bit is on the right and represents port 1. This is much easier to visualize than entering 72, the decimal equivalent.

4.5 Logical Expression Evaluation

4.5.1 What is True?

Several different words get used to describe a condition or the result of a test. The expression, X>5, is either **true** or **false**. However, when describing the state of a port or flag, **on** or **off** or **high** or **low** sounds better. In CRBasic there are a number of conditional tests or instruction parameters the result of which may be described with one of the words in Table 4.5-1. The CR5000 evaluates the test or parameter as a number; 0 is false, not equal to 0 is true.

TABLE 4.5-1. Synonyms for True and False

Predefined Constant	True (-1)	False (0)
Synonym	High	Low
Synonym	On	Off
Synonym	Yes	No
Synonym	Trigger	Do Not Trigger
Number	≠0	0
Digital port	5 Volts	0 Volts

4.5.2 Expression Evaluation

Conditional tests require the CR5000 to evaluate an expression and take one path if the expression is true and another if the expression is false. For example:
If X>=5 then Y=0
will set the variable Y to 0 if X is greater than or equal to 5.

The CR5000 will also evaluate multiple expressions linked with **and** or **or**. For example:
If X>=5 and Z=2 then Y=0
will only set Y=0 if both X>=5 and Z=2 are true.
If X>=5 or Z=2 then Y=0
will set Y=0 if either X>=5 or Z=2 is true (see And and Or in Section 9). A condition can include multiple **and** and **or** links.

4.5.3 Numeric Results of Expression Evaluation

The CR5000 expression evaluator evaluates an expression and returns a number. A conditional statement uses the number to decide which way to branch. The conditional statement is false if the number is 0 and true if the number is not 0. For example:
If 6 then Y=0,
is always true, Y will be set to 0 any time the conditional statement is executed.
If 0 then Y=0
is always false, Y will never be set to 0 by this conditional statement.

The CR5000 expression evaluator evaluates the expression, X>=5, and returns -1, if the expression is true, and 0, if the expression is false.
W=(X>Y)
will set W equal to -1 if X>Y or will set W equal to 0 if X<=Y.

The CR5000 uses -1 rather than some other non-zero number because the **and** and **or** operators are the same for logical statements and binary bitwise comparisons (see **and** and **or** in Section 8). The number -1 is expressed in binary with all bits equal to 1, the number 0 has all bits equal to 0. When -1 is anded with any other number the result is the other number, ensuring that if the other number is non-zero (true), the result will be non-zero

4.6 Flags

Any variable can be used as a flag as far as logical tests in CRBasic are concerned. If the value of the variable is non-zero the flag is high. If the value of the variable is 0 the flag is low (Section 4.5). PC9000 looks for the variable array with the name **Flag** when the option to display flag status is used in one of the real time screens.

4.7 Parameter Types

Instructions parameters allow different types of inputs these types are listed below and specifically identified in the description of the parameter in the following sections or in PC9000 CRBasic help.

Constant
Variable
Variable or Array
Constant, Variable, or Expression
Constant, Variable, Array, or Expression
Name
Name or list of Names
Variable, or Expression
Variable, Array, or Expression

Table 4.7-1 list the maximum length and allowed characters for the names for Variables, Arrays, Constants, etc.

TABLE 4.7-1. Rules for Names

Name for	Maximum Length (number of characters)	Allowed characters
Variable or Array	16	Letters A-Z, upper or lower. case, underscore "_", and numbers 0-9. The name must start with a letter. CRBasic is not case sensitive
Constant	16	
Alias	16	
Data Table Name	8	
Field name	16	

4.7.1 Expressions in Parameters

Many parameters allow the entry of expressions. If an expression is a comparison, it will return -1 if the comparison is true and 0 if it is false (Section 4.5.3). An example of the use of this is in the DataTable instruction where the trigger condition can be entered as an expression. Suppose the variable TC(1) is a thermocouple temperature:

DataTable(Name, TrigVar, Size)

DataTable(Temp, TC(1)>100, 5000)

Entering the trigger as the expression, TC(1)>100, will cause the trigger to be true and data to be stored whenever the temperature TC(1) is greater than 100.

4.7.2 Arrays of Multipliers Offsets for Sensor Calibration

If variable arrays are used as the multiplier and offset parameters in measurements that use repetitions, the instruction will automatically step through the multiplier and offset arrays as it steps through the channels. This allows a single measurement instruction to measure a series of individually

calibrated sensors, applying the correct calibration to each sensor. If the multiplier and offset are not arrays, the same multiplier and offset are used for each repetition.

VoltSE(Dest,Reps,Range,ASlot,SEChan,Delay,
Integ,Mult,Offset)

```
'Calibration factors:
Mult(1)=0.123 : Offset(1)= 0.23
Mult(2)=0.115 : Offset(2)= 0.234
Mult(3)=0.114 : Offset(3)= 0.224
VoltSE(Pressure(),3,mV1000,6,1,1,100,Mult(),Offset()
```

4.8 Program Access to Data Tables

Data stored in a table can be accessed from within the program. The format used is:

Tablename.Fieldname(fieldname index,records back)

Where *Tablename* is the name of the table in which the desired value is stored. *Fieldname* is the name of the field in the table. The fieldname is always an array even if it consists of only one variable; the *fieldname index* must always be specified. *Records back* is the number of records back in the data table from the current time (1 is the most recent record stored, 2 is the record stored prior to the most recent). For example, the expression:

Tdiff=Temp.TC_Avg(1,1)–Temp.TC_Avg(1,101)

could be used in the example program (Section 4.3) to calculate the change in the 10 ms average temperature of the first thermocouple between the most recent average and the one that occurred a second (100 x 10 ms) earlier.

In addition to accessing the data actually output in a table, there are some pseudo fields related to the data table that can be retrieved:

Tablename.record(1,n) = the record number of the record output n records ago.

Tablename.output(1,1) = 1 if data were output to the table the last time the table was called, = 0 if data were not output.

Tablename.timestamp(m,n) = element m of the timestamp output n records ago where:

timestamp(1,n) = microseconds since 1990
timestamp(2,n) = microseconds into the current year
timestamp(3,n) = microseconds into the current month
timestamp(4,n) = microseconds into the current day
timestamp(5,n) = microseconds into the current hour
timestamp(6,n) = microseconds into the current minute
timestamp(7,n) = microseconds into the current second

Tablename.eventend(1,1) is only valid for a data table using the DataEvent instruction, *Tablename*.eventend(1,1) = 1 if the last record of an event occurred the last time the table was called, = 0 if the data table did not store a record or if it is in the middle of an event.

NOTE The values of Tablename.output(1,1) and Tablename.eventend (1,1) are only updated when the tables are called.

The WorstCase example in Section 6.2 illustrates the use of this syntax.

Section 5. Program Declarations

Alias

Used to assign a second name to a variable.

Syntax
Alias *VariableA* = *VariableB*

Remarks
Alias allows assigning a second name to a variable. Within the datalogger program, either name can be used. Only the alias is available for Public variables. The alias is also used as the root name for data table fieldnames.

With aliases the program can have the efficiency of arrays for measurement and processing yet still have individually named measurements.

Alias Declaration Example

The example shows how to use the Alias declaration.

```
Dim TCTemp(4)
Alias TCTemp(1) = CoolantT
Alias TCTemp(2) = ManifoldT
Alias TCTemp(3) = ExhaustT
Alias TCTemp(4) = CatConvT
```

Const

Declares symbolic constants for use in place of numeric entries.

Syntax
Const *constantname = expression*

Remarks
The **Const** statement has these parts:

Part	Description
constantname	Name of the constant.
expression	Expression assigned to the constant. It can consist of literals (such as 1.0), other constants, or any of the arithmetic or logical operators.

Tip Constants can make your programs self-documenting and easier to modify. Unlike variables, constants can't be changed while your program is running.

Caution Constants must be defined before referring to them.

Tip Use all uppercase letters for constant names to make them easy to recognize in your program listings.

Const Declaration Example

The example uses Const to define the symbolic constant PI.

```
Const PI = 3.141592654          'Define constant.
Dim Area, Circum, Radius        'Declare variables.
Radius = Volt( 1 )              'Get measurement.
Circum = 2 * PI * Radius        'Calculate circumference.
Area = PI * ( Radius ^ 2 )      'Calculate area.
```

Dim

Declares variables and allocates storage space. In CRBasic, **ALL** variables **MUST** be declared.

Syntax
Dim *varname*[([*subscripts*]) [, *varname*[([*subscripts*])]]

Remarks
The **Dim** statement has these parts:

Part	Description
varname	Name of a variable.
subscripts	Dimensions of an array variable. You can declare multiple dimensions.

The argument subscripts has the following syntax:
size [size, size]

In CRBasic the Option Base is always 1. This means the lowest number in a dimension is 1 and not 0.

```
Dim A( 8, 3 )
```

The maximum number of array dimensions allowed in a **Dim** statement is 3. If a program uses a subscript that is greater than the dimensioned value, a subscript out of bounds error is recorded.

When variables are initialized, they are initialized to 0

Tip Put **Dim** statements at the beginning of the program.

Public

Dimensions a variable as public and available in the Public table of the CR5000.

Syntax
Public(list of [dimensioned] variables that make up the Public Table)

Remarks
More than one Public statement can be made.

Public Declaration Example

The example shows the use of the Public declaration.

```
Dim x( 3 ), y, z( 2, 3, 4 )
Public x, y, z
Public Dim x( 3 ), y, z( 2, 3, 4 )        'Dim is optional
Public x( 3 ),y, z( 2, 3, 4 )
Public w
```

Station Name

Sets the station name.

Syntax
StationName StaName

Remarks
StationName is used to set the datalogger station name with the program. The station name is displayed by PC9000 and stored in the data table headers (Section 2.4).

Units

Used to assign a unit name to a field associated with a variable.

Syntax
Units *Variable* = UnitName

Remarks
Units allows assigning a unit name to a field. Units are displayed on demand in the real-time windows of PC9000. The unit name also appears in the header of the output files and in the Data Table Info file of PC9000. The unit name is a text field that allows the user to label data. When the user modifies the units, the text entered is not checked by PC9000 or the CR5000.

Example
Dim TCTemp(1)

Units TCTemp(1) = Deg_C

Sub, Exit Sub, End Sub

Declares the name, variables, and code that form a Subroutine.

Syntax
Sub *SubName* [(*VariableList*)]
[*statementblock*]
[**Exit Sub**]
[*statementblock*]
End Sub

The Sub statement has these parts:

Part	Description
Sub	Marks the beginning of a Subroutine.
SubName	Name of the Subroutine. Because Subroutine names are recognized by all procedures in all modules, *subname* cannot be the same as any other globally recognized name in the program.
VariableList	List of variables that are passed to the Subroutine when it is called. The variable names used in this list should not be the same names as variables, aliases, or constants declared elsewhere. The variable names in this list can only be used within the Subroutine. Multiple variables are separated by commas. When the Subroutine is called, the call statement must list the program variables or values to pass into the subroutine variable. The number and sequence of the program variables/values in the call statement must match the number and sequence of the variable list in the sub declaration. Changing the value of one of the variables in this list inside the Subroutine changes the value of the variable passed into it in the calling procedure. The call may pass constants or expressions that evaluate to constants (i.e., do not contain a variable) into some of the variables. If a constant is passed, the "variable" it is passed to becomes a constant and cannot be changed by the subroutine. If constants will be passed, the subroutine should be written to not try to change the value of the "variables" they will be passed into.
statementblock	Any group of statements that are executed within the body of the Subroutine.
Exit Sub	Causes an immediate exit from a Subroutine. Program execution continues with the statement following the statement that called the Subroutine. Any number of **Exit Sub** statements can appear anywhere in a Subroutine.
End Sub	Marks the end of a Subroutine.

A Subroutine is a procedure that can take variables, perform a series of statements, and change the value of the variables. However, a Subroutine can't be used in an expression. You can call a Subroutine using the name followed by the variable list. See the Call statement for specific information on how to call Subroutines.

The list of Subroutine variables to pass is optional. Subroutines can operate on the global program variables declared by the Public or Dim statements. The advantage of passing variables is that the subroutine can be used to operate on whatever program variable is passed (see example).

Caution Subroutines can be recursive; that is, they can call themselves to perform a given task. However, recursion can lead to strange results.

Subroutine Example

```
'CR5000
'Declare Variables used in Program:
Public RefT, TC(4),PRTresist,PRTtemp,I

'Data output in deg C:
DataTable (TempsC,1,-1)
   DataInterval (0,5,Min,10)
   Average (1,RefT,FP2,0)
   Average (4,TC(),FP2,0)
   Average (1,PRTtemp,FP2,0)
EndTable

'Same Data output in F:
DataTable (TempsF,1,-1)
   DataInterval (0,5,Min,10)
   Average (1,RefT,FP2,0)
   Average (4,TC(),FP2,0)
   Average (1,PRTtemp,FP2,0)
EndTable

'Subroutine to convert temperature in degrees C to degrees F
Sub ConvertCtoF (Tmp)
   Tmp = Tmp*1.8 +32
EndSub

BeginProg
   Scan (1,Sec,3,0)
      'Measure Temperatures (panel, 4 thermocouples, and 100 ohm PRT) in deg C
      PanelTemp (RefT,250)
      TCDiff (TC(),4,mV20C ,1,TypeT,RefT,True ,0,250,1.0,0)
      Resistance (PRTresist,1,mV50,7,Ix1,1,500,True ,True ,0,250,0.01,0)
      PRT (PRTtemp,1,PRTresist,1.0,0)
      'Call Output Table for C
      CallTable TempsC
      'Convert Temperatures to F using Subroutine:
      Call ConvertCtoF(RefT)        'Subroutine call using Call statement
      For I = 1 to 4
         ConvertCtoF(TC(I))         'Subroutine call without Call statement
      Next I
      ConvertCtoF(PRTtemp)          'Subroutine call without Call statement
      'Call Output Table for F:
      CallTable TempsF
   NextScan
EndProg
```

Section 6. Data Table Declarations and Output Processing Instructions

6.1 Data Table Declaration

DataTable (Name, TrigVar, Size)

output trigger modifier

export data destinations

output processing instructions

EndTable

DataTable is used to declare/define a data table. The name of the table, output trigger and size of the table in RAM are set with DataTable. The Table declaration must be at the beginning of the code prior to BeginProg. The table declaration starts with DataTable and ends with EndTable. Within the declaration are output trigger modifiers (optional, e.g., DataInterval, DataEvent or WorstCase), the on-line storage devices to send the data to (optional, e.g., CardOut, DSP4), and the output processing instructions describing the data set in the table.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>Name
Name</td><td>The name for the data table. The table name is limited to eight characters.</td></tr>
<tr><td>TrigVar

Constant
Variable, or
Expression</td><td>The name of the variable to test for the trigger. Trigger modifiers add additional conditions.
Value | Result
0 | Do not trigger
≠ 0 | Trigger</td></tr>
<tr><td>Size
Constant</td><td>The size to make the data table. The number of data sets (records) to allocate memory for in static RAM. Each time a variable or interval trigger occurs, a line (or row) of data is output with the number of values determined by the output Instructions within the table. This data is called a record. The total number of records stored equals the size..
Note Enter a negative number and all remaining memory (after creating fixed size data tables) will be allocated to the table or partitioned between all tables with a negative value for size. The partitioning algorithm attempts to have the tables fill at the same time.</td></tr>
</table>

DataTable Example - *see native language Section 4.*

EndTable

Used to mark the end of a data table.

See DataTable

6.2 Trigger Modifiers

DataInterval (TintoInt, Interval, Units, Lapses)

Used to set the time interval for an output table. DataInterval is inserted into a data table declaration following the DataTable instruction to establish a fixed interval table. The fixed interval table requires less memory than a conditional table because time is not stored with each record. The time of each record is calculated by knowing the time of the most recent output and the interval of the data. DataInterval does not override the Trigger in the DataTable instruction. If the trigger is not set always true by entering a constant, it is a condition that must be met in addition to the time interval before data will be stored.

The **Interval** determines how frequently data are stored to the table. The interval is synchronized with the real time clock. Time is kept internally as the elapsed time since the start of 1990 (01-01-1990 00:00:00). When the interval divides evenly into this elapsed time it is time to output (elapsed time MOD interval = 0). Entering 0 for the Interval sets it equal to the scan Interval.

TintoInt allows the user to set the time into the Interval, or offset relative to real time, at which the output occurs([elapsed time + TintoInt] MOD interval = 0). For example, 360 (TintoInt) minutes into a 720 (Interval) minute (Units) interval specifies that output should occur at 6:00 (6 AM, 360 minutes from midnight) and 18:00 (6 PM, 360 minutes from noon) where the 720 minute (12 hour) interval is set relative to midnight 00:00. Enter 0 to keep output on the even interval.

Interval driven data allows a more efficient use of memory because it is not necessary to store time with each record. The CR5000 still stores time but on a fixed spacing, only about once per 1 K of memory used for the table. As each new record is stored, time is checked to ensure that the interval is correct. The datalogger keeps track of lapses or discontinuities in the data. If a lapse has occurred, the CR5000 inserts a time stamp into the data. When the data are retrieved a time stamp can be calculated and stored with each record.

This lapse time stamp takes up some memory that would otherwise be used for data. While the CR5000 allocates some extra memory for the table, if there are a lot of lapses, it is not possible to store as many records as requested in the DataTable declaration. The **Lapses** parameter allows the programmer to allocate additional space for the number of lapses entered. This is used in particular when the program is written in a way that will create lapses. For example, if the data output is controlled by a trigger (e.g., a user flag) in the DataTable instruction in addition to the DataInterval, lapses would occur each time the trigger was false for a period of time longer than the interval.

To take advantage of the more efficient memory use, always enter 1 or greater for the lapses parameter even if no lapses are expected. Entering 0 causes every record to be time stamped.

Entering a negative number tells the CR5000 not to keep track of lapses. Only the periodic time stamps (approximately once per K of data) are inserted.

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td>TintoInt
Constant</td><td>The time into the interval (offset to the interval) at which the table is to be output. The units for time are the same as for the interval.</td></tr>
<tr><td>Interval
Constant</td><td>Enter the time interval on which the data in the table is to be recorded. The interval may be in µs, ms, s, or minutes, whichever is selected with the Units parameter. Enter 0 to make the data interval the same as the scan interval.</td></tr>
<tr><td>Units

Constant</td><td>The units for the time parameters, PowerOff is the only instruction that uses hours or days.
<table>
<tr><th>Alpha Code</th><th>Numeric Code</th><th>Units</th></tr>
<tr><td>USEC</td><td>0</td><td>microseconds</td></tr>
<tr><td>MSEC</td><td>1</td><td>milliseconds</td></tr>
<tr><td>SEC</td><td>2</td><td>seconds</td></tr>
<tr><td>MIN</td><td>3</td><td>minutes</td></tr>
</table></td></tr>
<tr><td>Lapses
Constant</td><td>As each new record is stored, time is checked to ensure that the interval is correct. The datalogger keeps track of lapses or discontinuities in the data.</td></tr>
</table>

OpenInterval

When the DataInterval instruction is included in a data table, the CR5000 uses only values from within an interval for time series processing (e.g., average, maximum, minimum, etc.). When data are output every interval, the output processing instructions reset each time output occurs. To ensure that data from previous intervals is not included in a processed output, processing is reset any time an output interval is skipped. (An interval could be skipped because the table was not called or another trigger condition was not met.) The CR5000 resets the processing the next time that the table is called after an output interval is skipped. If this next call to the table is on a scheduled interval, it will not output. Output will resume on the next interval. (If Sample is the *only* output processing instruction in the table, data will be output any time the table is called on the interval because sampling uses only the current value and involves no processing.)

OpenInterval is used to modify an interval driven table so that time series processing in the table will include all values input since the last time the table output data. Data will be output whenever the table is called on the output interval (provided the other trigger conditions are met), regardless of whether or not output occurred on the previous interval.

OpenInterval Example:

In the following example, 5 thermocouples are measured every 500 milliseconds. Every 10 seconds, *while Flag(1) is true,* the averages of the reference and thermocouple temperatures are output. The user can toggle Flag(1) to enable or disable the output. Without the OpenInterval Instruction, the first averages output after Flag(1) is set high would include only the measurements within the previous 10 second interval. This is the default and is what most users desire. With the Open interval in the program (remove the initial single quote (‘) to uncomment the instruction) all the measurements made while the flag was low will be included in the first averages output after the flag is set high.

```
Const RevDiff 1         'Reverse input to cancel offsets
Const Del 0             'Use default delay
Const Integ 0           'Use no integration
Public RefTemp          'Declare the variable used for reference temperature
Public TC(5)            'Declare the variable used for thermocouple measurements
Public Flag(8)
Units RefTemp=degC                '
Units TC=degC

DataTable (AvgTemp,Flag(1),1000)     'Output when Flag(1)=true
        DataInterval(0,10,sec,10)    'Output every 10 seconds(while Flag(1)=true)
        'OpenInterval       'When Not Commented, include data while Flag(1)=false in next average
        Average(1,RefTemp,IEEE4,0)
        Average(5,TC,IEEE4,0)
EndTable

BeginProg
        Scan(500,mSec,0,0)
                PanelTemp (RefTemp,250)
                TCDiff (TC(),5,mV50C,9,TypeT,RefTemp,RevDiff,Del,Integ,1,0)
                CallTable AvgTemp
        NextScan
EndProg
```

DataEvent (RecsBefore, StartTrig, StopTrig, RecsAfter)

Used to set a trigger to start storing records and another trigger to stop storing records within a table. The number of records before the start trigger and the number of records after the stop trigger can also be set. A filemark (Section 8) is automatically stored in the table between each event.

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td>RecsBefore
Constant</td><td>The number of records to store before the Start Trigger.</td></tr>
<tr><td>StartTrig

Variable, or Expression</td><td>The variable or expression test to Trigger copying the pre trigger records into the data table and start storing each new record..
Value | Result
0 | Do not trigger
≠ 0 | Trigger</td></tr>
<tr><td>StopTrig
Variable, Expression or Constant</td><td>The variable, expression or constant to test to stop storing to the data table. The CR5000 does not start checking for the stop trigger until after the Start Trigger occurs. A non-zero (true) constant may be used to store a fixed number of records when the start trigger occurs (total number of records = PreTrigRecs+ 1 record for the trigger +PostTrigRecs.). Zero (false) could be entered if it was desired to continuously store data once the start trigger occurred.
Value | Result
0 | Do not trigger
≠ 0 | Trigger</td></tr>
<tr><td>RecsAfter
Constant</td><td>The number of records to store after the Stop Trigger occurs.</td></tr>
</table>

DataEvent Example:

In this example, 5 type T thermocouples are measured. The trigger for the start of an event is when TCTemp(1) exceeds 30 degrees C. The stop trigger is when TCTemp(1) less than 29 degrees C. The event consists of 20 records prior to the start trigger and continues to store data until 10 records following the stop trigger.

```
Const RevDiff 1          'Reverse input to cancel offsets
Const Del 0              'Use default delay
Const Integ 0            'Use no integration
Public RefTemp           'Declare the variable used for reference temperature
Public TC(5)             'Declare the variable used for thermocouple measurements
Public Flag(8)
Units RefTemp=degC       '
Units TC=degC

DataTable (Event,1,1000)
        DataInterval(0,00,msec,10)              'Set the sample interval equal to the scan
        DataEvent(20,TC(1)>30,TC(1)<29,10)      '20 records before TC(1)>30,
                                                'after TC(1)<29 store 10 more records
        Sample(1,RefTemp,IEEE4)                 'Sample the reference temperature
        Sample(5,TC,IEEE4)                      'Sample the 5 thermocouple temperatures
EndTable

BeginProg
        Scan(500,mSec,0,0)
                PanelTemp (RefTemp,250)
                TCDiff (TC(),5,mV50C,1,TypeT,RefTemp,RevDiff,Del,Integ,1,0)
                CallTable Event
        NextScan
EndProg
```

FillStop

Data Tables are by default ring memory where, once full, the newest data are written over the oldest. Entering **FillStop** into a data table declaration makes the table fill and stop. Once the table is filled, no more data are stored until the table has been reset. The table can be reset from within the program by executing the ResetTable instruction. Tables can also be reset from PC9000 real time windows or the collect data window.

Example:

```
DataTable (Temp,1,2000)
        DataInterval(0,10,msec,10)
        FillStop                        ' the table will stop collecting data after 2000 records.
        Average(1,RefTemp,fp2,0)
        Average(6,TC(1),fp2,0)
EndTable
```

WorstCase (TableName, NumCases, MaxMin, Change, RankVar)

Allows saving the most significant or "worst-case" events in separate data tables.

A data table is created that is sized to hold one event. This table acts as the event buffer. Each event that occurs is stored to this table. This table may use the DataEvent instruction or some other condition to determine when an event is stored. The significance of an event is determined by an algorithm in the program and a numerical ranking of the event is stored in a variable.

WorstCase creates as many clones of the specified table as the number of cases for which to keep data. When WorstCase is executed, it checks the ranking variable; if the value of the variable is a new worst case, the data in the event table replace the data in the cloned table that holds the least significant event currently stored.

An additional data table, *name*WC (e.g., EvntWC) is created that holds the values of the rank variables for each of the worst case tables and the time that that table was stored.

WorstCase must be used with data tables sent to the CPU. It will not work if the event table is sent to the PC card.

While WorstCase acts as Trigger Modifier and a data table declaration (creating the cloned data tables), it is entered within the program to call the worst case tables (see example).

Parameter *& Data Type*	**Enter**
TableName *name*	The name of the data table to clone. The length of this name should be 4 characters or less so the complete names of the worst case tables are retained when collected (see NumCases).
NumCases	The number of "worst" cases to store. This is the number of clones of the data table to create. The cloned tables use the name of the table being cloned (up to the first 6 characters) plus a 2 digit number (e.g., Evnt01, Evnt02, Evnt03, …). The numbers give the tables unique names, they have no relationship to the ranking of the events. PC9000 uses this same name modification when creating a new data file for a table. To avoid confusion and ambiguous names when collecting data with PC9000, keep the base name four characters or less (4character base name + 2 digit case identifier + 2 digit collection identifier = 8 character maximum length).
MaxMin *Constant*	A code specifying whether the maximum or minimum events should be saved. **Value** — **Result** 0 — **Min**, save the events associated with the minimum ranking; i.e., Keep track of the RankVar associated with each event stored. If a new RankVar is less than previous maximum, copy the event over the event with previous maximum) 1 — **Max**, save the events associated with the maximum ranking; i.e., copy if RankVar is greater than previous lowest (over event with previous minimum)
Change *Constant*	The minimum change that must occur in the RankVariable before a new worst case is stored.
RankVar *Variable*	The Variable to rank the events by.

WorstCase Example

This program demonstrates the Worst Case Instruction. Five type T thermocouples are measured. The event is similar to that in the example for the DataEvent instruction; the trigger for the start of a data event is when TC(1) exceeds 30 degrees C. However in this example, the stop trigger is set immediately true. This is done to set a fixed size for the event which can be duplicated in the worst case tables. To use the worst case instruction with events of varying duration, the event table size must be selected to accommodate the maximum duration expected (or needed). The event consists of 20 records prior to the start trigger and continues until 100 records following the start trigger.

The ranking criteria is the number of readings following the trigger that TC(1) stays above 30 degrees C. The greater the number the "worse" the event.

```
Const RevDiff 1          'Reverse input to cancel offsets
Const Del 0              'Use default delay
Const Integ 0            'Use default Integration
Const NumCases 5         'Number of Worst Cases to save
Const Max 1
Public RefTemp           'Declare the variable used for reference temperature
Public TC(5)             'Declare the variable used for thermocouple measurements
Public I, NumAbove30     'Declare index and the ranking variable
Units RefTemp=degC       '
Units TC=degC

DataTable (Evnt,1,125)
    DataInterval(0,00,msec,10)          'Set the sample interval equal to the scan
    DataEvent(20,TC(1)>30,-1,100)       '20 records before TC(1)>30,
                                        '100 records after TC(1)>30
    Sample(1,RefTemp,IEEE4)             'Sample the reference temperature
    Sample(5,TC,IEEE4)                  'Sample the 5 thermocouple temperatures
EndTable

BeginProg
    Scan(500,mSec,3,0)
        ModuleTemp(RefTemp,1,5,30)
        TCDiff(TC(),5,mV50C,5,9,TypeT,RefTemp,RevDiff,Del,Integ,1,0)
        CallTable Evnt
        If Evnt.EventEnd(1,1)                       Check if an Event just Ended
            I=100                                   'Initialize Index
            NumAbove30=0                            'Zero Ranking Variable
            Do                                      'Loop through the Event table
                NumAbove30=NumAbove30+1             'Counting the # of times TC(1)>30
                I=I-1
            Loop While I>0 and Evnt.TC(1,I)>=30     'Quit looping when at end or TC(1)<30
            WorstCase(Evnt,NumCases,Max,0,NumAbove30)       'Check for worst case
        End If
    NextScan
EndProg
```

6.3 Export Data Instructions

CardOut (StopRing, Size)

Used to send output data to the PCMCIA card. This instruction creates a data table on the PCMCIA card. CardOut must be entered within each data table declaration that is to store data on a PCMCIA card.

Parameter *& Data Type*	**Enter**
StopRing *Constant*	A code to specify if the Data Table on the PCMCIA card is fill and stop or ring (newest data overwrites oldest). **Value** / **Result** 0 Ring 1 Fill and Stop
Size *Constant*	The size to make the data table. The number of data sets (records) to allocate memory for in the PCMCIA card. Each time a variable or interval trigger occurs, a line (or row) of data is output with the number of values determined by the output Instructions within the table. This data is called a record. **Note** Enter a negative number and all remaining memory (after creating fixed size data tables) will be allocated to the table or partitioned between all tables with a negative value for size. The partitioning algorithm attempts to have the tables fill at the same time.

DSP4 (FlagVar, Rate)

Send data to the DSP4. If this instruction appears inside a DataTable, the DSP4 can display the fields of this Table, otherwise, the Public Variables are used by the DSP4. The Instruction can only be used once in a program; hence, only the public variables or a single data table can be viewed.

Parameter *& Data Type*	**Enter**
FlagVar *Array*	The variable array to use for the 8 flags that can be displayed and toggled by the DSP4. A value of 0 = low; ≠0 = high. If the array is dimensioned to less than 8, the DSP4 will only work with the flags up to the dimension. The array used for flags in the Real Time displays of PC9000 is Flag ().
Rate *Constant*	How frequently to send new values to the DSP4 in milliseconds.

Example

```
DSP4 (Flag( ), 200)
```

Use Flag( ) to work with the buttons, update the DSP4 display every 200 msec. (5 times a second).

GOESData (Dest, Table, TableOption, BufferControl, DataFormat)

The GOESData instruction is used to transmit data to the SAT HDR GOES satellite data transmitter. The GOESData instruction is not inserted within the Data Table declaration, it is inserted within the program, typically within the scan.

Data transfer to the transmitter can occur via the datalogger's CS I/O port only. The GOESData instruction has the following parameters:

NOTE When the datalogger sends a command, further processing tasks will be performed only after a response has been received from the HDR GOES Transmitter.

Parameter *& Data Type*	**Enter**	
Dest *Variable or Array*	The variable that holds a result code for the transmission. The result codes are:	
	Code	**Description**
	0	Command executed successfully
	2	Timed out waiting for STX character from transmitter after SDC addressing
	3	Wrong character received after SDC addressing.
	4	Something other than ACK returned when select data buffer command was executed
	5	Timed out waiting for ACK
	6	CS I/O port not available
	7	Random message transmit failure (could be no data in buffer)
Table *Table Name*	The data table from which record(s) should be transmitted.	
TableOption *Constant*	The TableOption indicates which records should be sent from the data table.	
	Code	**Description**
	0	send all records since last execution
	1	send only the most recent record stored in the table
BufferControl *Constant*	The BufferControl parameter specifies which buffer should be used (random or self-timed) and whether data should be overwritten or appended to the existing data. Data stored in the self-timed buffer is transmitted only during a predetermined time frame. Data is erased from the transmitter's buffer after each transmission. Data in the random buffer is transmitted immediately after a threshold has been exceeded. The transmission is randomly repeated to insure it is received.	
	Code	**Description**
	0	Append to self-timed buffer
	1	Overwrite self-timed buffer
	2	Append to random buffer
	3	Overwrite random buffer
	9	Clear random buffer

Parameter *& Data Type*	Enter
DataFormat *Constant*	The DataFormat parameter specifies the format of the data sent to the transmitter

Code	Description
0	CSI FP2 data; 3 bytes per data point
1	Floating point ASCII; 7 bytes per data point
2	18-bit binary integer; 3 bytes per data point, numbers to the right of the decimal are truncated
3	RAWS7; 7 data points:

Data Point	Description
1	total rainfall in inches, format = xx.xxx
2	wind speed MPH, format = xxx
3	vector average wind direction in degrees, format = xxx
4	air temperature in degrees F, format = xxx
5	RH percentage, format = xxx
6	fuel stick temperature in degrees F, format = xxx
7	battery voltage in VDC, format = xx.x

Code	Description
4	Fixed decimal ASCII xxx.x
5	Fixed decimal ASCII xx.xx
6	Fixed decimal ASCII x.xxx
7	Fixed decimal ASCII xxx
8	Fixed decimal ASCII xxxxx

GOESStatus (Dest, StatusCommand)

The GOESStatus instruction is used to request status and diagnostic information from the SAT HDR GOES satellite transmitter.

NOTE When the datalogger sends a command, further processing tasks will be performed only after a response has been received from the HDR GOES Transmitter.

Parameter *& Data Type*	Enter
Dest *Array*	An array that will hold the result codes returned from the transmitter. The size of the array is determined by the option chosen in the StatusCommand.

Code	Description
0	Command executed successfully
1	Checksum failure in response
2	Timeout waiting for STX character after SDC addressing
3	Wrong character (not STX) received after SDC addressing
4	Received a NAK
5	Timed out waiting for ACK
6	CS I/O port not available
7	Transmit random message failure, could be no data
9	Invalid command

Parameter & Data Type	Enter
StatusCommand *Constant*	The StatusCommand specifies the type of information requested from the transmitter.

Code	Description	Array Dim Required
0	Read time	4
1	Status	13
2	Last message status	14
3	Transmit random message	1
4	Read error register	10
5	Reset error register	1
6	Return transmitter to online mode	1

6.4 Output Processing Instructions

Average (Reps, Source, DataType, DisableVar)

This instruction stores the average value over the output interval for the source variable or each element of the array specified.

Parameter & Data Type	Enter
Reps *Constant*	The number of averages to calculate. When Reps is greater than one, the source must be an array.
Source *Variable*	The name of the Variable that is to be averaged.
DataType *Constant*	A code to select the data storage format.

Alpha Code	Numeric Code	Data Format
IEEE4	24	IEEE 4 byte floating point
FP2	7	Campbell Scientific 2 byte floating point

DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, in the Average instruction, when the disable variable is ≠0 the current input is not included in the average. The average that is eventually stored is the average of the inputs that occurred while the disable variable was 0.

Value	Result
0	Process current input
≠ 0	Do not process current input

Covariance (NumVals, Source, DataType, DisableVar, NumCov)

Calculates the covariance of values in an array over time. The Covariance of X and Y is calculated as:

$$Cov(X,Y) = \frac{\sum_{i=1}^{n}\left(X_i \cdot Y_i\right)}{n} - \frac{\sum_{i=1}^{n} X_i \cdot \sum_{i=1}^{n} Y_i}{n^2}$$

where n is the number of values processed over the output interval and X_i and Y_i are the individual values of X and Y.

Parameter& *Data Type*	**Enter**
NumVals *Constant*	The number of elements in the array to include in the covariance calculations
Source *Variable Array*	The variable array that contains the values from which to calculate the covariances. If the covariance calculations are to start at some element of the array later than the first, be sure to include the element number in the source (e.g., X(3)).
DataType *Constant*	A code to select the data storage format. **Alpha Code** | **Numeric Code** | **Data Format** IEEE4 | 24 | IEEE 4 byte floating point FP2 | 7 | Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. When the disable variable is ≠0 the current input is not included in the Covariance. **Value** | **Result** 0 | Process current input ≠ 0 | Do not process current input
NumCov *Constant*	The number of covariances to calculate. The maximum number of covariances is Z/2*(Z+1). Where Z= **NumVals.** If X(1) is the first specified element of the source array, the covariances are calculated and output in the following sequence: X_Cov(1)…X_Cov(Z/2*(Z+1)) = Cov[X(1),X(1)], Cov[X(1),X(2)], Cov[X(1),X(3)], … Cov[X(1),X(Z)], Cov[X(2),X(2)], Cov[X(2),X(3)], … Cov[X(2),X(Z)], … Cov[X(Z),X(Z)]. The first "NumCov" of these possible covariances are output.

FFT (Source, DataType, N, Tau, Units, Option)

The FFT performs a Fast Fourier Transform on a time series of measurements stored in an array. It can also perform an inverse FFT, generating a time series from the results of an FFT. Depending on the output option chosen, the output can be: 0) The real and imaginary parts of the FFT; 1) Amplitude spectrum. 2) Amplitude and Phase Spectrum; 3) Power Spectrum; 4) Power Spectral Density (PSD); or 5) Inverse FFT.

Parameter *& Data Type*	**Enter**
Source *Variable*	The name of the Variable array that contains the input data for the FFT.
DataType *Constant*	A code to select the data storage format. **Alpha Code** | **Numeric Code** | **Data Format** IEEE4 | 24 | IEEE 4 byte floating point FP2 | 7 | Campbell Scientific 2 byte floating point
N *Constant*	Number of points in the original time series. The number of points must be a power of 2 (i.e., 512, 1024, 2048, etc.).
Tau *Constant*	The sampling interval of the time series.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>Units
Constant</td><td>The units for Tau.
<table>
<tr><th>Alpha Code</th><th>Numeric Code</th><th>Units</th></tr>
<tr><td>USEC</td><td>0</td><td>microseconds</td></tr>
<tr><td>MSEC</td><td>1</td><td>milliseconds</td></tr>
<tr><td>SEC</td><td>2</td><td>seconds</td></tr>
<tr><td>MIN</td><td>3</td><td>minutes</td></tr>
</table></td></tr>
<tr><td>Options
Constant</td><td>A code to indicate what values to calculate and output.
<table>
<tr><th>Code</th><th>Result</th></tr>
<tr><td>0</td><td>FFT. The output is N/2 complex data points, i.e., the real and imaginary parts of the FFT. The first pair is the DC component and the Niquist component. This first pair is an exception because the DC and niquist components have no imaginary part.</td></tr>
<tr><td>1</td><td>Amplitude spectrum. The output is N/2 magnitudes. With Acos(wt); A is magnitude.</td></tr>
<tr><td>2</td><td>Amplitude and Phase Spectrum. The output is N/2 pairs of magnitude and phase; with Acos(wt - ϕ); A is amplitude, ϕ is phase ($-\pi$,π).</td></tr>
<tr><td>3</td><td>Power Spectrum. The output is N/2 values normalized to give a power spectrum. With Acos(wt - ϕ), the power is $A^2 / 2$. The summation of the N/2 values yields the total power in the time series signal.</td></tr>
<tr><td>4</td><td>Power Spectral Density (PSD). The output is N/2 values normalized to give a power spectral density (power per herz). The Power Spectrum multiplied by T = N*tau yields the PSD. The integral of the PSD over a given bandwidth yields the total power in that band. Note that the bandwidth of each value is 1/T hertz.</td></tr>
<tr><td>5</td><td>Inverse FFT. The input is N/2 complex numbers, organized as in the output of option 0, which is assumed to be the transform of some real time series. The output is the time series whose FFT would result in the input array.</td></tr>
</table></td></tr>
</table>

T = N*tau: the length, in seconds, of the time series.
Processing field: "FFT,N,tau,option". Tick marks on the x axis are 1/(N*tau) Hertz. N/2 values, or pairs of values, are output, depending upon the option code.

Normalization details:

Complex FFT result i, i = 1 .. N/2: ai*cos(wi*t) + bi*sin(wi*t).
wi = $2\pi(i-1)/T$.
ϕi = atan2(bi,ai) (4 quadrant arctan)
Power(1) = $(a1^2 + b1^2)/N^2$ (DC)
Power(i) = $2*(ai^2 + bi^2)/N^2$ (i = 2..N/2, AC)
PSD(i) = Power(i) * T = Power(i) * N * tau
A1 = $sqrt(a1^2 + b1^2)/N$ (DC)
Ai = $2*sqrt(ai^2 + bi^2)/N$ (AC)

Notes:

- Power is independent of the sampling rate (1/tau) and of the number of samples (N).
- The PSD is proportional to the length of the sampling period (T=N*tau), since the "width" of each bin is 1/T.
- The sum of the AC bins (excluding DC) of the Power Spectrum is the Variance (AC Power) of the time series.
- The factor of 2 in the Power(i) calculation is due to the power series being mirrored about the Niquist frequency N/(2*T); only half the power is represented in the FFT bins below N/2, with the exception of DC. Hence, DC does not have the factor of 2.
- The Inverse FFT option assumes that the data array input is the transform of a real time series. Filtering is performed by taking an FFT on a data set, zeroing certain frequency bins, and then taking the Inverse FFT. Interpolation is performed by taking an FFT, zero padding the result, and then taking the Inverse FFT of the larger array. The resolution in the time domain is increased by the ratio of the size of the padded FFT to the size of the unpadded FFT. This can be used to increase the resolution of a maximum or minimum, as long as aliasing is avoided.

FFT Example

```
Const SIZE_FFT 16
CONST PI 3.141592654
Const CYCLESperT 2
Const AMPLITUDE 3
Const DC 7
Const OPT_FFT 0
CONST PI 3.141592654

dim i
public x(SIZE_FFT),y(SIZE_FFT)

DataTable(Amp,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,1)
EndTable
DataTable(AmpPhase,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,2)
EndTable
DataTable(power,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,3)
EndTable
DataTable(PSD,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,4)
EndTable
DataTable(FFT,1,1)
  fft(x,IEEE4,SIZE_FFT,10 msec,0)
EndTable
```

```
DataTable(IFFT,1,1)                'inverse FFT
  fft(y,IEEE4,SIZE_FFT,10 msec,5)
EndTable

BeginProg

Scan(10, msec,0,SIZE_FFT)
  i=i+1
  X(i) = DC + Sin(PI/8+2*PI*CYCLESperT*i/SIZE_FFT) * AMPLITUDE + Sin(PI/2+PI*i)
Next Scan

CallTable(Amp)
CallTable(AmpPhase)
CallTable(Power)
CallTable(PSD)
CallTable(FFT)
for i = 1 to SIZE_FFT          ' get result back into y()
  y(i) = FFT.x_fft(i,1)
next
CallTable(IFFT)               ' inverse, result is the same as x()

EndProg
```

FieldNames “list of fieldnames”

The FieldNames instructions may be used to override the fieldnames that the CR5000 generates for results sent to the data table. **Fieldnames** must immediately follow the output instruction creating the data fields. Field names are limited to 19 characters. Individual names may be entered for each result generated by the previous output instruction or an array may be used to name multiple fields. When the program is compiled, the CR5000 will determine how many fields are created. If the list of names is greater than the number of fields the extra names are ignored. If the number of fields is greater than the number names in the list of fieldnames, the default names are used for the remaining fields.

Example

```
Sample(4, Temp(1), IEEE4)
FieldNames “IntakeT, CoolerT, PlenumT, ExhaustT”
```

The 4 values from the variable array temp are stored in the output table with the names IntakeT, CoolerT, PlenumT, and ExhaustT.

```
Sample(4, Temp(1), IEEE4)
FieldNames “IntakeT, CoolerT”
```

The 4 values from the variable array Temp are stored in the output table with 2 individual names and the remainder of the default array Temp:
IntakeT, CoolerT, Temp(3), and Temp(4),

```
Sample(4, Temp(1), IEEE4)
FieldNames " IntakeT(2)"
```

The 4 values from the variable array Temp are stored in the output table with IntakeT,an array of 2, and the remainder of the default array Temp: IntakeT(1), IntakeT(2), Temp(3), and Temp(4),

Histogram (BinSelect, DataType, DisableVar, Bins, Form, WtVal, LoLim, UpLim)

Processes input data as either a standard histogram (frequency distribution) or a weighted value histogram.

The standard histogram counts the time that the bin select variable is within a particular sub-range of its specified range. The count in a bin is incremented whenever the bin select input falls within the sub-range associated with the bin. The value that is output to the data table for each bin can either be the accumulated total count for each bin or a relative fraction of time computed by dividing the accumulated total in each bin by the total number of scans. This form of output is also referred to as a frequency distribution.

The weighted value histogram does not add a constant to the bin but instead adds the current value of a variable. That variable name is entered as the weighted value. Each time the instruction is executed, the weighted value is added to a bin. The sub-range that the bin select value is in determines the bin to which the weighted value is added. When the histogram is output, the value accumulated in each bin can be output or the totals can be divided by the TOTAL number of input scans and then output. These values are the contributions of the sub-ranges to the overall weighted value. A common use of a closed form weighted value histogram is the wind speed rose. Wind speed values (the weighted value input) are accumulated into corresponding direction sectors (bin select input).

To obtain the average of the weighted values that occurred while the bin select value was within a particular sub-range, the weighted value output must be divided by the fraction of time that the bin select value was within that particular sub-range (i.e., a standard histogram of the bin select value must also be output; for each bin the weighted value output must be divided by the frequency distribution output).

The frequency distribution histogram is specified by entering a constant in the weighted value parameter. Enter 1 to have frequency output as the fraction of the total time that the bin select value was within the bin range. Enter 100 to have the frequency output as the percent of time. Enter a variable name for the weighted value histogram.

At the user's option, the histogram may be either closed or open. The open form includes all values below the lower range limit in the first bin and all values above the upper range limit in the last bin. The closed form excludes any values falling outside the histogram range.

The difference between the closed and open form is shown in the following example for temperature values:

Lower range limit	10° C	
Upper range limit	30° C	
Number of bins	10	
	Closed Form	Open Form
Range of first bin	10 to <12°	< 12°
Range of last bin	28 to <30°	> 28°

Parameter *& Data Type*	**Enter**
BinSelect *Variable or Array*	The variable that is tested to determine which bin is selected. The histogram 4D instruction requires an array dimensioned with at least as many elements as histogram dimensions.
DataType *Constant*	A code to select the data storage format. **Alpha Code** | **Numeric Code** | **Data Format** IEEE4 | 24 | IEEE 4 byte floating point FP2 | 7 | Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the histogram. The histogram that is eventually stored includes the inputs that occurred while the disable variable was 0. **Value** | **Result** 0 | Process current input ≠ 0 | Do not process current input
Bins *Constant*	The number of bins or subranges to include in the histogram bin select range. The width of each subrange is equal to the bin select range (UpLim - LowLim) divided by the number of bins.
Form *Constant*	The Form argument is 3 digits - **ABC** **Code** | **Form** A = 0 | Reset histogram after each output. A = 1 | Do not reset histogram. B = 0 | Divide bins by total count. B = 1 | Output total in each bin. C = 0 | Open form. Include outside range values in end bins. C = 1 | Closed form. Exclude values outside range. 101 means: Do not reset. Divide bins by total count. Closed form.
WtVal *Constant or Variable*	The variable name of the weighted value. Enter a constant for a frequency distribution of the BinSelect value.
LoLim *Constant*	The lower limit of the range covered by the bin select value.
UpLim *Constant*	The upper limit of the range of the bin select value.

Histogram4D (BinSelect, Source, DataType, DisableVar, Bins1, Bins2, Bins3, Bins4, Form, WtVal, LoLim1, UpLim1, LoLim2, UpLim2, LoLim3, UpLim3, LoLim4, UpLim4)

Processes input data as either a standard histogram (frequency distribution) or a weighted value histogram of up to 4 dimensions.

The description of the Histogram instruction also applies to the Histogram4D instruction. The difference is that the Histogram4D instruction allows up to four bin select inputs (dimensions). The bin select values are specified as variable array. Each of the bin select values has its own range and number of bins. The total number of bins is the product of the number of bins in each dimension (Bins1 x Bins2 x Bins3 x Bins4).

Histogram4D Output Example

'The example program below is an example of using the Histogram4D
'instruction to calculate a 2 dimensional histogram of RPM distribution vs Gear

```
'\\\\\\\\\\\\\\\\\\\\ VARIABLES and CONSTANTS ////////////////////

Public RPM, Gear, Port(4)
Dim Bin(2)

'\\\\\\\\\\\\\\\\\\\\\\\\ OUTPUT SECTION ////////////////////////

DataTable (RPMvsG,1,100)
    DataInterval(0,60,Min,100)
    Histogram4D(Bin(), FP2, 0,4,8, 0, 0,000,1,0.5, 4.5, 0,8000, 0, 0, 0, 0)
    '4 bins for gear, range 0.5 to 4.5; 8 bins for RPM range 0 to 8000
    'Open form so that RPM >8000 is included in 7000 to 8000 bin
EndTable

'\\\\\\\\\\\\\\\\\\\\\\\\\\\\ PROGRAM ////////////////////////////

BeginProg
    Scan (100,mSec,3,0)
    PulseCount (RPM,1,1 ,1,1,0.4225,0)'RPM from pick up on 142 tooth fly wheel
                                                    '60 rpm/142 Hz = 0.42253 ...
    Portget (Port(1),1)                 'There are digital inputs to ports 1 to 4
    Portget (Port(2),2)                 'If C1 is high then the care is in first gear
    Portget (Port(3),3)                 'C2 indicates 2nd gear etc.
    Portget (Port(4),4)

    IF Port(1) then Gear = 1
    If Port(2) Then Gear = 2
    If Port(3) Then Gear = 3
    If Port(4) Then Gear = 4

    Bin(1) = Gear
    Bin(2) = RPM
    CallTable RPMvsG
    Next Scan

EndProg
```

LevelCrossing (Source, DataType, DisableVar, NumLevels, 2ndDim, CrossingArray, 2ndArray, Hysteresis, Option)

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td>Source
Variable or Array</td><td>The variable that is tested to determine if it crosses the specified levels. If a two dimensional level crossing is selected, the source must be an array. The second element of the array (or the next element beyond the one specified for the source) is the variable that is tested to determine the second dimension of the histogram.</td></tr>
<tr><td>DataType
Constant</td><td>A code to select the data storage format.
<table>
<tr><th>Alpha Code</th><th>Numeric Code</th><th>Data Format</th></tr>
<tr><td>IEEE4</td><td>24</td><td>IEEE 4 byte floating point</td></tr>
<tr><td>FP2</td><td>7</td><td>Campbell Scientific 2 byte floating point</td></tr>
</table></td></tr>
<tr><td>DisableVar
Constant, Variable, or Expression</td><td>A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the histogram. The histogram that is eventually stored includes the inputs that occurred while the disable variable was 0.
<table>
<tr><th>Value</th><th>Result</th></tr>
<tr><td>0</td><td>Process current input</td></tr>
<tr><td>≠ 0</td><td>Do not process current input</td></tr>
</table></td></tr>
<tr><td>NumLevels
Constant</td><td>The number levels on which to count crossings. This is the number of bins in which to store the number of crossings for the associated level. The actual levels are input in the Crossing Array. A count is added to a bin when the Source goes from less than the associated level to greater than the associated level (Rising edge or positive polarity). Or if Falling edge or negative polarity is selected, a count occurs if the source goes from greater than the level to less than the level.</td></tr>
<tr><td>2ndDim
Constant</td><td>The second dimension of the histogram. The total number of bins output = NumLevels*2ndDim. Enter 1 for a one dimensional histogram consisting only of the number of level crossings. If 2ndDim is greater than 1, the element of the source array following the one tested for level crossing is used to determine the second dimension.</td></tr>
<tr><td>Crossing Array
Array</td><td>The name of the Array that contains the Crossing levels to check. Because it does not make sense to change the levels while the program is running, the program should be written to load the values into the array once before entering the scan.</td></tr>
<tr><td>2ndArray
Array</td><td>The name of the Array that contains the levels that determine the second dimension. Because it does not make sense to change the levels while the program is running, the program should be written to load the values into the array once before entering the scan.</td></tr>
<tr><td>Hysteresis
Constant</td><td>The minimum change in the source that must occur for a crossing to be counted.</td></tr>
<tr><td>Option
Constant</td><td>The Option code is 3 digits - ABC
<table>
<tr><th>Code</th><th>Form</th></tr>
<tr><td>A = 0</td><td>Count on falling edge (source goes form > level to <level)</td></tr>
<tr><td>A = 1</td><td>Count on rising edge (source goes from < level to >level)</td></tr>
<tr><td>B = 0</td><td>Reset histogram counts to 0 after each output.</td></tr>
<tr><td>B = 1</td><td>Do not reset histogram; continue to accumulate counts.</td></tr>
<tr><td>C = 0</td><td>Divide count in each bin by total number of counts in all bins.</td></tr>
<tr><td>C = 1</td><td>Output total counts in each bin.</td></tr>
</table>
101 means: Count on rising edge, reset count to 0 after each output, output counts.</td></tr>
</table>

Processes data with the Level Crossing counting algorithm. The output is a two dimensional Level Crossing Histogram. One dimension is the levels crossed; the second dimension, if used, is the value of a second input at the time the crossings were detected. The total number of bins output = NumLevels*2ndDim. For a one dimensional level crossing histogram, enter 1 for 2ndDim.

The source value may be the result of a measurement or calculation. Each time the data table with the Level Crossing instruction is called, the source is checked to see if its value has changed from the previous value and if in any change it has crossed any of the specified crossing levels. The instruction can be programmed to count crossings on either the rising edge (source changes from less than the level to greater than the level) or on the falling edge (source changes from greater than the level to less than the level).

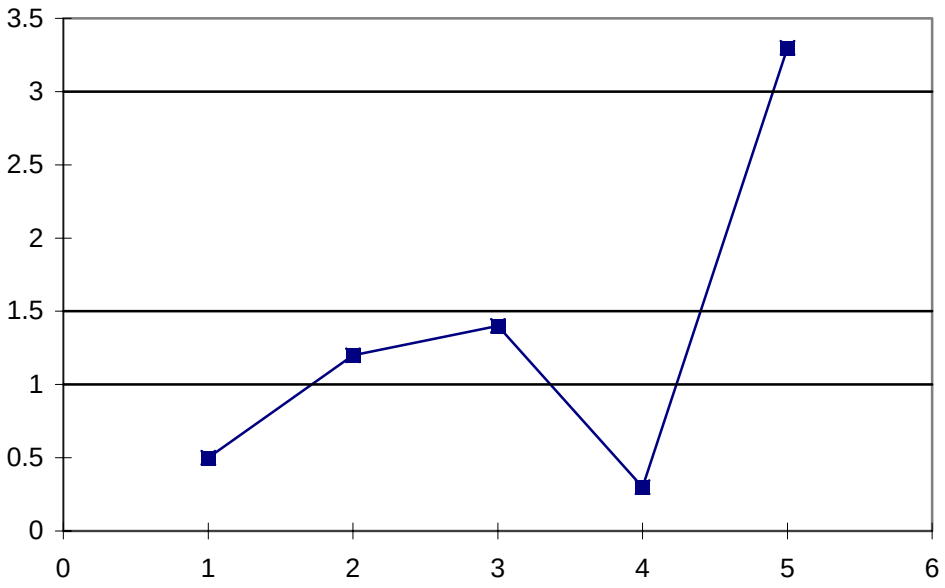


FIGURE 6.4-1. Example Crossing Data

As an example of the level crossing algorithm, assume we have a one dimension 3 bin level crossing histogram (the second dimension =1) and are counting crossings on the rising edge. The crossing levels are 1, 1.5, and 3. Figure 6.4-1 shows some example data. Going through the data point by point:

Point	Source	Action	Bin 1 (level=1)	Bin 2 (level=1.5)	Bin 3 (level=3)
1	0.5	First value, no counts	0	0	0
2	1.2	Add one count to first bin, the signal crossed 1	1	0	0
3	1.4	No levels crossed, no counts	1	0	0
4	0.3	Crossed a level but was falling edge, no counts	1	0	0
5	3.3	Add one count to first, second, and third bins, the signal crossed 1, 1.5 and 3.	2	1	1

The second dimension, when greater than 1, is determined by the value of the element in the source array following the element checked for the crossing. It is the value of this variable at the time the crossings are detected that determines the second dimension.

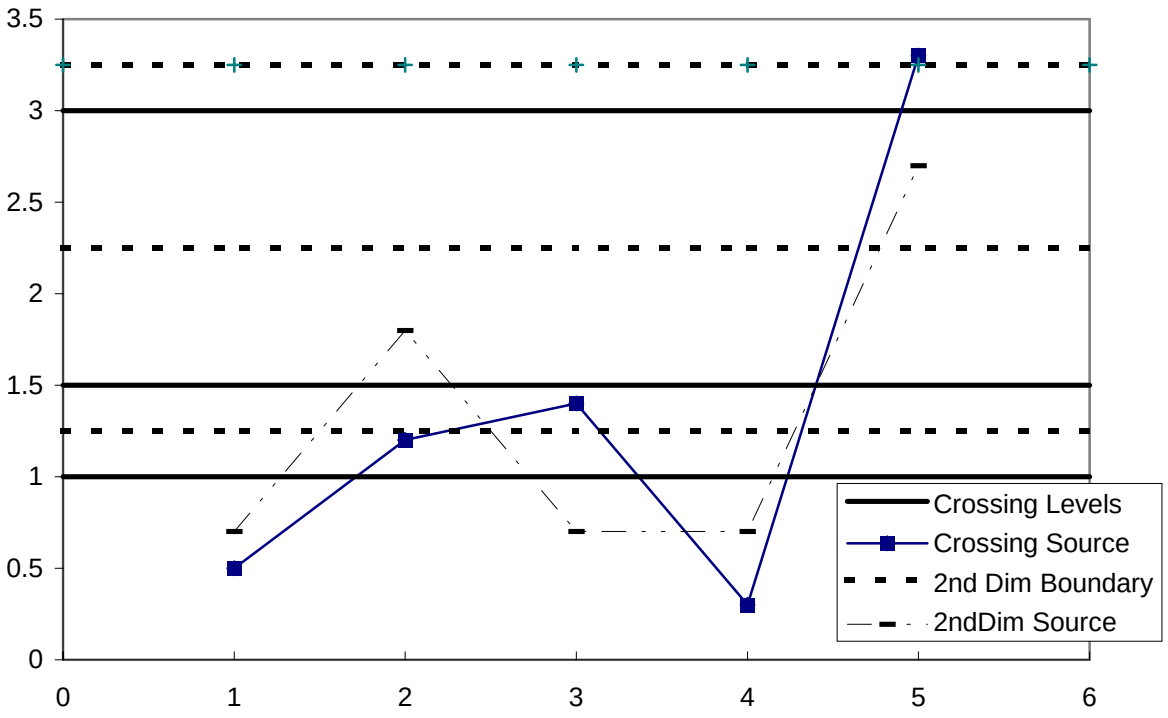


FIGURE 6.4-2. Crossing Data with Second Dimension Value

Point	Crossing Source	2nd Dim Source	Action
1	0.5	.7	First value, no counts
2	1.2	1.8	Add one count to first crossing, second 2D bin, the signal crossed 1

Histogram:

	2D < 1.25	1.25<2D<2.25	2.25<2D<3.25
Cross 1	0	1	0
Cross 1.5	0	0	0
Cross 3	0	0	0

3	1.4	.7	No levels crossed, no counts
4	0.3	.7	Crossed a level but was falling edge, no counts
5	3.3	2.7	Add one count to first, second, and third crossing bins in the third 2D bin, the signal crossed 1, 1.5 and 3.

Histogram:

	2D < 1.25	1.25<2D<2.25	2.25<2D<3.25
Cross 1	0	1	1
Cross 1.5	0	0	1
Cross 3	0	0	1

Note that the first bin of the second dimension is always "open". Any value less than the specified boundary is included in this bin. The last bin of the second dimension is always "closed". It only includes values that are less than its upper boundary and greater than or equal to the upper boundary of the previous bin. If you want the histogram to be "open" on both ends of the second dimension, enter an upper boundary for the last bin that is greater than any possible second dimension source value.

The crossing levels and the boundaries for the second dimension are not specified in the LevelCrossing instruction but are contained in variable arrays. This allows the levels to be spaced in any manner the programmer desires. The arrays need to be dimensioned to at least the same size as the dimensions of the histogram. If a one dimension level crossing histogram is selected (1 entered for the second dimension) the name of the Crossing Array can also be entered for the 2nd Array to avoid declaring an unused array. The program must load the values into these arrays.

The array specifying the boundaries of the second dimension is loaded with the upper limits for each bin. For example, assume the second dimension is 3, and the upper limits loaded into the array containing the second dimension boundaries are 1, 3, and 6.

The value of each element (bin) of the histogram can be either the actual number of times the signal crossed the level associated with that bin or it can be the fraction of the total number of crossings counted that were associated with that bin (i.e., number of counts in the bin divided by total number of counts in all bins).

The hysteresis determines the minimum change in the input that must occur before a crossing is counted. If the value is too small, "crossings" could be counted which are in reality just noise. For example, suppose 5 is a crossing level. If the input is not really changing but is varying from 4.999 to 5.001, a hysteresis of 0 would allow all these crossings to be counted. Setting the hysteresis to 0.1 would prevent this noise from causing counts.

Maximum (Reps, Source, DataType, DisableVar, Time)

This instruction stores the MAXIMUM value that occurs in the specified Source variable over the output interval. Time of maximum value(s) is OPTIONAL output information, which is selected by entering the appropriate code in the time parameter.

Parameter *& Data Type*	**Enter**
Reps *Constant*	The number of maximum values to determine. When repetitions are greater than 1, the source must be an array..
Source *Variable*	The name of the Variable that is the input for the instruction.
DataType *Constant*	A code to select the data storage format. **Alpha Code** | **Numeric Code** | **Data Format** IEEE4 | 24 | IEEE 4 byte floating point FP2 | 7 | Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not checked for a new maximum. The maximum that is eventually stored is the maximum that occurred while the disable variable was 0. **Value** | **Result** 0 | Process current input ≠ 0 | Do not process current input
Time *Constant*	Option to store time of Maximum. When time is output, the maximums for all reps are output first followed by the respective times at which they occurred. **Value** | **Result** 0 | Do not store time 1 | Store time

Minimum (Reps, Source, DataType, DisableVar, Time)

This instruction stores the MINIMUM value that occurs in the specified Source variable over the output interval. Time of minimum value(s) is OPTIONAL output information, which is selected by entering the appropriate code for

Parameter *& Data Type*	**Enter**
Reps *Constant*	The number of minimum values to determine. When repetitions are greater than 1, the source must be an array..
Source *Variable*	The name of the Variable that is the input for the instruction.

Parameter *& Data Type*	**Enter**
DataType *Constant*	A code to select the data storage format. **Alpha Code** / **Numeric Code** / **Data Format** IEEE4 / 24 / IEEE 4 byte floating point FP2 / 7 / Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not checked for a new minimum. The minimum that is eventually stored is the minimum that occurred while the disable variable was 0. **Value** / **Result** 0 / Process current input ≠ 0 / Do not process current input
Time *Constant*	Option to store time of Minimum. When time is output, the minimum values for all repetitions are output first followed by the times at which they occurred. **Value** / **Result** 0 / Do not store time 1 / Store time

RainFlow (Source, DataType, DisableVar, MeanBins, AmpBins, Lowlimit, Highlimit, MinAmp, Form)

Parameter *& Data Type*	**Enter**
Source *Variable*	The variable that is tested to determine which bin is selected
DataType *Constant*	A code to select the data storage format. **Alpha Code** / **Numeric Code** / **Data Format** IEEE4 / 24 / IEEE 4 byte floating point FP2 / 7 / Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the histogram. The histogram that is eventually stored includes the inputs that occurred while the disable variable was 0. **Value** / **Result** 0 / Process current input ≠ 0 / Do not process current input
MeanBins *Constant*	This parameter allows sorting by the mean value of the signal during a stress strain cycle. The number entered is the number of bins or subranges to sort the mean values into. Enter 1 to disregard the signal value and only sort by the amplitude of the signal. The width of each subrange is equal to the HiLimit - LowLimit divided by the number of bins. The lowest bin's minimum value is the low limit and the highest bin's maximum value is the High limit
AmpBins *Constant*	The number of bins or subranges to sort the amplitude of a stress strain cycle into. The width of each subrange is equal to the HiLimit - LowLimit divided by the number of bins.
LowLim *Constant*	The lower limit of the input signal and the Mean Bins.
UpLim *Constant*	The upper limit of the input signal and the Mean Bins.
MinAmp *Constant*	The minimum amplitude that a stress strain cycle must have to be counted.

Parameter *& Data Type*	**Enter**
Form *Constant*	The Form code is 3 digits - **ABC** **Code** / **Form** A = 0 — Reset histogram after each output. A = 1 — Do not reset histogram. B = 0 — Divide bins by total count. B = 1 — Output total in each bin. C = 0 — Open form. Include outside range values in end bins. C = 1 — Closed form. Exclude values outside range. 101 means: Do not reset. Divide bins by total count. Closed form.

Processes data with the rainflow counting algorithm, essential to estimating cumulative damage fatigue to components undergoing stress/strain cycles. Data can be provided by making measurements in either the standard or the burst mode. The Rainflow Instruction can process either a swath of data following the burst mode, or it can process "on line" similar to other processing instructions.

The output is a two dimensional Rainflow Histogram. One dimension is the amplitude of the closed loop cycle (i.e., the distance between peak and valley); the other dimension is the mean of the cycle (i.e., [peak value + valley value]/2). The value of each element (bin) of the histogram can be either the actual number of closed loop cycles that had the amplitude and average value associated with that bin or the fraction of the total number of cycles counted that were associated with that bin (i.e., number of cycles in bin divided by total number of cycles counted).

The user enters the number of mean bins, the number of amplitude bins, and the upper and lower limits of the input data.

The values for the amplitude bins are determined by the difference between the upper and lower limits on the input data and by the number of bins. For example, if the lower limit is 100 and the upper limit is 150, and there are 5 amplitude bins, the maximum amplitude is 150 - 100 = 50. The amplitude change between bins and the upper limit of the smallest amplitude bin is 50/5 = 10. Cycles with an amplitude, A, less than 10 will be counted in the first bin. The second bin is for $10 \leq A < 20$, the third for $20 \leq A < 30$, etc.

In determining the ranges for mean bins, the actual values of the limits are used as well as the difference between them. The lower limit of the input data is also the lower limit of the first mean bin. Assume again that the lower limit is 100, the upper limit 150, and that there are 5 mean bins. In this case the first bin is for cycles which have a mean value M, $100 \leq M < 110$, the second bin $110 \leq M < 120$, etc.

If $C_{m,a}$ is the count for mean range m and amplitude range a, and M and N are the number of mean and amplitude bins respectively, then the output of one repetition is arranged sequentially as ($C_{1,1}$, $C_{1,2}$, ... $C_{1,N}$, $C_{2,1}$, $C_{2,2}$, ... $C_{M,N}$). Multiple repetitions are sequential in memory. Shown in two dimensions, the output is:

$$\begin{matrix} C_{1,1} & C_{1,2} & . & . & . & C_{1,N} \\ C_{2,1} & C_{2,2} & . & . & . & C_{2,N} \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ C_{M,1} & C_{M,2} & . & . & . & C_{M,N} \end{matrix}$$

The histogram can have either open or closed form. In the open form, a cycle that has an amplitude larger than the maximum bin is counted in the maximum bin; a cycle that has a mean value less than the lower limit or greater than the upper limit is counted in the minimum or maximum mean bin. In the closed form, a cycle that is beyond the amplitude or mean limits is not counted.

The minimum distance between peak and valley, MinAmp, determines the smallest amplitude cycle that will be counted. The distance should be less than the amplitude bin width ([high limit - low limit]/no. amplitude bins) or cycles with the amplitude of the first bin will not be counted. However, if the value is too small, processing time will be consumed counting "cycles" which are in reality just noise.

Outputs Generated: No. Mean Bins x No. Amplitude Bins x Reps

Sample (Reps, Source, DataType)

This instruction stores the current value(s) at the time of output from the specified variable or array.

Parameter *& Data Type*	**Enter**		
Reps *Constant*	The number of values to sample. When repetitions are greater than 1, the source must be an array.		
Source *Variable*	The name of the Variable to sample.		
DataType *Constant*	A code to select the data storage format.		
	Alpha Code	**Numeric Code**	**Data Format**
	IEEE4	24	IEEE 4 byte floating point
	FP2	7	Campbell Scientific 2 byte floating point

StdDev (Reps, Source, DataType, DisableVar)

StdDev calculates the standard deviation of the Source(s) over the output interval.

$$\delta(x) = \left(\left(\sum_{i=1}^{i=N} x_i^2 - \left(\sum_{i=1}^{i=N} x_i \right)^2 / N \right) / N \right)^{\frac{1}{2}}$$

where $\delta(x)$ is the standard deviation of x, and N is the number of samples

Parameter *& Data Type*	**Enter**
Reps *Constant*	The number of standard deviations to calculate. When repetitions are greater than 1, the source must be an array.
Source *Variable*	The name of the Variable that is the input for the instruction.
DataType *Constant*	A code to select the data storage format. **Alpha Code** | **Numeric Code** | **Data Format** IEEE4 | 24 | IEEE 4 byte floating point FP2 | 7 | Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the standard deviation. The standard deviation that is eventually stored is the standard deviation of the inputs that occurred while the disable variable was 0. **Value** | **Result** 0 | Process current input ≠ 0 | Do not process current input

Totalize (Reps, Source, DataType, DisableVar)

This instruction stores the total(s) of the values of the source(s) over the given output interval.

Parameter *& Data Type*	**Enter**
Reps *Constant*	The number of totals to calculate. When repetitions are greater than 1, the source must be an array.
Source *Variable*	The name of the Variable that is the input for the instruction.
DataType *Constant*	A code to select the data storage format. **Alpha Code** | **Numeric Code** | **Data Format** IEEE4 | 24 | IEEE 4 byte floating point FP2 | 7 | Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the total. The total that is eventually stored is the total of the inputs that occurred while the disable variable was 0. **Value** | **Result** 0 | Process current input ≠ 0 | Do not process current input

WindVector (Repetitions, Speed/East, Direction/North, DataType, DisableVar, Subinterval, SensorType, OutputOpt)

WindVector processes wind speed and direction from either polar (wind speed and direction) or orthogonal (fixed East and North propellers) sensors. It uses the raw data to generate the mean wind speed, the mean wind vector magnitude, and the mean wind vector direction over an output interval. Two different calculations of wind vector direction (and standard deviation of wind vector direction) are available, one of which is weighted for wind speed.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>Repetitions
Constant</td><td>The name of the data table to clone. The length of this name should be 4 characters or less so the complete names of the worst case tables are retained when collected (see NumCases).</td></tr>
<tr><td>Speed/East
Dir/North
Variables or Arrays</td><td>The source variables for wind speed and direction or, in the case of orthogonal sensors, East and North wind speeds. If repetitions are greater than 1 the source variables must be arrays containing elements for all repetitions.</td></tr>
<tr><td>DataType
Constant</td><td>A code to select the data storage format.
<table>
<tr><th>Alpha Code</th><th>Numeric Code</th><th>Data Format</th></tr>
<tr><td>IEEE4</td><td>24</td><td>IEEE 4 byte floating point</td></tr>
<tr><td>FP2</td><td>7</td><td>Campbell Scientific 2 byte floating point</td></tr>
</table></td></tr>
<tr><td>DisableVar
Constant, Variable, or Expression</td><td>A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the total. The total that is eventually stored is the total of the inputs that occurred while the disable variable was 0.
<table>
<tr><th>Value</th><th>Result</th></tr>
<tr><td>0</td><td>Process current input</td></tr>
<tr><td>≠ 0</td><td>Do not process current input</td></tr>
</table></td></tr>
<tr><td>Subinterval
Constant</td><td>The number of samples per sub-interval calculation. Enter 0 for no sub-interval calculations.</td></tr>
<tr><td>SensorType
Constant</td><td>The type of wind sensors
<table>
<tr><th>Value</th><th>Sensor Type</th></tr>
<tr><td>0</td><td>Speed and Direction</td></tr>
<tr><td>1</td><td>East and North</td></tr>
</table></td></tr>
<tr><td>OutputOpt
Constant</td><td>
<table>
<tr><th>Value</th><th>Outputs (for each rep)</th></tr>
<tr><td>0</td><td>1. Mean horizontal wind speed, S.
2. Unit vector mean wind direction, Θ1.
3. Standard deviation of wind direction, σ(Θ1).

Standard deviation is calculated using the Yamartino algorithm. This option complies with EPA guidelines for use with straight-line Gaussian dispersion models to model plume transport.</td></tr>
<tr><td>1</td><td>1. Mean horizontal wind speed, S.
2. Unit vector mean wind direction, Θ1.</td></tr>
<tr><td>2</td><td>1. Mean horizontal wind speed, S.
2. Resultant mean wind speed, $\overline{U}$.
3. Resultant mean wind direction, Θu.
4. Standard deviation of wind direction, σ(Θu).

This standard deviation is calculated using Campbell Scientific's wind speed weighted algorithm.

Use of the Resultant mean horizontal wind direction is not recommended for straight-line Gaussian dispersion models, but may be used to model transport direction in a variable-trajectory model.</td></tr>
</table></td></tr>
</table>

When used with polar sensors, the instruction does a modulo divide by 360 on wind direction, which allows the wind direction (in degrees) to be 0 to 360, 0 to 540, less than 0, or greater than 540. The ability to handle a negative reading is useful where a difficult to reach wind vane is improperly oriented. For example, a vane outputs 0 degrees at a true reading of 340 degrees. The simplest solution is to enter an offset of -20 in the instruction measuring the wind vane, which results in 0 to 360 degrees following the modulo divide.

When a wind speed sample is 0, the instruction uses 0 to process scalar or resultant vector wind speed and standard deviation, but the sample is not used in the computation of wind direction. The user may not want a sample less than the sensor threshold used in the standard deviation. If this is the case, Write the datalogger program to check wind speed, and if it is less than the threshold set the wind speed variable equal to 0 prior to calling the data table.

Standard deviation can be processed one of two ways: 1) using every sample taken during the output period (enter 0 for the **Subinterval** parameter), or 2) by averaging standard deviations processed from shorter sub-intervals of the output period. Averaging sub-interval standard deviations minimizes the effects of meander under light wind conditions, and it provides more complete information for periods of transition[1].

Standard deviation of horizontal wind fluctuations from sub-intervals is calculated as follows:

$$\sigma(\Theta)=[((\sigma\Theta_1)^2+(\sigma\Theta_2)^2 ...+(\sigma\Theta_M)^2)/M]^{1/2}$$

where $\sigma(\Theta)$ is the standard deviation over the output interval, and $\sigma\Theta_1$ ... $\sigma\Theta_M$ are sub-interval standard deviations.

A sub-interval is specified as a number of scans. The number of scans for a sub-interval is given by:

Desired sub-interval (secs) / scan rate (secs)

For example if the scan rate is 1 second and the Data Interval is 60 minutes, the standard deviation is calculated from all 3600 scans when the sub-interval is 0. With a sub-interval of 900 scans (15 minutes) the standard deviation is the average of the four sub-interval standard deviations. The last sub-interval is weighted if it does not contain the specified number of scans.

Measured raw data:

S_i = horizontal wind speed
Θ_i = horizontal wind direction
Ue_i = east-west component of wind
Un_i = north-south component of wind
N = number of samples

[1] EPA On-site Meteorological Program Guidance for Regulatory Modeling Applications.

Calculations:

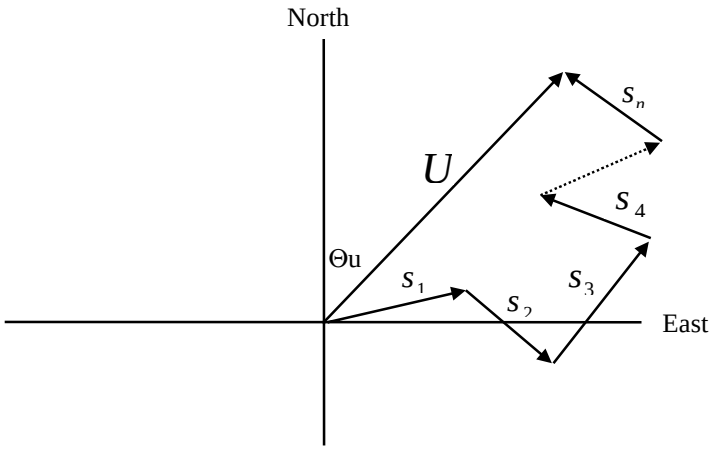


FIGURE 6.4-2. Input Sample Vectors

In Figure 6.4-2, the short, head-to-tail vectors are the input sample vectors described by s_i and Θ_i, the sample speed and direction, or by Ue_i and Un_i, the east and north components of the sample vector. At the end of output interval T, the sum of the sample vectors is described by a vector of magnitude U and direction Θu. If the input sample interval is t, the number of samples in output interval T is $N = T / t$. The mean vector magnitude is $\overline{U} = U / N$.

Scalar mean horizontal wind speed, S:

$$S=(\Sigma s_i)/N$$

where in the case of orthogonal sensors:

$$S_i=(Ue_i^2+Un_i^2)^{1/2}$$

Unit vector mean wind direction, Θ1:

$$\Theta 1=\text{Arctan } (Ux/Uy)$$

where

$$Ux=(\Sigma \sin \Theta_i)/N$$
$$Uy=(\Sigma \cos \Theta_i)/N$$

or, in the case of orthogonal sensors

$$Ux=(\Sigma(Ue_i/U_i))/N$$
$$Uy=(\Sigma(Un_i/U_i))/N$$

$$\text{where } U_i=(Ue_i^2+Un_i^2)^{1/2}$$

Standard deviation of wind direction, σ(Θ1), using Yamartino algorithm:

$$\sigma(\Theta 1)=\text{arc sin}(\varepsilon)[1+0.1547\ \varepsilon^3]$$

where,

$$\varepsilon=[1-((Ux)^2+(Uy)^2)]^{1/2}$$

and Ux and Uy are as defined above.

Resultant mean horizontal wind speed, $\overline{U}$:

$$\overline{U} = (Ue^2 + Un^2)^{1/2}$$

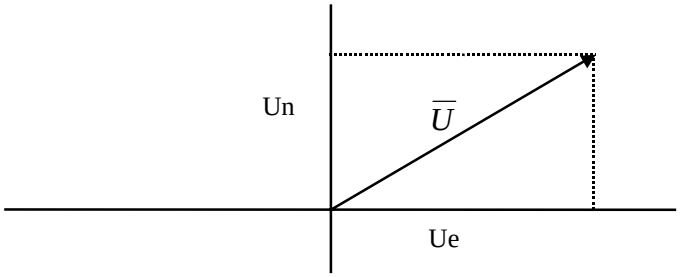


FIGURE 6.4-3. Mean Wind Vector

where for polar sensors:

$$Ue = (\Sigma S_i \text{ Sin } \Theta_i)/N$$
$$Un = (\Sigma S_i \text{ Cos } \Theta_i)/N$$

or, in the case of orthogonal sensors:

$$Ue = (\Sigma Ue_i)/N$$
$$Un = (\Sigma Un_i)/N$$

Resultant mean wind direction, Θu:

$$\Theta u = \text{Arctan } (Ue/Un)$$

Standard deviation of wind direction, σ(Θu), using Campbell Scientific algorithm:

$$\sigma(\Theta u) = 81(1 - \overline{U}/S)^{1/2}$$

The algorithm for σ(θu) is developed by noting (Figure 6.4-4) that

$$\text{Cos } (\Theta_i') = U_i / s_i; \text{ where } \Theta_i' = \Theta_i - \Theta u$$

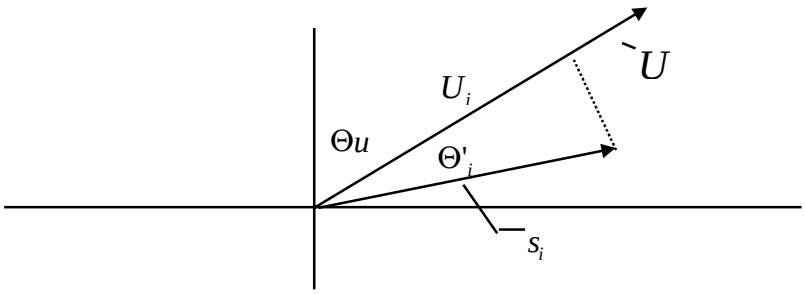


FIGURE 6.2-3. Standard Deviation of Direction

The Taylor Series for the Cosine function, truncated after 2 terms is:

$$\text{Cos } (\Theta_i') \cong 1 - (\Theta_i')^2 / 2$$

For deviations less than 40 degrees, the error in this approximation is less than 1%. At deviations of 60 degrees, the error is 10%.

The speed sample may be expressed as the deviation about the mean speed,

$$s_i = s_i' + S$$

Equating the two expressions for Cos (θ‘) and using the previous equation for s_i;

$$1-(\Theta_i')^2/2 = U_i/(s_i'+S)$$

Solving for $(\Theta_i')^2$, one obtains;

$$(\Theta_i')^2 = 2-2U_i/S-(\Theta_i')^2 s_i'/S+2s_i'/S$$

Summing $(\Theta_i')^2$ over N samples and dividing by N yields the variance of Θu. Note that the sum of the last term equals 0.

$$(\sigma(\Theta u))^2 = \sum_{i=1}^{N}(\Theta_i')^2/N = 2(1-\overline{U}/S)-\sum_{i=1}^{N}((\Theta_i')^2 s_i')/NS$$

The term, $\sum((\Theta_i')^2 s_i')/NS$, is 0 if the deviations in speed are not correlated with the deviation in direction. This assumption has been verified in tests on wind data by CSI; the Air Resources Laboratory, NOAA, Idaho Falls, ID; and MERDI, Butte, MT. In these tests, the maximum differences in

$$\sigma(\Theta u) = (\sum(\Theta_i')^2/N)^{1/2} \text{ and } \sigma(\Theta u) = (2(1-\overline{U}/S))^{1/2}$$

have never been greater tan a few degrees.

The final form is arrived at by converting from radians to degrees (57.296 degrees/radian).

$$\sigma(\Theta u) = (2(1-\overline{U}/S))^{1/2} = 81(1-\overline{U}/S)^{1/2}$$

Section 7. Measurement Instructions

7.1 Voltage Measurements

7.2 Thermocouple Measurements

Measure the Output of Thermocouples and Convert to Temperature.

Resistance Bridge Measurements

Bridge measurements combine an excitation with voltage measurements and are used to measure sensors that change resistance in response to the phenomenon being measured. These sensors include RTDS, thermistors, potentiometers, strain gauges, and pressure and force transducers.

7.3 Half Bridges

7.4 Full Bridges

7.5 Current Excitation

7.6 Excitation / Continuous Analogue Output

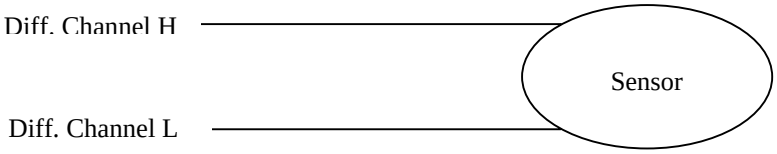
7.7 Self Measurements

7.8 Digital I/O

7.9 Peripheral Devices

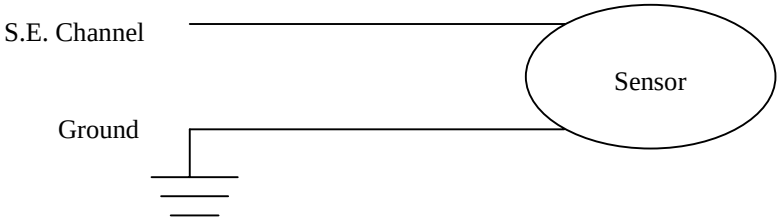
7.1 Voltage Measurements

VoltDiff (Dest, Reps, Range, DiffChan, RevDiff, SettlingTime, Integ, Mult, Offset)



This instruction measures the voltage difference between the HI and Low inputs of a differential channel. Both the high and low inputs must be within ± 5V of the datalogger's ground (See Common Mode Range, Section 3.2). With a multiplier of one and an offset of 0, the result is in millivolts or volts depending on the range selected.

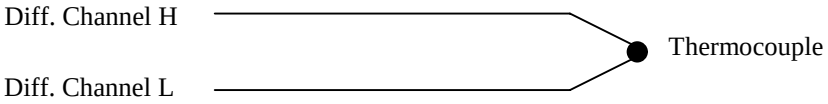
VoltSE (Dest, Reps, Range, SEChan, MeasOfs, SettlingTime, Integ, Mult, Offset)



This instruction measures the voltage at a single ended input with respect to ground. With a multiplier of one and an offset of 0, the result is in millivolts or volts depending on the range selected.

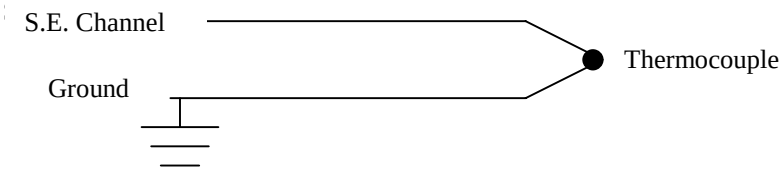
7.2 Thermocouple Measurements

TCDiff (Dest, Reps, Range, DiffChan, TCType, TRef, RevDiff, SettlingTime, Integ, Mult, Offset)



This instruction measures a thermocouple with a differential voltage measurement and calculates the thermocouple temperature (°C) for the thermocouple type selected. The instruction adds the measured voltage to the voltage calculated for the reference temperature relative to 0° C, and converts the combined voltage to temperature in °C. The mV50C and mV200C ranges briefly (10 µs) connect the differential input to reference voltages prior to making the voltage measurement to insure that it is within the common mode range and to test for an open thermocouple.

TCSE (Dest, Reps, Range, SEChan, TCType, TRef, MeasOfs, SettlingTime, Integ, Mult, Offset)



This instruction measures a thermocouple with a single-ended voltage measurement and calculates the thermocouple temperature (°C) for the thermocouple type selected. The instruction adds the measured voltage to the voltage calculated for the reference temperature relative to 0° C, and converts the combined voltage to temperature in °C.

Voltage and Thermocouple Parameters

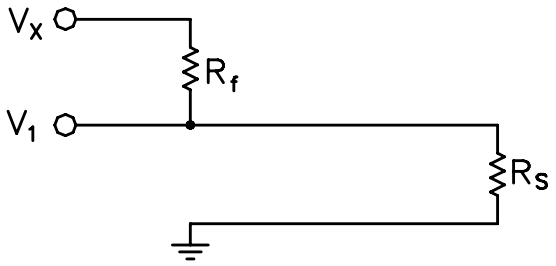
<table>
<tr><td>Parameter
& Data Type</td><td colspan="4">Enter</td></tr>
<tr><td>Dest Variable or Array</td><td colspan="4">The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.</td></tr>
<tr><td>Reps
Constant</td><td colspan="4">The number of repetitions for the measurement or instruction.</td></tr>
<tr><td>Range
Constant</td><td>Alpha Code</td><td>Numeric Code</td><td>Voltage Range</td><td></td></tr>
<tr><td></td><td>mV5000</td><td>0</td><td>± 5000 mV</td><td></td></tr>
<tr><td></td><td>mV1000</td><td>1</td><td>± 1000 mV</td><td></td></tr>
<tr><td></td><td>mV200</td><td>2</td><td>± 200 mV</td><td></td></tr>
<tr><td></td><td>mV50</td><td>3</td><td>± 50 mV</td><td></td></tr>
<tr><td></td><td>mV20</td><td>4</td><td>± 20 mV</td><td></td></tr>
<tr><td></td><td>AutoRange</td><td>5</td><td>mV20 - mV5000</td><td>Selects range (Sect. 3.1)</td></tr>
<tr><td></td><td>mV200C</td><td>20</td><td>± 200 mV</td><td>The mV200C, mV50C, and mV20C</td></tr>
<tr><td></td><td>mV50C</td><td>30</td><td>± 50 mV</td><td>ranges pull the channel into common</td></tr>
<tr><td></td><td>mV20C</td><td>40</td><td>± 20 mV</td><td>mode range and check for open input</td></tr>
<tr><td>DiffChan
Constant</td><td colspan="4">The differential channel number on which to make the first measurement. When Reps are used, subsequent measurements will be automatically made on the following channels. If the channel is entered as a negative number, all reps occur on that channel.</td></tr>
<tr><td>SEChan
Constant</td><td colspan="4">The single-ended channel number on which to make the first measurement. When Reps are used, subsequent measurements will be automatically made on the following channels. If the channel is entered as a negative number, all reps occur on that channel.</td></tr>
<tr><td>TCType

Constant</td><td>Alpha Code</td><td>Numeric Code</td><td colspan="2">Thermocouple Type</td></tr>
<tr><td></td><td>TypeT</td><td>0</td><td colspan="2">Copper Constantan</td></tr>
<tr><td></td><td>TypeE</td><td>1</td><td colspan="2">Chromel Constantan</td></tr>
<tr><td></td><td>TypeK</td><td>2</td><td colspan="2">Chromel Alumel</td></tr>
<tr><td></td><td>TypeJ</td><td>3</td><td colspan="2">Iron Constantan</td></tr>
<tr><td></td><td>TypeB</td><td>4</td><td colspan="2">Platinum Rhodium</td></tr>
<tr><td></td><td>TypeR</td><td>5</td><td colspan="2">Platinum Rhodium</td></tr>
<tr><td></td><td>TypeS</td><td>6</td><td colspan="2">Platinum Rhodium</td></tr>
<tr><td>TRef
Variable</td><td colspan="4">The name of the variable that is the reference temperature for the thermocouple measurements.</td></tr>
</table>

<table>
<tr><td>Parameter
& Data Type</td><td colspan="4">Enter</td></tr>
<tr><td>RevDiff
Constant</td><td>Code</td><td>Value</td><td colspan="2">Result (Reversing requires twice as much time to complete)</td></tr>
<tr><td></td><td>False
True</td><td>0
≠0</td><td colspan="2">Signal is measured with the high side referenced to the low
A second measurement is made after reversing the inputs to cancel offsets</td></tr>
<tr><td>MeasOfs
Constant</td><td>Code</td><td>Value</td><td colspan="2">Result the Ground offset voltage is subtracted from single ended measurements.</td></tr>
<tr><td></td><td>False
True</td><td>0
≠0</td><td colspan="2">Offset voltage is corrected from background calibration
Offset voltage is measured each scan</td></tr>
<tr><td>SettlingTime
Constant</td><td colspan="4">The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement. (1 microsecond resolution)
Entry | Voltage Range | Integration | Settling Time
0 | ± 20 mV | Flash | 200 µS (default)
0 | All but ± 20 mV, | Flash | 100 µS (default)
0 | All | 250 µS | 200 µS (default)
0 | All | _50Hz, _60 Hz | 3 mS (default)
>=100 | All | All | µS entered</td></tr>
<tr><td>Integ
Constant</td><td colspan="4">The time spent on integration in microseconds for each of the channels measured.
Entry | Integration
0, 200 | Flash Conversion; 200 = 2 flash conversions 100 µS apart and averaged
250; 500-16000 | 250 µS; multiples of 500: 250 µS integrations starting each 500 µS and averaged
_60Hz or 16667 | 16,667 µS (reject 60 Hz noise)
_50 Hz or 20000 | 20,000 µS (reject 50 Hz noise)</td></tr>
<tr><td>Mult, Offset
Constant, Variable, Array, or Expression</td><td colspan="4">A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the TCDiff instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.</td></tr>
</table>

7.3 Half Bridges

BrHalf (Dest, Reps, Range, SEChan, ExChan, MeasPEx, ExmV, RevEx, SettlingTime, Integ, Mult, Offset)

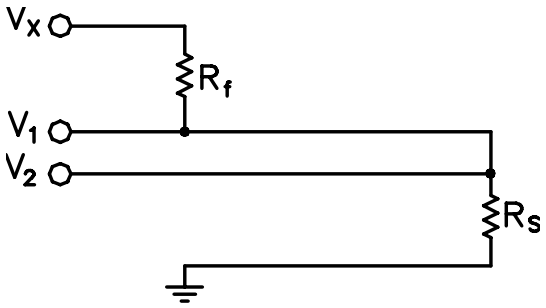


X = result w/mult = 1, offset = 0

$$X = \frac{V_1}{V_x} = \frac{R_s}{R_s + R_f}$$

This Instruction applies an excitation voltage, delays a specified time and then makes a single ended voltage measurement. The result with a multiplier of 1 and an offset of 0 is the ratio of the measured voltage divided by the excitation voltage.

BrHalf3W (Dest, Reps, Range, SEChan, ExChan, MeasPEx, ExmV, RevEx, SettlingTime, Integ, Mult, Offset)



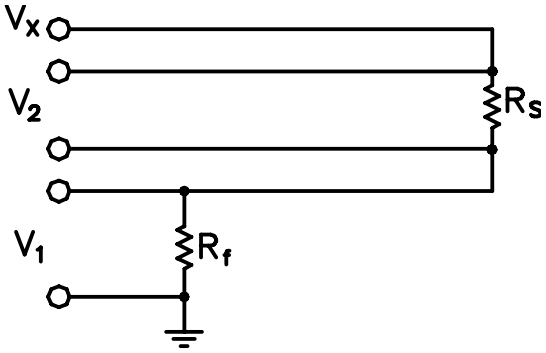
X = result w/mult = 1, offset = 0

$$X = \frac{2V_2 - V_1}{V_x - V_1} = \frac{R_s}{R_f}$$

This Instruction is used to determine the ratio of the sensor resistance to a known resistance using a separate voltage sensing wire from the sensor to compensate for lead wire resistance.

The measurement sequence is to apply an excitation voltage and make two voltage measurements on two adjacent single-ended channels: the first on the reference resistor and the second on the voltage sensing wire from the sensor. The two measurements are used to calculate the resulting value (multiplier = 1, offset = 0) that is the ratio of the voltage across the sensor to the voltage across the reference resistor.

BrHalf4W (Dest, Reps, Range1, Range2, DiffChan, ExChan, MeasPEx, ExmV, RevEx, RevDiff, SettlingTime, Integ, Mult, Offset)



X = result w/mult = 1, offset = 0

$$X = \frac{V_2}{V_1} = \frac{R_s}{R_f}$$

This Instruction applies an excitation voltage and makes two differential voltage measurements, then reverses the polarity of the excitation and repeats the measurements. The measurements are made on sequential channels. The result is the voltage measured on the second channel (V_2) divided by the voltage measured on the first (V_1). The connections are made so that V_1 is the voltage drop across the fixed resistor (R_f), and V_2 is the drop across the sensor (R_s). The result with a multiplier of 1 and an offset of 0 is V_2 / V_1 which equals R_s / R_f.

Half Bridge Parameters

Parameter *& Data Type*	**Enter**
Dest *Variable or Array*	The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.
Reps *Constant*	The number of repetitions for the measurement or instruction.

Range *Constant*

Alpha Code	**Numeric Code**	**Voltage Range**	
mV5000	0	± 5000 mV	
mV1000	1	± 1000 mV	
mV200	2	± 200 mV	
mV50	3	± 50 mV	
mV20	4	± 20 mV	
AutoRange	5	mV20 - mV5000	Selects range (Sect. 3.1)
mV200C	20	± 200 mV	The mV200C, mV50C, and mV20C
mV50C	30	± 50 mV	ranges pull the channel into common
mV20C	40	± 20 mV	mode range and check for open input

SEChan *Constant*	The single-ended channel number on which to make the first measurement. When Reps are used, subsequent measurements will be automatically made on the following single-ended channels. If the channel is entered as a negative number, all reps occur on that channel.

ExChan *Constan*

Enter the excitation channel number to excite the first measurement.

Alpha Code	**Code/ Channel**	**Result**
VX1	1	Switched excitation channels, are switched to the excitation voltage.
VX2	2	for the measurement and switched off between measurements.
VX3	3	
VX4	4	

MeasPEx *Constant*	The number of sensors to excite with the same excitation channel before automatically advancing to the next excitation channel. To excite all the sensors with the same excitation channel, the number should equal the number of Reps.
ExmV *Constant*	The excitation voltage in millivolts. Allowable range ± 5000 mV. **RevEx** may be used to excite with both a positive and negative polarity to cancel offset voltages.

RevEx *Constant*

Code	**Value**	**Result** (Reversing requires twice as much time to complete)
False	0	Excite only with the excitation voltage entered
True	≠0	A second measurement is made with the voltage polarity reversed to cancel offsets

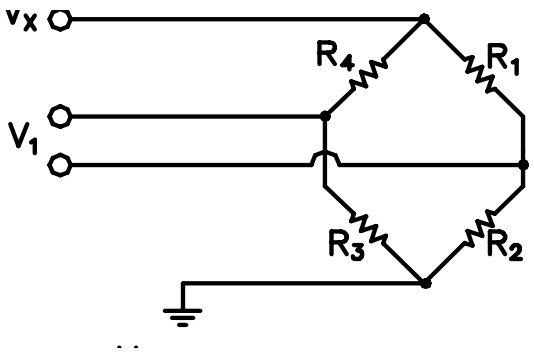
RevDiff *Constant*

Code	**Value**	**Result** (Reversing requires twice as much time to complete)
False	0	Signal is measured with the high side referenced to the low
True	≠0	A second measurement is made after reversing the inputs to cancel offsets

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>SettlingTime
Constant</td><td>The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement. (1 microsecond resolution)
<table>
<tr><th>Entry</th><th>Voltage Range</th><th>Integration</th><th>Settling Time</th></tr>
<tr><td>0</td><td>± 20 mV</td><td>Flash</td><td>200 µS (default)</td></tr>
<tr><td>0</td><td>All but ± 20 mV,</td><td>Flash</td><td>100 µS (default)</td></tr>
<tr><td>0</td><td>All</td><td>250 µS</td><td>200 µS (default)</td></tr>
<tr><td>0</td><td>All</td><td>_50Hz, _60 Hz</td><td>3 mS (default)</td></tr>
<tr><td>>=100</td><td>All</td><td>All</td><td>µS entered</td></tr>
</table></td></tr>
<tr><td>Integ
Constant</td><td>The time spent on integration in microseconds for each of the channels measured.
<table>
<tr><th>Entry</th><th>Integration</th></tr>
<tr><td>0, 200</td><td>Flash Conversion; 200 = 2 flash conversions 100 µS apart and averaged</td></tr>
<tr><td>250; 500-16000</td><td>250 µS; multiples of 500: 250 µS integrations starting each 500 µS and averaged</td></tr>
<tr><td>_60Hz or 16667</td><td>16,667 µS (reject 60 Hz noise)</td></tr>
<tr><td>_50 Hz or 20000</td><td>20,000 µS (reject 50 Hz noise)</td></tr>
</table></td></tr>
<tr><td>Mult, Offset
Constant, Variable, Array, or Expression</td><td>A multiplier and offset by which to scale the raw results of the measurement.</td></tr>
</table>

7.4 Full Bridges

BrFull (Dest, Reps, Range, DiffChan, ExChan, MeasPEx, ExmV, RevEx, RevDiff, SettlingTime, Integ, Mult, Offset)

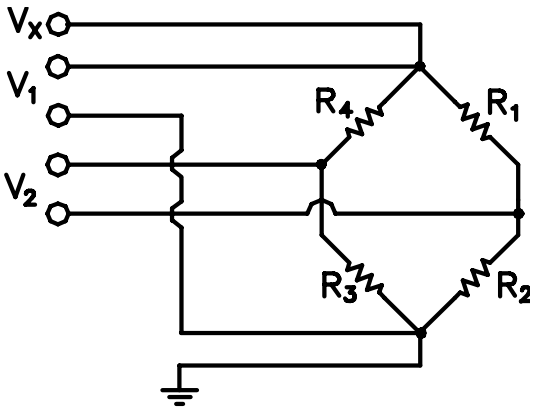


X = result w/mult = 1, offset = 0

$$X = 1000\frac{V_1}{V_x} = 1000\left(\frac{R_3}{R_3+R_4} - \frac{R_2}{R_1+R_2}\right)$$

This Instruction applies an excitation voltage to a full bridge and makes a differential voltage measurement of the bridge output. The resulting value (multiplier = 1, offset = 0) is the measured voltage in millivolts divided by the excitation voltage in volts (i.e., millivolts per volt).

BrFull6W (Dest, Reps, Range1, Range2, DiffChan, ExChan, MeasPEx, ExmV, RevEx, RevDiff, SettlingTime, Integ, Mult, Offset)



X = result w/mult = 1, offset = 0

$$X = 1000\frac{V_2}{V_1} = 1000\left(\frac{R_3}{R_3+R_4} - \frac{R_2}{R_1+R_2}\right)$$

This Instruction applies an excitation voltage and makes two differential voltage measurements. The measurements are made on sequential channels. The result is the voltage measured on the second channel (V_2) divided by the voltage measured on the first (V_1). The result is 1000 times V_2 / V_1 or millivolts output per volt of excitation. The connections are made so that V_1 is the measurement of the voltage drop across the full bridge, and V_2 is the measurement of the bridge output.

Full Bridge Parameters

Parameter *& Data Type*	**Enter**			
Dest *Variable or Array*	The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.			
Reps *Constant*	The number of repetitions for the measurement or instruction.			
Range *Constant*	**Alpha Code**	**Numeric Code**	**Voltage Range**	
	mV5000	0	± 5000 mV	
	mV1000	1	± 1000 mV	
	mV200	2	± 200 mV	
	mV50	3	± 50 mV	
	mV20	4	± 20 mV	
	AutoRange	5	mV20 - mV5000	Selects range (Sect. 3.1)
	mV200C	20	± 200 mV	The mV200C, mV50C, and mV20C
	mV50C	30	± 50 mV	ranges pull the channel into common
	mV20C	40	± 20 mV	mode range and check for open input
DiffChan *Constant*	The differential channel number on which to make the first measurement. When **Reps** are used, subsequent measurements will be automatically made on the following differential channels. If the channel is entered as a negative number, all reps occur on that channel.			

ExChan *Constan*

Enter the excitation channel number to excite the first measurement.

Alpha Code	Code/ Channel	Result
VX1	1	Switched excitation channels, are switched to the excitation voltage.
VX2	2	for the measurement and switched off between measurements.
VX3	3	
VX4	4	

MeasPEx *Constant*

The number of sensors to excite with the same excitation channel before automatically advancing to the next excitation channel. To excite all the sensors with the same excitation channel, the number should equal the number of Reps.

ExmV *Constant*

The excitation voltage in millivolts. Allowable range ± 5000 mV. **RevEx** may be used to excite with both a positive and negative polarity to cancel offset voltages.

RevEx *Constant*

Code	Value	**Result** (Reversing requires twice as much time to complete)
False	0	Excite only with the excitation voltage entered
True	≠0	A second measurement is made with the voltage polarity reversed to cancel offsets

RevDiff *Constant*

Code	Value	**Result** (Reversing requires twice as much time to complete)
False	0	Signal is measured with the high side referenced to the low
True	≠0	A second measurement is made after reversing the inputs to cancel offsets

SettlingTime *Constant*

The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement. (1 microsecond resolution)

Entry	Voltage Range	Integration	Settling Time
0	± 20 mV	Flash	200 µS (default)
0	All but ± 20 mV,	Flash	100 µS (default)
0	All	250 µS	200 µS (default)
0	All	_50Hz, _60 Hz	3 mS (default)
>=100	All	All	µS entered

Integ *Constant*

The time spent on integration in microseconds for each of the channels measured.

Entry	Integration
0, 200	Flash Conversion; 200 = 2 flash conversions 100 µS apart and averaged
250; 500-16000	250 µS; multiples of 500: 250 µS integrations starting each 500 µS and averaged
_60Hz or 16667	16,667 µS (reject 60 Hz noise)
_50 Hz or 20000	20,000 µS (reject 50 Hz noise)

Mult, Offset *Constant, Variable, Array, or Expression*

A multiplier and offset by which to scale the raw results of the measurement.

7.5 Current Excitation

Resistance (Dest, Reps, Range, DiffChan, IxChan, MeasPEx, ExuA, RevEx, RevDiff, SettlingTime, Integ, Mult,Offset)

The Resistance instruction applies a precision current excitation and measures a differential voltage. The result is the measured voltage divided by the excitation current. The measurement can be used to measure resistance or, with some sensors, to provide a current excitation for measuring a full bridge.

The excitation current is set in the instruction with a maximum current of 2500 microamps. The compliance voltage is ± 5 volts. The compliance voltage is the maximum or minimum voltage to which the excitation can go in order to drive the current specified. Some of the consequences of this are:

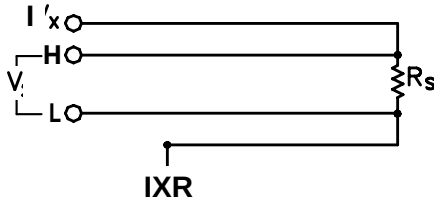
The maximum resistance that can be excited with 2500 μA= 5V/0.0025A = 2000 Ohms.

The maximum resistance that can be excited with 1000 μA = 5V/0.001A = 5000 Ohms.

The minimum resistance required to fill the 20 mV range = 0.02V/0.0025A = 8 Ohms.

The resistance instruction can be used for full bridge measurements were the sensor is calibrated for current excitation or the resistance across the bridge is constant (or assumed to be). The 2.5 mA maximum current excitation provides for reasonable output from the full bridge when the bridge resistance is in the range of 2000 to 5000 ohms. For lower resistance bridges such as 120 ohm or 350 ohm, measuring with the voltage excitation full bridge BrFull will allow a higher output from the bridge.

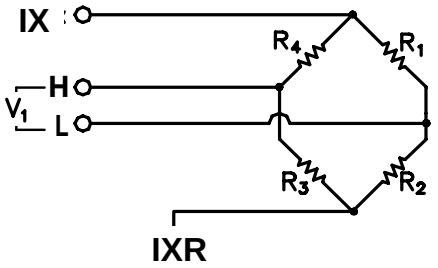
Resistance *used to measure resistance*



X = result w/mult = 1, offset = 0

$$X = \frac{V}{I_x} = R_s$$

Resistance *used to measure full bridge*



X = result w/mult = 1, offset = 0

$$X = \frac{V_1}{I_x} = R_{bridge}\left(\frac{R_3}{R_3+R_4} - \frac{R_2}{R_1+R_2}\right)$$

$$= \frac{R_3(R_1+R_2) - R_2(R_3+R_4)}{R_1+R_2+R_3+R_4}$$

$$R_1 = \frac{-R_2R_4 - X(R_2+R_3+R_4)}{X-R_3}$$

$$R_2 = \frac{R_1R_3 - X(R_1+R_3+R_4)}{X+R_4}$$

$$R_3 = \frac{-R_2R_4 - X(R_1+R_2+R_4)}{X-R_1}$$

$$R_4 = \frac{R_1R_3 - X(R_1+R_2+R_3)}{X+R_2}$$

Resistance Parameters

Parameter *& Data Type*	**Enter**			
Dest *Variable or Array*	The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.			
Reps *Constant*	The number of repetitions for the measurement or instruction.			
Range *Constant*	**Alpha Code**	**Numeric Code**	**Voltage Range**	
	mV5000	0	± 5000 mV	
	mV1000	1	± 1000 mV	
	mV200	2	± 200 mV	
	mV50	3	± 50 mV	
	mV20	4	± 20 mV	
	AutoRange	5	mV20 - mV5000	Selects range (Sect. 3.1)
DiffChan *Constant*	The differential channel number on which to make the first measurement. When **Reps** are used, subsequent measurements will be automatically made on the following differential channels. If the channel is entered as a negative number, all reps occur on that channel.			
IxChan *Constant*	Enter the current excitation channel number to use to excite the first measurement.			
	Alpha Code	**Code/ Channel**	**Result**	
	IX1	1	Current Excitation Channels	
	IX2	2		
	IX3	3		
	IX4	4		
MeasPEx *Constant*	The number of sensors to excite with the same excitation channel before automatically advancing to the next excitation channel. To excite all the sensors with the same excitation channel, the number should equal the number of Reps. With current excitation, the sensors must be wired in series when more than one measurement per excitation is made.			
ExuA *Constant*	The excitation current in microamps. The allowable range is ±2500 µA. **RevEx** may be used to excite with both a positive and negative polarity to cancel offset voltages.			
RevEx *Constant*	**Code**	**Value**	**Result** (Reversing requires twice as much time to complete)	
	False	0	Excite only with the current polarity entered	
	True	≠0	A second measurement is made with the voltage polarity reversed to cancel offsets	
RevDiff *Constant*	**Code**	**Value**	**Result** (Reversing requires twice as much time to complete)	
	False	0	Signal is measured with the high side referenced to the low	
	True	≠0	A second measurement is made after reversing the inputs to cancel offsets	

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>SettlingTime
Constant</td><td>The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement. (1 microsecond resolution)
<table>
<tr><th>Entry</th><th>Voltage Range</th><th>Integration</th><th>Settling Time</th></tr>
<tr><td>0</td><td>± 20 mV</td><td>Flash</td><td>200 µS (default)</td></tr>
<tr><td>0</td><td>All but ± 20 mV,</td><td>Flash</td><td>100 µS (default)</td></tr>
<tr><td>0</td><td>All</td><td>250 µS</td><td>200 µS (default)</td></tr>
<tr><td>0</td><td>All</td><td>_50Hz, _60 Hz</td><td>3 mS (default)</td></tr>
<tr><td>>=100</td><td>All</td><td>All</td><td>µS entered</td></tr>
</table></td></tr>
<tr><td>Integ
Constant</td><td>The time spent on integration in microseconds for each of the channels measured.
<table>
<tr><th>Entry</th><th>Integration</th></tr>
<tr><td>0, 200</td><td>Flash Conversion; 200 = 2 flash conversions 100 µS apart and averaged</td></tr>
<tr><td>250; 500-16000</td><td>250 µS; multiples of 500: 250 µS integrations starting each 500 µS and averaged</td></tr>
<tr><td>_60Hz or 16667</td><td>16,667 µS (reject 60 Hz noise)</td></tr>
<tr><td>_50 Hz or 20000</td><td>20,000 µS (reject 50 Hz noise)</td></tr>
</table></td></tr>
<tr><td>Mult, Offset
Constant, Variable, Array, or Expression</td><td>A multiplier and offset by which to scale the raw results of the measurement.</td></tr>
</table>

7.6 Excitation/Continuous Analogue Output

ExciteCAO (CAOChan, CAOmV, Boost, FastUpdate)

ExciteCAO sets the voltage of a Continuous Analogue Output channel.

The datalogger has two continuous analogue out (CAO) channels. These channels are used to provide a continuous analogue output. The output may be used for proportional control. The CAO channels can source 15 mA and sink 5 mA. If Boost is enabled, sink is increased to 15 mA.

To ensure the CAO voltage settles to with in the ±10 mV accuracy specification of the CAOmV value, a settling time of ten time constants (~2.6 ms) is required. This time is required for the worst case voltage changes (e.g., -5000 to +5000 mV). In instances where the change in CAO voltage from one scan to the next is small *and* the time required for measurements is approaching the scan time, the FastScan parameter can be used to shorten the execution time. When FastScan is true the settling time is shortened to approximately one time constant and the time required to execute the instruction to approximately 300 µs. In one time constant the signal settles to 63% of the step change which will not be within ±10 mV for large steps.

Parameter *& Data Type*	**Enter**
CAOChan *Constant*	The CAO channel to set. **Alpha** / **Numeric** / **Description** CAO1 / 1 / CAO channel 1 CAO2 / 2 / CAO channel 2
CAOmV *Constant or Variable*	The voltage, in millivolts, to apply to the CAO Channel. The allowable range is ±5000 mV. The voltage can be constant or a variable can be used to vary the analogue output in response to time or measurements.
Boost *Constant*	Sets the current that the CAO can sink (The CAO can always source up to 15 mA.) **Alpha Code** / **Value** / **Result** False / 0 / Boost off: Maximum current sink 5 mA True / ≠0 / Boost Enabled: Maximum current sink 15 mA
FastUpdate	Shortens the execution time of the instruction and the settling time for the excitation. **Alpha Code** / **Value** / **Result** False / 0 / Settle 10 time constants (~2.6 ms) True / ≠0 / Settle 1 time constant (~300 µs)

ExciteI (IxChan, IxuA, XDelay)

This instruction sets the specified current excitation channel to the current specified.

Parameter *& Data Type*	**Enter**
IxChan *Constant*	Enter the current excitation channel number to use to excite the first measurement. **Alpha Code** / **Code/ Channel** / **Result** IX1 / 1 / Current Excitation Channels (IxChan) IX2 / 2 IX3 / 3 IX4 / 4
ExuA *Constant*	The excitation current in microamps. The allowable range is ±2500 µA.
XDelay	Specifies the length of time the current channel is enabled, after which, the channel is set low and the datalogger moves on to the next instruction. If XDelay is set to 0, the excitation will be on until the end of the program scan or until another instruction sets an excitation voltage, CAO, or current channel.

ExciteV (ExChan, ExmV, XDelay)

This instruction sets the specified switched voltage excitation channel to the voltage specified. The XDelay parameter is used to specify the length of time the excitation channel is enabled, after which, the channel is set low and the datalogger moves on to the next instruction. If the SettlingTime is set to 0, the excitation channel will be enabled and the voltage will be held until the end of the program scan or until another instruction sets an excitation voltage, CAO, or current channel.

Parameter *& Data Type*	**Enter**		
ExChan *Constan*	Enter the excitation channel number to excite the first measurement.		
	Alpha Code	**Code/ Channel**	**Result**
	VX1	1	Switched excitation channels, are switched to the excitation voltage.
	VX2	2	for the measurement and switched off between measurements.
	VX3	3	
	VX4	4	
ExmV *Constant*	The excitation voltage in millivolts. Allowable range ± 5000 mV.		
XDelay	Specifies the length of time the excitation is enabled, after which, the channel is set low and the datalogger moves on to the next instruction. If XDelay is set to 0, the excitation will be on until the end of the program scan or until another instruction sets an excitation voltage, CAO, or current channel.		

7.7 Self Measurements

Battery (Dest)

This instruction reads the battery voltage and stores it in the destination variable. The units for battery voltage are volts.

PanelTemp (Dest, Integ)

This instruction measures the panel temperature in °C.

Parameter *& Data Type*	**Enter**	
Dest *Variable or Array*	The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.	
Integ *Constant*	The time spent on integration in microseconds for each of the channels measured.	
	Entry	**Integration**
	0, 200	Flash Conversion; 200 = 2 flash conversions 100 µS apart and averaged
	250; 500-16000	250 µS; multiples of 500: 250 µS integrations starting each 500 µS
	_60Hz or 16667	16,667 µS (reject 60 Hz noise)
	_50 Hz or 20000	20,000 µS (reject 50 Hz noise)

Calibrate (Dest, AllRanges)

The Calibrate instruction places the CR5000 self calibration under program control. Placing the Calibrate instruction in the program disables the automatic self calibration that is normally run in the background (Section 3.8).

The Calibrate parameters are optional and are only used to place the results of the calibration in a variable array. With no parameters the Calibrate instruction does not return data.

Parameter *& Data Type*	**Enter**
Dest *Array*	If present the array must contain at least 60 elements (more if excitation is used in the program. With no parameters no data are returned.
AllRanges *Constant*	Option to calibrate ranges not being used. Dest must be entered before AllRanges parameter

Alpha Code	**Value**	**Result**
False	=0	Calibrate only Voltage ranges used in program
True	≠0	Calibrate all Voltage ranges

NOTE In most cases the background calibration is adequate and the calibrate instruction should not be used in the program.

There are three valid situations for using the Calibrate instruction:

1) With the normal set of measurements there is not time for the Calibration to run in the background but the program can periodically stop making measurements and run the calibration in a separate scan.

2) The CR5000 will experience extremely rapid temperature change and the Calibration instruction is run to update the calibration before each set of measurements.

3) The program is run by a repair technician specifically to get the results of the calibration. (Calibration values are also available in the status table without running a special program.)

If there is not enough time leftover in a fast scan for the background calibration to run, the message: “Warning when Fast Scan x is running background calibration will be disabled.” will be returned when the program is compiled (x is the number of the fast scan where the first fast scan entered in the program is 1, the next scan is 2, etc.) If you see this message you have the options of letting the scan run without any calibration (if the temperature remains constant there will be little shift, Section 3.8), reducing the number of measurements or the time it takes to make them (e.g., shorten the integration), or periodically changing to a different scan to run the calibration.

In cases of rapid temperature change, such as bringing a vehicle from equilibrium at -30°C to a hot Arizona day, running the Calibration instruction in the program can improve the accuracy of the measurements. It has to be a rapid change to require this; the background calibration filters new readings and has a time constant (63% response to a step change) of approximately 36 seconds. When the calibration instruction is run in the program the calibration is completely updated each time the instruction is run.

Unless the AllRanges option is selected, the calibrate instruction only measures the range and integration combinations that occur in the measurements in the program. For the 250 μs and zero integration calibrations multiple measurements

are averaged for the calibration values. The 250 μs integration calibration averages five measurements and the zero integration calibration averages ten measurements.

The Calibration instruction can occur in a fast scan or in a slow sequence scan. In a fast scan the entire calibration is completed at once. In a slow sequence scan the calibration measurements are separated into sections that can be spliced on to the end of fast sequence scans.

If it is necessary to update the calibration more rapidly than is done by the background calibration, try running the Calibrate instruction in the fast scan with the measurements. If there isn't time for it to run there it can be placed in a slow sequence scan, but remember, unless the slow scan is faster than about 40 seconds the calibration isn't being updated any faster than with the background calibration.

Running Calibrate in a slow sequence scan is not an option when there is not time for the automatic background calibration. The instruction requires more time because of the multiple measurements for the 250 μs and zero integrations.

When the results of the calibration are placed in an array, the array must have at least 60 elements, more if the program contains instructions which use excitations. The calibration values will be in the following order, followed by the calibrations of the excitations if any. If a range is not calibrated, 0 will be returned for the gain and offset.

Table 7.7-1. Calibrate Return Value Decode

Array Element	Description
1	zero integrate 5000 mV single ended offset
2	zero integrate 5000 mV differential offset
3	zero integrate 5000 mV gain
4	zero integrate 1000 mV single ended offset
5	zero integrate 1000 mV differential offset
6	zero integrate 1000 mV gain
7	zero integrate 200 mV single ended offset
8	zero integrate 200 mV differential offset
9	zero integrate 200 mV gain
10	zero integrate 50 mV single ended offset
11	zero integrate 50 mV differential offset
12	zero integrate 50 mV gain
13	zero integrate 20 mV single ended offset
14	zero integrate 20 mV differential offset
15	zero integrate 20 mV gain
16	250 μSec integrate 5000 mV single ended offset
17	250 μSec integrate 5000 mV differential offset
18	250 μSec integrate 5000 mV gain
19	250 μSec integrate 1000 mV single ended offset
20	250 μSec integrate 1000 mV differential offset
21	250 μSec integrate 1000 mV gain
22	250 μSec integrate 200 mV single ended offset
23	250 μSec integrate 200 mV differential offset
24	250 μSec integrate 200 mV gain
25	250 μSec integrate 50 mV single ended offset
26	250 μSec integrate 50 mV differential offset
27	250 μSec integrate 50 mV gain

28	250 μSec integrate 20 mV single ended offset
29	250 μSec integrate 20 mV differential offset
30	250 μSec scriptsizeintegrate 20 mV gain
31	60 Hz rejection 5000 mV single ended offset
32	60 Hz rejection 5000 mV differential offset
33	60 Hz rejection 5000 mV gain
34	60 Hz rejection 1000 mV single ended offset
35	60 Hz rejection 1000 mV differential offset
36	60 Hz rejection 1000 mV gain
37	60 Hz rejection 200 mV single ended offset
38	60 Hz rejection 200 mV differential offset
39	60 Hz rejection 200 mV gain
40	60 Hz rejection 50 mV single ended offset
41	60 Hz rejection 50 mV differential offset
42	60 Hz rejection 50 mV gain
43	60 Hz rejection 20 mV single ended offset
44	60 Hz rejection 20 mV differential offset
45	60 Hz rejection 20 mV gain
46	50 Hz rejection 5000 mV single ended offset
47	50 Hz rejection 5000 mV differential offset
48	50 Hz rejection 5000 mV gain
49	50 Hz rejection 1000 mV single ended offset
50	50 Hz rejection 1000 mV differential offset
51	50 Hz rejection 1000 mV gain
52	50 Hz rejection 200 mV single ended offset
53	50 Hz rejection 200 mV differential offset
54	50 Hz rejection 200 mV gain
55	50 Hz rejection 50 mV single ended offset
56	50 Hz rejection 50 mV differential offset
57	50 Hz rejection 50 mV gain
58	50 Hz rejection 20 mV single ended offset
59	50 Hz rejection 20 mV differential offset
60	50 Hz rejection 20 mV gain

7.8 Digital I/O

PeriodAvg (Dest, Reps, Range, SEChan, Threshold, PAOption, Cycles, Timeout, Mult, Offset)

This instruction measures the period of a signal on any single-ended input channel. The specified number of Cycles are timed with a resolution of 70 ns, making the resolution of the period measurement 70 ns divided by the number of Cycles chosen.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>Dest
Variable or Array</td><td>The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.</td></tr>
<tr><td>Reps
Constant</td><td>The number of repetitions for the instruction on adjacent channels.</td></tr>
<tr><td>Range
Constant</td><td>The voltage range for the measurement, which determines the gain applied to the signal prior to a zero-crossing detector. Maximum frequency decreases with increasing gain.
<table>
<tr><th rowspan="2">Range Code</th><th rowspan="2">Gain</th><th colspan="2">Signal (pk-pk)[1]</th><th rowspan="2">Minimum Pulse Width</th><th rowspan="2">Maximum Frequency[2]</th></tr>
<tr><th>Min</th><th>Max</th></tr>
<tr><td>mV5000</td><td>0.8</td><td>600 mV</td><td>10.0 V</td><td>2.5 µs</td><td>200 kHz</td></tr>
<tr><td>mV1000</td><td>4.0</td><td>100 mV</td><td>2.0 V</td><td>5.0 µs</td><td>100 kHz</td></tr>
<tr><td>mV200</td><td>20</td><td>4 mV</td><td>2.0 V</td><td>25 µs</td><td>20 kHz</td></tr>
</table>
[1] Signals must cross threshold to trigger the voltage comparator.
[2] Maximum frequency equals 1/(Twice Minimum Pulse Width) for 50% duty cycle signals.</td></tr>
<tr><td>SEChan
Constant</td><td>The single-ended channel number on which to make the first measurement. If the channel is entered as a negative number, all reps occur on that channel.</td></tr>
<tr><td>Threshold
Constant</td><td>The voltage in millivolts that the input must cross for a count to occur. For a signal centred around CR5000 ground (Figure 7.8-1) the threshold should be 0. If the input signal is a 0 to 5 V CMOS signal then a threshold of 2500 mV would result in the voltage comparator switching at 2.5 V.</td></tr>
<tr><td>PAoption</td><td>Specifies whether to output the period in µs or the frequency in Hz.
<table>
<tr><th>Numeric Code</th><th>Voltage Range</th></tr>
<tr><td>0</td><td>Period of the signal is returned</td></tr>
<tr><td>1</td><td>Frequency of the signal is returned</td></tr>
</table></td></tr>
<tr><td>Cycles
Constant</td><td>The number of cycles to be measured for the average calculation.</td></tr>
<tr><td>Timeout
Constant</td><td>The maximum time duration (in msec) that the logger will wait for the number of Cycles to be measured for the average calculation.</td></tr>
<tr><td>Mult, Offset
Constant, Variable, Array, or Expression</td><td>A multiplier and offset by which to scale the raw results of the measurement.</td></tr>
</table>

Low-level signals are amplified prior to a voltage comparator for the period averaging measurement. The internal voltage comparator is referenced to the user entered threshold. The threshold parameter allows a user to reference the internal voltage comparator to voltages other than 0 V. For example, a threshold of 2500 mV allows a 0 to 5 V digital signal to be sensed by the internal comparator without the need of any additional input conditioning circuitry. The threshold allows direct connection of standard digital signals, but is not recommended for small amplitude sensor signals. For sensor amplitudes less than 20 mV pk-pk a dc blocking capacitor, see Figure 7.8-1, is recommended to centre the signal at CR5000 ground (threshold = 0) because of offset voltage drift along with limited accuracy (± 10 mV) and resolution (1.2 mV) of a threshold other than 0.

The minimum pulse width requirements increase (maximum frequency decreases) with increasing gain as shown in range parameter. Signals larger than the specified maximum for a range will saturate the gain stages and prevent operation up to the maximum specified frequency. Back-to-back diodes, Figure 7.8-1, are recommended to limit large amplitude signals to within the input signal ranges.

CAUTION

Noisy signals with slow transitions through the voltage threshold have the potential for extra counts around the comparator switch point. A voltage comparator with 20 mV of hysteresis follows the voltage gain stages. The effective input referred hysteresis equals 20 mV divided by the selected voltage gain. The effective input referred hysteresis on the ± 200 mV range is 1 mV. Consequently, 1 mV of noise on the input signal could cause extraneous counts on the ± 200 mV range. **For best results, select the largest input range (smallest gain) that will meet the minimum input signal requirements.**

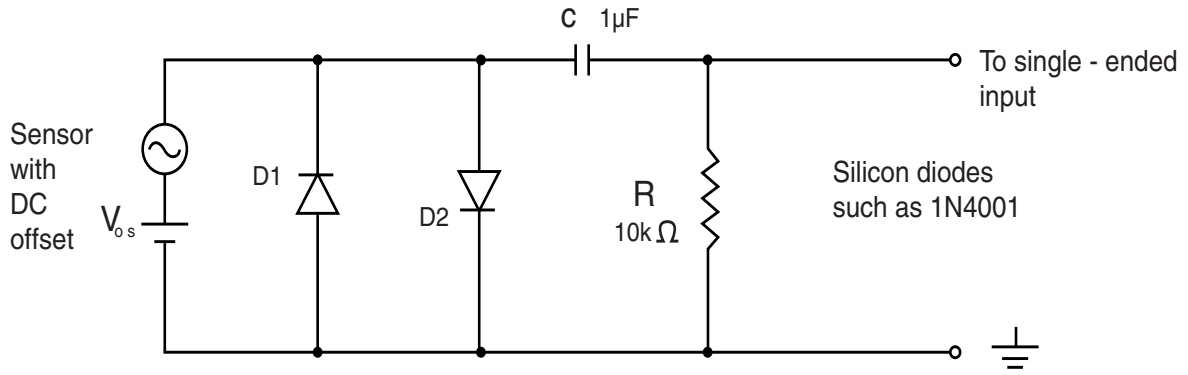


FIGURE 7.8-1. Input conditioning circuit for low-level and high level period averaging.

Figure 7.8-1 shows a circuit that capacitively couples an input signal to centre it around ground and also limits the amplitude of the input to allowable levels. The capacitor C is a dc blocking capacitor for offset voltage removal. Resistor R1 is used to bias the datalogger side of the input circuit to ground. The reactance of the dc blocking capacitor ($Xc = (2 \cdot \pi \cdot f \cdot C)^{-1}$) and resistor R1 form a voltage divider at low frequencies ($R1/(R1 + Xc)$) that attenuates the applied input signal. This attenuation sets a lower limit on low-frequency operation and the minimum size of R1. The circuit attenuates the input signal by a factor of 2 at 16 Hz.

The back-to-back silicon diodes D1 and D2 provide ESD protection of capacitor C and the sensor, and also limit the amplitude of large amplitude sensor signals. These diodes clip large amplitude signals to approximately 1.4 V pk-pk which is within the recommended input signal ranges for all range codes. Diodes D1 and D2 along with resistor R1 are recommended to limit large amplitude sensor signals, even when dc blocking capacitor C is not used. Sensors outputting large voltages may cause large currents to flow through these back-to-back diodes. A current limiting resistor may be desirable to minimize these currents in some situations.

The current flow through these clipping diodes may also induce single-ended offset voltages if it returns into the ⏚ ground terminals. Single-ended offset voltages of up to 2 µV/mA of current that flows into the ⏚ ground terminals can

be induced across the front panel. The back-to-back diodes can be tied into the G ground terminals, rather than ⏚ ground terminals, if this is a problem.

Period Average Example

The following program instructions demonstrate the use of the PeriodAvg function to measure a water content reflectometer that outputs a square wave that can be measured by the datalogger. The measured period is converted to volumetric water content using a polynomial.

```
Dim Time1(9)                    'array for the realtime instruction
Public H20period                'declaration
Const H20percent=0              'set value
Public H20percent               'declaration

BeginProg

Scan (5,Sec,3,0)                'set 5 second scan rate
    If Time5=0 Then             'if the time is the top of the hour Then DO
    Portset (1 ,1 )             'Turn on Sensor by setting port 1 high
    PeriodAvg(H20period,1,0,1,0,10,0.05,.001,0)
    Portset (1 ,0)              'Turn off sensor by setting port 1 low
    'Run through a polynomial to calculate percent
    H20percent=(-0.187)+(0.037*H20period)+((0.335*H20period)^2)
    EndIf Time5<>0              'if the time is not the top of the hour, End
    NextScan
    EndProg
```

PortGet (Dest, Port)

The PortGet function is used to read the status of one of the eight control ports.

Remarks
This instruction will read the status of the specified port and place the result in the Dest variable. This instruction is controlled by the task sequencer, which sets up the measurement order. This results in the PortGet instruction always occurring directly after the measurement instruction preceding it in the program, regardless if the PortGet instruction is in a conditional block. If it is desired to read the status of a port conditionally, the ReadIO instruction should be utilized.

Parameter *& Data Type*	**Enter**
Dest *Variable*	The variable in which to store the result of the instruction. A 1 is stored if the port is high; 0 is stored if the port is low.
Port *Constant*	The control port number (1-8) for which the status should be obtained.

PortGet Example

This example uses PortGet to read the status of port 1 at the beginning of each scan.

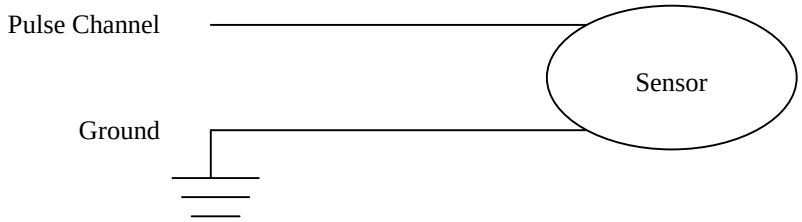
```
Const RATE = 500                    'Scan interval number
Const RUNITS = 1                    'Scan interval units (mSecs)
Const TBLINT1 = 200                 'Table 1 interval number
Const UNITS1 = 1                    'Table 1 interval units (mSecs)
Dim Tblk1( 1 ),PortStatus           'Block #1 dimensioned source
Dim Tref                            'Declare reference temperature variable
Public Flag( 8 )                    'Flag dimension

DataTable( TEST, Flag( 1 ), 1000 )  'Trigger, buffer of 1000 Records
    DataInterval( 0, TBLINT1, UNITS1, 100 )    '200 mSecs, 100 lapses
    Average( 1, Tblk1(), FP2, 0 )   'Reps,Source,Res,Disable
    Sample ( 1, PortStatus, FP2 )   'Reps,Source,Res,Disable
EndTable                            'End of table TEST

BeginProg                           'Program begins here

Scan( RATE, RUNITS, 3, 0 )          'Scan 500 (mSecs)
    PortGet( PortStatus, 1)         'Set port 1 high
    PanelTemp( Tref. Integ )        'PanelTemp
    TCDiff( Tblk1(), 1, 1, 1, 1, Tref, 0, 20, 40, 1.8, 32 )
    CallTable TEST                  'Run Table TEST
    Next Scan                       'Loop up for the next scan
EndProg                             'Program ends here
```

PortSet (Port, State)

This Instruction will set the specified port high or low.

Parameter *& Data Type*	**Enter**
Port *Constant, Variable, or Expression*	The number of the port (1-8) to set with the instruction.
State *Constant, Variable, or Expression*	The state (high or low) to set the port to. **Code** / **Value** / **State** True / 0 / Low False / ≠ 0 / High

PulseCount (Dest, Reps, PChan, PConfig, POption, Mult, Offset)



<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>Dest
Variable or Array</td><td>The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.</td></tr>
<tr><td>Reps
Constant</td><td>The number of repetitions for the measurement or instruction.</td></tr>
<tr><td>PChan
Constant</td><td>The number of the pulse channel (1 or 2) for the measurement.</td></tr>
<tr><td>PConfig
Constant</td><td>A code specifying the type of pulse input to measure.
<table>
<tr><th>Code</th><th>Input Configuration</th></tr>
<tr><td>0</td><td>High Frequency</td></tr>
<tr><td>1</td><td>Low Level AC</td></tr>
<tr><td>2</td><td>Switch Closure</td></tr>
</table></td></tr>
<tr><td>POption
Constant</td><td>A code that determines if the raw result (multiplier = 1, offset = 0) is returned as counts or frequency. The running average can be used to smooth out readings when a low frequency relative to the scan rate causes large fluctuations in the measured frequency from scan to another.
<table>
<tr><th>Code</th><th>Result</th></tr>
<tr><td>0</td><td>Counts</td></tr>
<tr><td>1</td><td>Frequency (Hz) counts/scan interval in seconds</td></tr>
<tr><td>>1</td><td>Running average of frequency. The number entered is the time period over which the frequency is averaged in milliseconds.</td></tr>
</table></td></tr>
<tr><td>Mult, Offset
Constant, Variable, Array, or Expression</td><td>A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the TCDiff instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.</td></tr>
</table>

The PulseCount instruction is used to measure counts or frequency on one of the pulse channels.

NOTE The PulseCount instruction can not be used in a Slow Sequence scan.

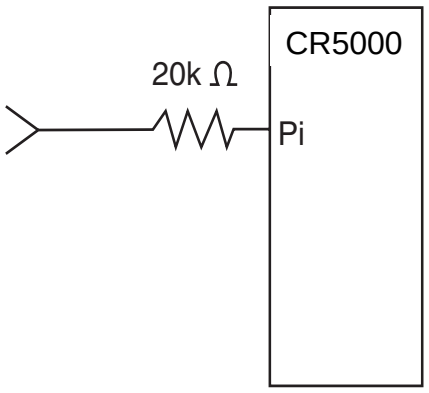


FIGURE 7.8-1. Conditioning Large Voltage Pulses

The maximum input voltage on a pulse channel is ±20 V. Refer to Figure 7.8-1 if reducing input voltage is required.

- Pulse Channels
  Maximum Input Voltage: ±20 V

High Frequency Input

- Pulse Channels
  Minimum Pulse Width: 1.2 microsecond
  Maximum Frequency: 400 kHz
  (50% Duty Cycle)
  Lower Threshold: 1.5 V*
  Upper Threshold: 3.5 V*

When a pulse channel is configured for high-frequency pulse, there is an internal 100 kohm pull-up resistor to 5 V on the pulse channel. This pull-up resistor accommodates open-collector output devices for high-frequency input.

*Larger input transitions are required at high frequencies because of the input RC filter with 1.2 microsecond time constant. Signals up to 400 kHz will be counted if centred around +2.5 V with deviations ≥ ± 2.5 V for ≥ 1.2 microseconds.

Low Level AC (Pulse Channels Only)

Input Hysteresis: 15 mV
Maximum Input Voltage: 20 V peak-to-peak
Input Voltage and Frequency Range
(16 bit counter required above 2.56 kHz)

20 mV	1.0 Hz to 1 kHz
200 mV	0.5 Hz to 10 kHz
1000 mV	0.3 Hz to 16 kHz

Switch Closure

- Pulse Channels
  A switch closure is connected between P1..P2 and analogue ground. When the switch is open, the CR5000 pulls the pulse channel to 5 V through a 100 kOhm impedance. When the switch is closed, the pulse channel is pulled to ground. The count is incremented when the switch opens.

  Minimum Switch Closed Time: 5 ms
  Minimum Switch Open Time: 6 ms
  Maximum Bounce Time: 1 ms open without being counted

PulseCountReset

PulseCountReset is used to reset the pulse counters and the running averages used in the pulse count instruction. The 16 bit counters can count up to decimal 65535. More counts than 65535 result in an over-range condition. With each scan, the CR5000 reads the counts accumulated since the last scan and then resets the counter. If the scans stop, as in a program with more that one Scan loop, the counter continues to accumulate counts until another scan is initiated or it over-ranges. If the running averaging is in use, the over-range value will be included in the average until for the duration of the averaging period (e.g., with a 1000 millisecond running average, the over-range will be the value from the **PulseCount(...)** instruction until 1 second has passed. Resetting the average prior to (re)starting the scan avoids this.

ReadIO (Dest, Mask)

ReadIO is used to read the status of selected control I/O channels (ports) on the CR5000. There are 8 ports. The status of these ports is considered to be a binary number with a high port (+5 V) signifying 1 and a low port (0 V) signifying 0. For example, if ports 1 and 3 are high and the rest low, the binary representation is 00000101, or 5 decimal. The mask parameter is used to select which of the ports to read, it too is a binary representation of the ports, a 1 means pay attention to the status of the port, a 0 means ignore the status of the port (the mask is "anded" with the port status; the "and" operation returns a 1 for a digit if the mask digit and the port status are both 1 and a 0 if either or both is 0). CRBasic allows the entry of numbers in binary format by preceding the number with "&B". For example if the mask is entered as &B100 (leading zeros can be omitted in binary format just as in decimal) and ports 3 and 1 are high as in the previous example, the result of the instruction will be 4 (decimal, binary = 100); if port 3 is low, the result would be 0.

Examples

```
ReadIO(Port3, &B100) ' read port 3 if port 3 is high then Port3 =
        4, if port 3 is low then Port3 = 0
```

SW12

The SW12 instruction is used to set a Switched 12 volt supply high or low.

Syntax
SW12(State)

Remarks
The datalogger has a switched 12 volt output with two terminals. This switched 12 volts is used to provide a continuous 12 volt supply to external peripherals. At room temperature the switched 12 volt supply can source 900 mA between the SW-12 terminal and Ground. The State parameter indicates whether the switched 12 volts is High (non-zero) or low (0).

NOTE The SW-12 supply is unregulated and can supply up to 900 mA at 20C and up to 630 mA at 50C. A resettable polymeric fuse protects against over-current. Reset is accomplished by removing the load or turning off the SW-12 for several seconds.

SW12 Example
This example sets the switched 12 volt supply high at the beginning of each scan, and low at the end of the program.

```
Const RATE = 500                        'Scan interval number
Const RUNITS = 1                        'Scan interval units (mSecs)
Const TBLINT1 = 200                     'Table 1 interval number
Const UNITS1 = 1                        'Table 1 interval units (mSecs)
Dim Tblk1( 1 )                          'Block #1 dimensioned source
Dim Tref                                'Declare reference temperature variable
Public Flag( 8 )                        'Flag dimension

DataTable( TEST, Flag( 1 ), 1000 )      'Trigger, buffer of 1000 Records
DataInterval( 0, TBLINT1, UNITS1, 100 )  '200 mSecs, 100 lapses
    Average( 1, Tblk1(), FP2, 0 )       'Reps,Source,Res,Disable
EndTable                                'End of table TEST

BeginProg                               'Program begins here
Scan( RATE, RUNITS, 1, 0 )              'Scan 500 (mSecs), no fast buffer
SW12( 1 )                               'Set SW12 high
    PanelTemp( Tref., Integ )           'PanelTemp
    TCDiff( Tblk1(), 1, 1, 1, 1, Tref, 0, 20, 40, 1.8, 32 )
    CallTable TEST                      'Run Table TEST
SW12 ( 0 )                              'Set SW12 log
Next Scan                               'Loop up for the next scan
EndProg                                 'Program ends here
```

TimerIO

The TimerIO instruction is used to measure the time between edges (state transitions) or frequency on the digital I/O ports of the datalogger. The timing resolution is 50 nanoseconds. The longest interval that can be timed is 214.7 seconds (2^{32} x 50 nanoseconds).

Syntax
TimerIO(Dest, Edges, Function, Timeout, Units)

Remarks
There are eight control ports on the datalogger. Six of the ports (1 through 6) can be used for edge timing and low frequency period/frequency measurements (<1 kHz). The other two ports (7 and 8) can be used to measure high frequency signals (up to about 5 MHz) with edge counting only (no edge timing). When port 7 is enabled, ports 3 and 4 are not available for edge timing. When port 8 is enabled, ports 5 and 6 cannot be used for edge timing.

The TimerIO instruction has the following parameters:

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td>Dest</td><td>The Dest parameter is a variable array in which to store the results of the measurement. The array must be dimensioned equal to the number of ports for which results are requested.</td></tr>
<tr><td>Edges</td><td>The Edge parameter consists of 8 digits used to configure each of the 8 ports. The digits represent the ports in descending order from left to right. If a 1 is entered for a port, the transition will be counted on the rising edge (from <1.5V to >3.5V). If a 0 is entered, the transition will be counted on the falling edge (from >3.5V to <1.5V).</td></tr>
<tr><td>Function</td><td>The Function parameter has 8 digits. Each digit is used to program the results for each port. Only one function may be programmed per port. The number of values returned is determined by the number of ports for which a result is requested.
<table>
<tr><th>Digit</th><th>Results</th></tr>
<tr><td>0</td><td>The associated port is not used.</td></tr>
<tr><td>1</td><td>Calculate the period of the signal on the specified port (in microseconds).</td></tr>
<tr><td>2</td><td>Calculate the frequency (Hz) for the specified port.</td></tr>
<tr><td>3</td><td>Calculate the time from an edge on the previous port (1 number lower) to an edge on the specified port (in microseconds).</td></tr>
<tr><td>4</td><td>Calculate the time from an edge on port 1 to an edge on the specified port (in microseconds).</td></tr>
<tr><td>5</td><td>Not used.</td></tr>
<tr><td>6</td><td>Not used.</td></tr>
<tr><td>7</td><td>Pulse counting period on port 7 or 8 (in microseconds).</td></tr>
<tr><td>8</td><td>Pulse counting frequency on port 7 or 8 (in Hz).</td></tr>
</table></td></tr>
<tr><td>Timeout</td><td>A constant is entered for the Timeout parameter to determine the results to be returned when no changes have occurred in the specified ports since the last execution of the instruction. If the timeout period has expired and no change has occurred, a null value will be returned (0 for frequency and NaN for period or time since last edge). If the timeout period has not expired the last valid result will be returned. This is useful for measuring signals with periods greater than the scan interval. The maximum time out is 2^{32} x the scan.</td></tr>
<tr><td>Units</td><td>The Units parameter is used to specify the unit of measure for the timeout period. An alphabetical or numeric code can be entered.
<table>
<tr><th>Numeric</th><th>Alpha</th><th>Description</th></tr>
<tr><td>0</td><td>usec</td><td>microseconds</td></tr>
<tr><td>1</td><td>msec</td><td>milliseconds</td></tr>
<tr><td>2</td><td>sec</td><td>seconds</td></tr>
<tr><td>3</td><td>min</td><td>minutes</td></tr>
</table></td></tr>
</table>

WriteIO (Mask, Source)

WriteIO is used to set the status of selected control I/O channels (ports) on the CR5000. (See Also PortSet.) There are 8 ports. The status of these ports is considered to be a binary number with a high port (+5 V) signifying 1 and a low port (0 V) signifying 0. For example, if ports 1 and 3 are high and the rest low, the binary representation is 00000101, or 5 decimal. The source value is interpreted as a binary number and the ports set accordingly. The mask parameter is used to select which of the ports to set, it too is a binary representation of the ports, a 1 means set the port according to the source, a 0 means do not change the status of the port. CRBasic allows the entry of numbers in binary format by preceding the number with "&B". For example if the mask is entered as &B110 (leading zeros can be omitted in binary format just as in decimal) and the source is 5 decimal (binary 101) port 3 will be set high and port 2 will be set low. The

mask indicates that only 3 and 2 should be set. While the value of the source also has a 1 for port 1, it is ignored because the mask indicates 1 should not be changed.

Example

```
WriteIO (&B100, &B100)    ' Set port 3 high.
```

Parameter *& Data Type*	**Enter**
Mask *Constant*	The Mask allows the read or write to only act on certain ports. The Mask is ANDed with the value obtained when reading and ANDed with the source before writing.
Source *Constant Variable*	The Variable or number that is to be written to the I/O ports.

7.9 Peripheral Devices

AM25T (Dest, Reps, Range, AM25TChan, DiffChan, TCType, Tref, ClkPort, ResPort, VxChan, RevDiff, SettlingTime, Integ, Mult, Offset)

This Instruction controls the AM25T Multiplexer.

<table>
<tr><td>Parameter
& Data Type</td><td colspan="4">Enter</td></tr>
<tr><td>Dest Variable or Array</td><td colspan="4">The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.</td></tr>
<tr><td>Reps</td><td colspan="4">The number of channels to measure on the AM25T. Enter 0 to just measure Temperature.</td></tr>
<tr><td>Range
Constant</td><td>Alpha Code</td><td>Numeric Code</td><td>Voltage Range</td><td></td></tr>
<tr><td></td><td>mV5000</td><td>0</td><td>± 5000 mV</td><td></td></tr>
<tr><td></td><td>mV1000</td><td>1</td><td>± 1000 mV</td><td></td></tr>
<tr><td></td><td>mV200</td><td>2</td><td>± 200 mV</td><td></td></tr>
<tr><td></td><td>mV50</td><td>3</td><td>± 50 mV</td><td></td></tr>
<tr><td></td><td>mV20</td><td>4</td><td>± 20 mV</td><td></td></tr>
<tr><td></td><td>AutoRange</td><td>5</td><td>mV20 - mV5000</td><td>Selects range (Sect. 3.1)</td></tr>
<tr><td></td><td>mV200C</td><td>20</td><td>± 200 mV</td><td rowspan="3">The mV200C, mV50C, and mV20C ranges pull the channel into common mode range and check for open input</td></tr>
<tr><td></td><td>mV50C</td><td>30</td><td>± 50 mV</td></tr>
<tr><td></td><td>mV20C</td><td>40</td><td>± 20 mV</td></tr>
<tr><td>Am25tChan
Constant</td><td colspan="4">The starting input channel on the multiplexer.</td></tr>
<tr><td>DiffChan
Constant</td><td colspan="4">The Differential channel that will be used to make the actual measurements from the AM25T. If the channel is entered as a negative number, all reps occur on that channel.</td></tr>
<tr><td>TCType
Constant</td><td colspan="4">The Thermocouple Type Code. Enter -1 to return voltage measurements.</td></tr>
<tr><td>Tref
Variable</td><td colspan="4">The variable in which to store/read the AM25T reference temperature.</td></tr>
<tr><td>ClkPort
Constant</td><td colspan="4">The Digital Output port number that will be used to clock the AM25T. One clock port may be used with several AM25Ts.</td></tr>
<tr><td>ResPort
Constant</td><td colspan="4">The Digital Output port number that will be used to enable and reset the AM25T. Each AM25T must have it's own unique Reset line</td></tr>
<tr><td>VxChan
Constant</td><td colspan="4">The Excitation Channel number that will be used to provide excitation for the PRT reference temperature measurement. If 0 is entered for the excitation channel, the temperature is not measured.</td></tr>
</table>

<table>
<tr><td>RevDiff
Constant</td><td>
<table>
<tr><th>Code</th><th>Value</th><th>Result (Reversing requires twice as much time to complete)</th></tr>
<tr><td>False</td><td>0</td><td>Signal is measured with the high side referenced to the low</td></tr>
<tr><td>True</td><td>≠0</td><td>A second measurement is made after reversing the inputs to cancel offsets</td></tr>
</table>
</td></tr>
<tr><td>SettlingTime
Constant</td><td>The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement. (1 microsecond resolution)
<table>
<tr><th>Entry</th><th>Voltage Range</th><th>Integration</th><th>Settling Time</th></tr>
<tr><td>0</td><td>± 20 mV</td><td>Flash</td><td>200 µS (default)</td></tr>
<tr><td>0</td><td>All but ± 20 mV,</td><td>Flash</td><td>100 µS (default)</td></tr>
<tr><td>0</td><td>All</td><td>250 µS</td><td>200 µS (default)</td></tr>
<tr><td>0</td><td>All</td><td>_50Hz, _60 Hz</td><td>3 mS (default)</td></tr>
<tr><td>>=100</td><td>All</td><td>All</td><td>µS entered</td></tr>
</table>
</td></tr>
<tr><td>Integ
Constant</td><td>The time spent on integration in microseconds for each of the channels measured.
<table>
<tr><th>Entry</th><th>Integration</th></tr>
<tr><td>0, 200</td><td>Flash Conversion; 200 = 2 flash conversions 100 µS apart and averaged</td></tr>
<tr><td>250; 500-16000</td><td>250 µS; multiples of 500: 250 µS integrations starting each 500 µS and averaged</td></tr>
<tr><td>_60Hz or 16667</td><td>16,667 µS (reject 60 Hz noise)</td></tr>
<tr><td>_50 Hz or 20000</td><td>20,000 µS (reject 50 Hz noise)</td></tr>
</table>
</td></tr>
<tr><td>Mult, Offset
Constant, Variable, Array, or Expression</td><td>A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the TCDiff instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.</td></tr>
</table>

AO4 (Source, Reps, SDMAdress)

This instruction is used to set the voltage to an SDM-AO4 four channel analogue output device.

The AO4 instruction has the following parameters:

Parameter *& Data Type*	**Enter**
Source	The Source parameter is the variable that hold the voltage, in millivolts, that should be sent to the SDM-AO4. If multiple SDM-AO4s are to be triggered with one instruction, this parameter must be an array dimensioned to the size of the Reps.
Reps	The Reps parameter determines the number of SDM-AO4 devices to supply a voltage using this instruction. The SDM-A04s must have sequential SDM addresses if the Reps parameter is greater than 1.
SDMAddress	The SDMAddress parameter defines the address of the SDM-AO4 to which a voltage should be applied. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.

CANBUS (Dest, SDMAddress, TimeQuanta, TSEG1, TSEG2, ID, DataType, StartBit, NumBits, NumVals, Multiplier, Offset)

The CANBUS instruction is used to measure and control the SDM-CAN interface.

Multiple CANBUS instructions may be used within a program. The initial function of the instruction is to configure the SDM-CAN interface when the datalogger program is compiled. Subsequent instructions can be used to determine what data is passed between the CAN-bus network and the datalogger, set and/or read the SDM-CAN's internal switches, and read and/or reset detected errors.

The SDMTrigger instruction can be used to trigger simultaneous measurements from one or more SDM-CANs and other SDM devices connected to the datalogger. When the SDMTrigger instruction is encountered in a program, it broadcasts a special SDM message which causes all the SDM-CAN devices to copy the last data values captured from the CAN-bus into the working data buffers. Refer to the SDM-CAN manual for additional help.

The CANBUS instruction has the following parameters:

Parameter *& Data Type*	**Enter**
Dest	The Dest parameter is a variable array in which to store the results of the measurement. It must be an array of sufficient size to hold all of the values that will be returned by the function chosen (defined by the DataType parameter).
SDMAddress	The SDMAddress parameter defines the address of the SDM-CAN with which to communicate. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.
TimeQuanta	Three time segments are used to set the bit rate and other timing parameters for the CAN-bus network, TimeQuanta, TSEG1, and TSEG2. These parameters are entered as integer numbers. The relationship between the three time segments is defined as: $t_{bit} = t_q + t_{TSEG1} + t_{TSEG2}$ The first time segment, the synchronization segment (S-SG), is defined by the TimeQuanta parameter. To calculate a suitable value for TiimeQuanta, use the following equation: $\text{TimeQuanta} = t_q * 8 * 10^6$ where tq = the TimeQuanta. There are between 8 and 25 time quanta in the bit time. The bit time is defined as 1/baud rate.
TSEG1	The second time segment, TSEG1, is actually two time segments known as the propagation segment and phase segment one. The value entered is determined by the characteristics of the network and the other devices on the network. It can be calculated as: $T_{SEG1} = t_{TSEG1} / t_q$

Parameter *& Data Type*	**Enter**
TSEG2	The third time segment, TSEG2 (the phase segment two), is defined by the TSEG2 parameter. The value of TSEG2 can be calculated using the equation: $T_{SEG2} = t_{TSEG2} / t_q$ The relative values of TSEG1 and TSEG2 determine when the SDM-CAN samples the data bit.
ID	Each device on a CAN-bus network prefaces its data frames with an 11 or 29 bit identifier. The ID parameter is used to set this address. The ID is entered as a single decimal equivalent. Enter a positive value to signify a 29 bit ID or a negative value to signify an 11 bit ID.
DataType	The DataType parameter defines what function the CANBUS instruction will perform. This instruction can be used to collect data, buffer data for transmission to the CAN-bus, transmit data to the CAN-bus, read or reset error counters, read the status of the SDM-CAN, read the SDM-CAN's OS signature and version, send a remote frame, or read or set the SDM-CAN's internal switches. Enter the numeric value for the desired option:

Value	**Description**
1	Retrieve data; unsigned integer, most significant byte first
2	Retrieve data; unsigned integer, least significant byte first
3	Retrieve data; signed integer, most significant byte first
4	Retrieve data; signed integer, least significant byte first
5	Retrieve data; 4-byte IEEE floating point number; most significant byte first
6	Retrieve data; 4-byte IEEE floating point number; least significant byte first
7	Build data frame in SDM-CAN memory; unsigned integer, most significant byte first. Overwrite existing data.
8	Build data frame in SDM-CAN memory; unsigned integer, least significant byte first. Overwrite existing data.
9	Build data frame in SDM-CAN memory; signed integer, most significant byte first. Overwrite existing data.
10	Build data frame in SDM-CAN memory; signed integer, least significant byte first. Overwrite existing data.
11	Build data frame in SDM-CAN memory; 4-byte IEEE floating point number; most significant byte first. Overwrite existing data.
12	Build data frame in SDM-CAN memory; 4-byte IEEE floating point number; least significant byte first. Overwrite existing data.
13	Build data frame in SDM-CAN memory; unsigned integer, most significant byte first. Logical "OR" with existing data.
14	Build data frame in SDM-CAN memory; unsigned integer, least significant byte first. Logical "OR" with existing data.
15	Build data frame in SDM-CAN memory; signed integer, most significant byte first. Logical "OR" with existing data.
16	Build data frame in SDM-CAN memory; signed integer, least significant byte first. Logical "OR" with existing data.
17	Build data frame in SDM-CAN memory; 4-byte IEEE floating point number; most significant byte first. Logical "OR" with existing data.

Parameter *& Data Type*	**Enter**
	18 Build data frame in SDM-CAN memory; 4-byte IEEE floating point number; least significant byte first. Logical "OR" with existing data. 19 Transmit data value to the CAN-bus; unsigned integer, most significant byte first. 20 Transmit data value to the CAN-bus; unsigned integer, least significant byte first. 21 Transmit data value to the CAN-bus; signed integer, most significant byte first.
	22 Transmit data value to the CAN-bus; signed integer, least significant byte first. 23 Transmit data value to the CAN-bus; 4-byte IEEE floating point number; most significant byte first. 24 Transmit data value to the CAN-bus; 4-byte IEEE floating point number; least significant byte first. 25 Transmit previously built data frame to the CAN-bus. 26 Set up previously built data frame as a Remote Frame Response. 27 Read Transmit, Receive, Overrun, and Watchdog errors. The errors are placed consecutively in the array specified by the Dest parameter. 28 Read Transmit, Receive, Overrun, and Watchdog errors. The errors are placed consecutively in the array specified by the Dest parameter. Reset error counters to 0 after reading. 29 Read SDM-CAN status; result is placed into the array specified in the Dest parameter. The result codes are as follows: **Status** **Description** 0000 The SDM-CAN is involved in bus activities; error counters are less than 96. 0001 The SDM-CAN is involved in bus activities; one or more error counters is greater than or equal to 96. 0002 The SDM-CAN is not involved in bus activities; error counters are less than 96. 0003 The SDM-CAN is not involved in bus activities; one or more error counters is greater than or equal to 96. 30 Read SDM-CAN operating system and version number; results are placed in two consecutive array variables beginning with the variable specified in the Dest parameter. 31 Send Remote Frame Request. 32 Set SDM-CAN's internal switches. The code is stored in the array specified in the Dest parameter and is entered in the form of ABCD. **Switch** **Code** **Description** A 0 Currently not used; set to 0. B 0 SDM-CAN returns the last value captured from the network, even if that value has been read before (default).

Parameter *& Data Type*	**Enter**		
		1	SDM-CAN returns -99999 if a data value is requested by the datalogger and a new value has not been captured from the network since the last request.
		2-9	Currently not used.
	C	0	Disable I/O interrupts (default).
		1	Enable I/O interrupts, pulsed mode.
		2	Enable I/O interrupts, fast mode.
		3-7	Currently not used.
		8	Place the SDM-CAN into low power stand-by mode
		9	Leave switch setting unchanged.
	D	0	Listen only (error passive) mode. CAN transmissions are not confirmed.
		1	Transmit once. Data will not be retransmitted in case of error or loss of arbitration. Frames received without error are acknowledged.
		2	Self-reception. A frame transmitted from the SDM-CAN that was acknowledged by an external node will also be received by the SDM-CAN but no retransmission will occur in the event of loss of arbitration or error. Frames received correctly from an external node are acknowledged.
		3	Normal, retransmission will occur in the event of loss of arbitration or error. Frames received correctly from an external node are acknowledged. This is the typical setting to use if the SDM-CAN is to be used to transmit data.
		4	Transmit once; self-test. The SDM-CAN will perform a successful transmission even if there is no acknowledgment from an external CAN node. Frames received correctly from an external node are acknowledged.
		5	Self-reception; self -test. The SDM-CAN will perform a successful transmission even if there is no acknowledgment from an external CAN node. Frames received correctly from an external node are acknowledged. SDM-CAN will receive its own transmission.

Parameter *& Data Type*	**Enter**
	6 Normal; self-test. The SDM-CAN will perform a successful transmission even if there is no acknowledgment from an external CAN node. Frames received correctly from an external node are acknowledged. 7 Not defined. 8 Not defined. 9 Leave switch setting unchanged. 33 Read SDM-CAN's internal switches. Place results in the array specified in the Dest parameter.
StartBit	The StartBit parameter is used to identify the least significant bit of the data value within the CAN data frame to which the instruction relates. The bit number can range from 1 to 64 (there are 64 bits in a CAN data frame). The SDM-CAN adheres to the ISO standard where the least significant bit is referenced to the right most bit of the data frame. If a negative value is entered, the least significant bit is referenced to the left most bit of the data frame.
NumBits	The NumBits parameter is used to specify the number of bits that will be used in a transaction. The number can range from 1 to 64 (there are 64 bits in a CAN data frame). The SDM-CAN can be configured to notify the datalogger when new data is available by setting a control port high. This allows data to be stored in the datalogger tables faster than the program execution interval. This interrupt function is enabled by entering a negative value for this parameter. **Note: This parameter may be overridden by a fixed number of bits, depending upon the data type selected.**
NumVals	The NumVals parameter defines the number of values (beginning with the value stored in the Dest array) that will be transferred to or from the datalogger during one operation. For each value transferred, the Number of Bits (NumBits) will be added to the Start Bit number so that multiple values can be read from or stored to one data frame.
Mult, Offset	The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement.

NOTE If more than one Instruction 118 is used within a datalogger program, the values used for TimeQuanta, TSEG1 and TSEG2 must be the same for each instruction.

CANBUS Example
The following example reads a 16-bit engine speed value from a CAN-bus network running at 250K baud.

```
'Set Scan Rate
Const Period=1
Const P_Units=2
' \\\\\\\\\\\\\ CANBUS Constants / / / / / / / / / / / /
'------------------------ Physical Network Parameters ------------
'Set SDM-CAN to 250K
Const TQUANT=4
Const TSEG1=5
Const TSEB2=2

'----------------------- Data Frame Parameters -------------------
'_____________________ Canbus Block1_____________________
'Collect and retrieve 16-bit data value
'Data Type 1, unsigned integer, most significant byte first
Const CANREP1=1                 'Repetitions
Const ADDR1=0                   'Address of SDM-CAN module
Const DTYPE1=1                  'Data values to collect
Const STBIT1=33                 'Start position in data frame
Const NBITS1=16                 'Number of bits per value
Const NVALS1=1                  'Number of values
Const CMULT1=0.4                'Multiplier
Const COSET1=0                  'Offset
Dim CANBlk1(CANREP1)            'Dimensioned Dest
' \\\\\\\\\\\\\ Aliases and other Variables / / / / / / / /
Alias Canblk1(1)=Engine_Speed
' \\\\\\\\\\\\\\\ PROGRAM / / / / / / / / / / / / / / / / /
BeginProg
      Scan(PERIOD,P_UNITS,0,0)
      '_____________________ CAN Blocks_____________________
      'Retrieve Data from CAN-bus network
      Canbus (CANBLK1(), ADDR1, TQUANT, TSEG1, TSEG2, 217056256,
DTYPE1, STBIT1, NBITS1, NVALS2, CMJLT1, COSET1)
      Next Scan
EndProg
```

CD16AC (Source, Reps, SDMAddress)

The CD16AC instruction is used to control an SDM-CD16AC, SDM-CD16, or SDM-CD16D 16 channel relay/control port device.

A port on an SDM-CD16xx is enabled/disabled (turned on or off) by sending a value to it using the CD16AC instruction. A non-zero value will turn the port on; a zero value will turn it off. The values to be sent to the CD16AC are held in the Source array.

Parameter *& Data Type*	**Enter**
Source *Array*	The array which holds the values that will be sent to the SDM-CD16AC to enable/disable its ports. An SDM-CD16AC has 16 ports; therefore, the source array must be dimensioned to 16 times the number of Repetitions (the number of SDM-CD16AC devices to be controlled). As an example, with the array CDCtrl(32), the value held in CDCtrl(1) will be sent to port 1, the value held in CDCtrl(2) will be sent to port 2, etc. The value held in CDCtrl(32) would be sent to port 16 on the second SDM-CD16AC..
Reps *Constant*	The Reps parameter is the number of SDM-CD16AC devices to be controlled with this instruction.
SDMAddress *Constant*	The address of the first CD16AC that will be controlled with this instruction. Valid SDM addresses are 0 through 15. If the SDMTrigger instruction is used in the program, address 15 should not be used. If the Reps parameter is greater than 1, the datalogger will increment the SDM address for each subsequent device that it communicates with.

CS7500 (Dest, Reps, SDMAddress, CS7500Cmd)

Communicates with the CS7500 open path CO_2 and H_2O sensor. See CS7500 manual for more information.

Parameter *& Data Type*	**Enter**
Dest	The Dest parameter is the input variable name in which to store the data from each CS7500 associated with this instruction. The length of the input variable array will depend on the number of Repetitions and on the selected Command.

Command	**Input Variable Length per CS7500**
0 and 1	2
2	4
3	3
4	11
5	3

Reps	The Reps parameter determines the number of CS7500 gas analyzers with which to communicate using this instruction. The CS7500s must have sequential SDM addresses if the Reps parameter is greater than 1
SDMAddress	The SDMAddress parameter defines the address of the CS7500 with which to communicate. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. If the Reps parameter is greater than 1, the datalogger will increment the SDM address for each subsequent CS7500 that it communicates with. The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.

Parameter *& Data Type*	Enter
CS7500Cmd	The CS7500Cmd parameter requests the data to be retrieved from the sensor. The command is sent first to the device specified by the SDMAddress parameter. If the Reps parameter is greater than 1, subsequent CS7500s will be issued the command with each rep. The results for the command will be returned in the array specified by the Dest parameter. A numeric code is entered to request the data:

Code	Description
0	Get CO2 & H2O molar density (mmol/m3)
1	Get CO2 & H2O absorptance
2	Get internal pressure estimate (kPa), auxiliary measurement A, auxiliary measurement B, and cooler voltage (V)
3	Get cell diagnostic value, output bandwidth (Hz), and programmed delay [230 + (delay * 6.579)] (msec)
4	Get all data (CO2 molar density (mmol/m3), H2O molar density (mmol/m3), CO2 absorptance, H2O absorptance, internal pressure estimate (kPa), auxiliary measurement A, auxiliary measurement B, cooler voltage (V), cell diagnostic value, output bandwidth (Hz), and programmed delay [230 + (delay * 6.579)] (msec))
5	Get CO2 & H2O molar density (mmol/m3) and internal pressure estimate (kPa)

CSAT3 (Dest, Reps, SDMAddress, CSAT3Cmd, CSAT3Opt)

Communicates with the CSAT3 three dimensional sonic anemometer. See CSAT3 manual for more information.

Parameter *& Data Type*	Enter
Dest	The Dest parameter is a variable in which to store the results of the measurement. This variable must be dimensioned to a length of five to hold the CSAT3 Ux, Uy, Uz, speed of sound, and diagnostic data.
Reps	The Reps parameter is the number of times the measurement should be made. Measurements are made on consecutive channels. If the Reps parameter is greater than 1, the Dest parameter must be a variable array.
SDMAddress	The SDMAddress parameter defines the address of the CSAT3 with which to communicate. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. If the Reps parameter is greater than 1, the datalogger will increment the SDM address for each subsequent CSAT3 that it communicates with. The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td>Command</td><td>Commands 90 - 92 send a measurement trigger to the CSAT3 with the SDM address specified by the SDMAddress argument. The CSAT3 also sends data to the datalogger. Options 97 - 99 get data after a group trigger, SDMTrigger(), from the CSAT3 specified by the SDMAddress parameter without triggering a new CSAT3 measurements. The CSAT() instruction must be preceded by the SDMTrigger() instruction in order to used Options 97 - 99.
<table>
<tr><th>Code</th><th>Description</th></tr>
<tr><td>90</td><td>Trigger and Get wind & speed of sound data</td></tr>
<tr><td>91</td><td>Trigger and Get wind & sonic temperature data</td></tr>
<tr><td>92</td><td>Trigger and Get wind & speed of sound data minus 340 m/s</td></tr>
<tr><td>97</td><td>Get wind & speed of sound data minus 340 m/s after a Group Trigger</td></tr>
<tr><td>98</td><td>Get wind & sonic temperature data after a Group Trigger</td></tr>
<tr><td>99</td><td>Get wind & speed of sound data after a Group Trigger</td></tr>
</table></td></tr>
<tr><td>Rate</td><td>The Rate argument sets the CSAT3’s execution parameter. This parameter tells the CSAT3 which measurement parameters to use and what frequency to expect the measurement trigger from the datalogger. See the table below for a brief description of each of the parameter and the CSAT3 manual for a detailed description.
<table>
<tr><th>Code</th><th>Description</th></tr>
<tr><td>1</td><td>Set Execution Parameter to 1 Hz</td></tr>
<tr><td>2</td><td>Set Execution Parameter to 2 Hz</td></tr>
<tr><td>3</td><td>Set Execution Parameter to 3 Hz</td></tr>
<tr><td>5</td><td>Set Execution Parameter to 5 Hz</td></tr>
<tr><td>6</td><td>Set Execution Parameter to 6 Hz</td></tr>
<tr><td>10</td><td>Set Execution Parameter to 10 Hz</td></tr>
<tr><td>12</td><td>Set Execution Parameter to 12 Hz</td></tr>
<tr><td>20</td><td>Set Execution Parameter to 20 Hz</td></tr>
<tr><td>30</td><td>Set Execution Parameter to 30 Hz</td></tr>
<tr><td>60</td><td>Set Execution Parameter to 60 Hz</td></tr>
<tr><td>61</td><td>Set Execution Parameter to 60 Hz to 10 Hz Oversample Mode</td></tr>
<tr><td>62</td><td>Set Execution Parameter to 60 Hz to 20 Hz Oversample Mode</td></tr>
</table></td></tr>
</table>

INT8 (Dest, Address, Config8_5, Config4_1, Funct8_5, Funct4_1, OutputOpt, CaptureTrig, Mult, Offset)

This Instruction allows the use of the SDM-INT8, 8 Channel Interval Timer, with the CR5000. The INT8 is a (S)ynchronous (D)evice for the (M)easurement of intervals, counts between events, frequencies, periods, and/or time since an event. See the INT8 manual for more information about its capabilities.

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td>Dest
Variable or Array</td><td>The array where the results of the instruction are stored. For all output options except Capture All Events, the Dest argument should be a one dimensional array with as many elements as there are programmed INT8 channels. If the "Capture All Events" OutputOption is selected, then the Dest array must be two dimensional. The magnitude of first dimension should be set to the number of functions (up to 8), and the magnitude of the second dimension should be set to at least the number of events to be captured. The values will be loaded into the array in the sequence of all of the time ordered events captured from the lowest programmed channel to the time ordered events of the highest programmed channel.</td></tr>
<tr><td>Address
Constant</td><td>The address is entered as a base 10 number. Valid addresses are 0 to 15. The INT8 is addressable using internal jumpers. The jumpers are set at the factory for address 00. See Appendix A of the INT8 manual for details on changing the INT8 address.</td></tr>
<tr><td>Config8_5
Config4_1
Constants</td><td>Each of the 8 input channels can be configured for either high or low level voltage inputs, and for rising or falling edges. Config8_5 is a four digit code to configure the INT8's channels 5 through 8. Config4_1 is a four digit code to configure the INT8's channels 1 through 4. The digits represent the channels in descending order left to right. For example, the code entered for Config8_5 to program channels 8 and 6 to capture the rising edge of a high level voltage, and channels 5 and 7 to capture the falling edge of a low level voltage would be "0303". See section 2 of the INT8 manual for information about the specification requirements of high and low level voltage signals.
<table>
<tr><th>Digit</th><th>Edge</th></tr>
<tr><td>0</td><td>High level, rising edge</td></tr>
<tr><td>1</td><td>High level, falling edge</td></tr>
<tr><td>2</td><td>Low level, rising edge</td></tr>
<tr><td>3</td><td>Low level falling edge</td></tr>
</table></td></tr>
<tr><td>Funct8_5
Funct4_1
Constants</td><td>Each of the 8 input channels can be independently programmed for one of eight different timing functions. Funct8_5 is a four digit code to program the timing functions of INT8 channels 5 through 8. Funct4_1 is a four digit code to program the timing functions of INT8 channels 1 through 4. See section 5.3 of the INT8 manual for further details about these functions.
<table>
<tr><th>Digit</th><th>Results</th></tr>
<tr><td>0</td><td>None</td></tr>
<tr><td>1</td><td>Period (msec) between edges on this channel</td></tr>
<tr><td>2</td><td>Frequency (kHz) of edges on the channel</td></tr>
<tr><td>3</td><td>Time between an edge on the previous channel and the edge on this channel (msec)</td></tr>
<tr><td>4</td><td>time between an edge on channel 1 and the edge on this channel (msec)</td></tr>
<tr><td>5</td><td>Number of edges on channel 2 between the last edge on channel 1 and the edge on this channel using linear interpolation</td></tr>
<tr><td>6</td><td>Low resolution frequency (kHz) of edges on this channel</td></tr>
<tr><td>7</td><td>Total number of edges on this channel since last interrogation</td></tr>
<tr><td>8</td><td>Integer number of edges on channel 2 between the last edge on channel 1 and the edge on this channel.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td></td><td>For example, 4301 in the second function parameter means to return 3 values: the period for channel 1, (nothing for channel 2) the time between an edge on channel 2 and an edge on channel 3, and the time between an edge on channel 1 and an edge on channel 4. The values are returned in the sequence of the channels, 1 to 16.

Note: the destination array must be dimensioned large enough to hold all the functions requested.</td></tr>
<tr><td>OutputOpt</td><td>Code to select one of the five different output options. The Output Option that is selected will be applied to the data collection for all of the INT8 channels. The numeric code for each option is listed below with a brief explanation of each. See the INT8 manual for detailed explanations of each option.

<table>
<tr><th>Code</th><th>Result</th></tr>
<tr><td>0:</td><td>Average of the event data since the last time that the INT8 was interrogated by the datalogger. If no edges were detected, 0 will be returned for frequency and count functions, and 99999 will be returned for the other functions. The INT8 ceases to capture events during communications with the logger, thus some edges may be lost.</td></tr>
<tr><td>32768</td><td>Continuous averaging, which is utilized when input frequencies have a slower period than the execution interval of the datalogger. If an edge was not detected for a channel since the last time that the INT8 was polled, then the datalogger will not update the input location for that channel. The INT8 will capture events even during communications with the datalogger.</td></tr>
<tr><td>nnnn</td><td>Averages the input values over "nnnn" milliseconds. The datalogger program is delayed by this instruction while the INT8 captures and processes the edges for the specified time duration and sends the results back to the logger. If no edges were detected, 0 will be returned for frequency and count functions, and 99999 will be returned for the other functions.</td></tr>
</table></td></tr>
<tr><td></td><td>-nnnn</td><td>Instructs the INT8 to capture all events until "nnnn" edges have occurred on channel 1, or until the logger addresses the INT8 with the CaptureTrig argument true, or until 8000 (storage space limitation) events have been captured. When the CaptureTrig argument is true, the INT8 will return up to the last nnnn events for each of the programmed INT8 channels, reset its memory and begin capturing the next nnnn events. The Dest array must be dimensioned large enough to receive the captured events.</td></tr>
<tr><td></td><td>-9999</td><td>Causes the INT8 to perform a self memory test. The signature of the INT8's PROM is returned to the datalogger.
<table>
<tr><th>RESULT CODE</th><th>DEFINITION</th></tr>
<tr><td>0</td><td>Bad ROM</td></tr>
<tr><td>-0</td><td>Bad ROM, & bad RAM</td></tr>
<tr><td>positive integer</td><td>ROM signature, good RAM</td></tr>
<tr><td>negative integer</td><td>ROM signature, bad RAM</td></tr>
</table></td></tr>
</table>

Parameter *& Data Type*	**Enter**
CaptureTrig *Constant, Variable, or Expression*	This argument is used when the "Capture All Events" output option is used. When CaptureTrig is true, the INT8 will return the last *nnnn* events.
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

SDMSpeed (BitPeriod)

Changes the rate that the CR5000 uses to clock the SDM data. Slowing down the clock rate may be necessary when long cables lengths are used to connect the CR5000 and SDM devices.

Parameter *& Data Type*	**Enter**
BitPeriod *Constant or variable*	The time per bit, in microseconds. Initial Setting (default): 30.0 µS Resolution: 50 µS Min Setting: 8 µS Max Setting 3 mS SDMSpeed(30) gives: 30.0 µS SDMSpeed(k) gives: bit_rate = INT(k*20) * Resolution When calculating SDMSpeed(k), the loggers round down to the next higher bit rate.

SDMTrigger

When SDMTrigger is executed, the CR5000 sends a "measure now" group trigger to all connected SDM devices. SDM stands for Synchronous Device for Measurement. SDM devices make measurements independently and send the results back to the datalogger serially. The SDMTrigger instruction allows the CR5000 to synchronize when the measurements are made. Subsequent Instructions communicate with the SDM devices to collect the measurement results. Not all SDM devices support the group trigger; check the manual on the device for more information.

SIO4 (Dest, Reps, SDMAddress, Mode, Command, Param1, Param2, ValuesPerRep, Multiplier, Offset)

The SIO4 instruction is used to control and transmit/retrieve data from a Campbell Scientific SIO4 Interface (4 Channel Serial Input/Output device). See the SDM-SIO4 Serial Input Interface manual for operation details.

<table>
<tr><th>Parameter
& Data Type</th><th>Enter</th></tr>
<tr><td>Dest</td><td>The Dest parameter is the variable in which to store the results of the instruction when retrieving data from the SIO4. If data is being sent to the SIO4, then Dest becomes the source array for the data to be sent. The Dest array must be at least as large as the Reps parameter value multiplied by the ValuesPerRep parameter value.</td></tr>
<tr><td>Reps</td><td>The Reps parameter defines the number of sequential SIO4s that will be called by the instruction. The datalogger will poll the SIO4 with the address set by the Address parameter first, receive or send the number of values set by the ValuesPerRep parameter next, and then poll the SIO4 with the next sequential address. If the Reps parameter is 2, the ValuesPerRep is 3, and the Command parameter is set to receive, then three values from the first SIO4 would be sent to the first three elements of the Dest array, and three values from the second SIO4 would be received and written to the forth through sixth elements of the Dest array.</td></tr>
<tr><td>SDMAddress</td><td>The SDMAddress parameter defines the address of the CSAT3 with which to communicate. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. If the Reps parameter is greater than 1, the datalogger will increment the SDM address for each subsequent CSAT3 that it communicates with.

The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.</td></tr>
<tr><td>Mode</td><td>The mode parameter defines which port the instruction will affect.
<table>
<tr><th>Code</th><th>Description</th></tr>
<tr><td>1</td><td>Send/Receive Port 1</td></tr>
<tr><td>2</td><td>Send/Receive Port 2</td></tr>
<tr><td>3</td><td>Send/Receive Port 3</td></tr>
<tr><td>4</td><td>Send/Receive Port 4</td></tr>
<tr><td>5</td><td>Send to all Ports (global)</td></tr>
</table></td></tr>
<tr><td>Command</td><td>The Command parameter is used to configure the SIO4. The commands are listed briefly below. See the SDM-SIO4 manual for details.
<table>
<tr><th>Code</th><th>Description</th></tr>
<tr><td>1</td><td>Poll of available data.</td></tr>
<tr><td>2</td><td>Get EPROM and memory signatures.</td></tr>
<tr><td>3</td><td>Flush all receive buffers.</td></tr>
<tr><td>4</td><td>Send data to datalogger.</td></tr>
<tr><td>5</td><td>Return number of watchdog errors, invalid command executed, and lithium battery voltage.</td></tr>
<tr><td>6</td><td>Flush transmit buffer.</td></tr>
<tr><td>7</td><td>Activate command line.</td></tr>
<tr><td>8</td><td>Poll TX buffers for data.</td></tr>
<tr><td>9</td><td>Flush converted data buffer.</td></tr>
<tr><td>66</td><td>Send single-byte data to datalogger.</td></tr>
<tr><td>67</td><td>Get return code.</td></tr>
<tr><td>320</td><td>Send byte data to SDM-SIO4.</td></tr>
<tr><td>321</td><td>Execute command line command.</td></tr>
<tr><td>1024</td><td>Send string to SIO4.</td></tr>
<tr><td>1025</td><td>Transmit a byte.</td></tr>
<tr><td>1026</td><td>Serial port status.</td></tr>
<tr><td>1027</td><td>Manual handshake mode.</td></tr>
</table></td></tr>
</table>

Parameter *& Data Type*	Enter
	2049 Communication parameters. 2054 Set up receive filter. 2304 Transmit string and/or data to device (formatter/filter). 2305 Transmit bytes.
Param1	Param1 is the first parameter that should be passed on to the SIO4 for the selected Command. Refer to the SDM-SIO4 manual for details.
Param2	Param2 is the second parameter that should be passed on to the SIO4 for the selected Command. Refer to the SDM-SIO4 manual for details.
ValuesPerRep	The ValuesPerRep parameter is the number of values to be sent or received from each SIO4 each time this instruction is performed.
Mult, Offset	These parameters are the multiplier and offset with which to scale the values received by the datalogger from the SIO4.

SW8A (Dest, Reps, SDMAddress, FunctOp, SW8AStartChan, Mult, Offset)

The SW8A instruction is used to control the SDM-SW8A Eight-Channel Switch Closure module, and store the results of its measurements to a variable array.

Parameter *& Data Type*	Enter
Dest *Variable or Array*	The variable in which to store the results of the SW8A measurement. The variable array for this parameter must be dimensioned to the number of Reps.
Reps *Constant*	The number of channels that will be read on the SW8A. If (StartChan +Reps –1) is greater than 8, measurement will continue on the next sequential SW8A. In this instance, the addresses of the SDM devices must be consecutive.
SDMAddress *Constant*	The address of the first SW8A with which to communicate. Valid SDM addresses are 0 through 15. If the SDMTrigger instruction is used in the program, address 15 should not be used. If the Reps parameter used more channels than are available on the first SW8A, the datalogger will increment the SDM address for each subsequent device that it communicates with.
FunctOp *Constant*	The FunctOp is used to determine the result that will be returned by the SW8A. **Numeric Code** — **Function** 0 — Returns the state of the signal at the time the instruction is executed. A 0 is stored for low and a 1 is stored for high. 1 — Returns the duty cycle of the signal. The result is the percentage of time the signal is high during the scan interval. 2 — Returns a count of the number of positive transitions of the signal. 3 — Returns a value indicating the condition of the module: **positive integer:** ROM and RAM are good **negative value:** RAM is bad **Zero:** ROM is bad

Parameter *& Data Type*	**Enter**
StartChan *Constant*	The first channel that should be read on the SW8A. If the Reps parameter is greater than 1, measurements will be made on sequential channels.
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

Parameter *& Data Type*	**Enter**
StartChan *Constant*	The first channel that should be read on the SW8A. If the Reps parameter is greater than 1, measurements will be made on sequential channels.
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

Section 8. Processing and Math Instructions

Operators

^	Raise to Power
*	Multiply
/	Divide
+	Add
-	Subtract
=	Equals
<>	Not Equal
>	Greater Than
<	Less Than
>=	Greater Than or Equal
<=	Less Than or Equal

ABS (Source)

Returns the absolute value of a number.

Syntax
x = ABS (*source***)**

Remarks
Source can be any valid numeric expression. The absolute value of a number is its unsigned magnitude. For example, **ABS(**-1**)** and **ABS(**1**)** both return 1.

ABS Function Example
The example finds the approximate value for a cube root. It uses Abs to determine the absolute difference between two numbers.

```
Dim Precision, Value, X, X1, X2              'Declare variables.
Precision = .00000000000001
Value = Volt(3)                              'Volt(3) will be evaluated.
X1 = 0: X2 = Value                           'Make first two guesses.
'Loop until difference between guesses is less than precision.
Do Until Abs(X1 - X2) < Precision
X = (X1 + X2) / 2
If X * X * X - Value < 0 Then                'Adjust guesses.
      X1 = X
Else
      X2 = X
End If
Loop

'X is now the cube root of Volt(3).
```

ACOS (Source)

The ACOS function returns the arc cosine of a number.

Syntax
x = ACOS *(source)*

Remarks
The source can be any valid numeric expression that has a value between -1 and 1 inclusive.

The ACOS function takes the ratio of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side adjacent to the angle divided by the length of the hypotenuse. The result is expressed in radians and is in the range $-\pi/2$ to $\pi/2$ radians.

To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

ACOS is the inverse trigonometric function of COSINE, which takes an angle as its argument and returns the length ratio of the side adjacent to the angle to the hypotenuse.

ACOS Function Example
The example uses ACOS to calculate π. By definition, a full circle is 2π radians. ACOS(0) is $\pi/2$ radians (90 degrees).

```
Public Pi                          'Declare variables.
Pi = 2 * ACOS( 0 )                 'Calculate Pi.
```

AND

Used to perform a logical conjunction on two expressions.

Syntax
result **=** *expr1* **And** *expr2*

Remarks
If, and only if, both expressions evaluate True, result is True. If either expression evaluates False, result is False. The following table illustrates how result is determined:

If *expr1* **is**	**And** *expr2* **is**	**The** *result* **is**
True	True	True
True	False	False
False	True	False
False	False	False

The And operator also performs a bit-wise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following truth table:

If bit in *expr1* **is**	**And bit in** *expr2* **is**	**The** *result* **is**
0	0	0
0	1	0
1	0	0
1	1	1

And Operator Example
The example assigns a value to Msg that depends on the value of variables A, B, and C, assuming that no variable is a Null. If A = 10, B = 8, and C = 6, both expressions evaluate True. Because both expressions are True, the And expression is also True.

```
Dim A, B, C, Msg                    'Declare variables.
A = 10: B = 8: C = 6                'Assign values.
If A > B And B > C Then              'Evaluate expressions.
      Msg = True
Else
      Msg = False
End If
```

ASIN (Source)

The ASIN function returns the arc sin of a number.

Syntax
x = ASIN *(source)*

Remarks
Source can be any valid numeric expression that has a value between -1 and 1 inclusive.

The ASIN function takes the ratio of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite to the angle divided by the length of the hypotenuse. The result is expressed in radians and is in the range $-\pi/2$ to $\pi/2$ radians.

To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

ASIN is the inverse trigonometric function of Sin, which takes an angle as its argument and returns the length ratio of the side opposite the angle to the hypotenuse.

ASIN Function Example
The example uses ASIN to calculate π. By definition, a full circle is 2π radians. ASIN(1) is $\pi/2$ radians (90 degrees).

```
Public Pi                           'Declare variables.
Pi = 2 * ASin( 1 )                  'Calculate Pi.
```

ATN (Source)

Returns the arctangent of a number.

Syntax
x = ATN (*source***)**

Remarks
Source can be any valid numeric expression.

The **Atn** function takes the ratio of two sides of a right triangle and returns the corresponding angle. The ratio is the length of the side opposite the angle divided by the length of the side adjacent to the angle. The result is expressed in radians and is in the range -π/2 to π/2 radians.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Atn is the inverse trigonometric function of Tan, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse Atn with the cotangent, which is the simple inverse of a tangent (1/tangent).

Atn FunctionExample

The example uses Atn to calculate π. By definition, a full circle is 2π radians. Atn(1) is π/4 radians (45 degrees).

```
Dim  Pi                 'Declare variables.
Pi = 4 * Atn(1)         'Calculate Pi.
```

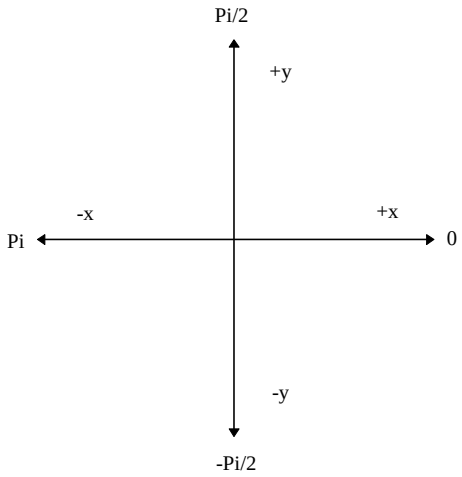
ATN2()

The ATN2 function returns the arctangent of y/x.

Syntax
x = ATN2 *(Y, X)*

Remarks
ATN2 function calculates the arctangent of Y/X returning a value in the range from Pi to -Pi radians, using the signs of both parameters to determine the quadrant of the return value. ATN2 is defined for every point other than the origin (X = 0 and Y = 0). Y and X can be variables, constants, or expressions.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.



ATN2 is the inverse trigonometric function of TAN, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse ATN2 with the cotangent, which is the simple inverse of a tangent (1/tangent).

ATN2 Function Example
The example uses ATN2 to calculate π. By definition, a full circle is 2π radians. ATN2(1,1) is π/4 radians (90 degrees).

```
Dim Pi                  'Declare variables.
Pi = 4 * ATN2( 5, 5 )   'Calculate Pi.
```

AvgSpa (Dest, Swath, Source)

Computes the spatial average of the values in the source array.

Syntax
AvgSpa (*Dest, Swath, Source***)**

Remarks
Find the average of the values in the given array and place the result in the variable named in Dest. The Source must be a particular element in an array (e.g., Temp(1)); it is the first element in the array to include in the average. The Swath is the number of elements to include in the average.

$$Dest = \frac{\sum_{i=j}^{i=j+swath} X(i)}{swath}$$

Where *X(j)* = Source

Parameter & Data Type	Enter
Dest *Variable*	The variable in which to store the results of the instruction.
Swath *Constant*	The number of values of the source array to average.
Source *Array*	The name of the variable array that is the input for the instruction.

Average Spatial Output Example

This example uses AvgSpa to find the average value of the five elements Temp(6) through Temp(10) and store the result in the variable AvgTemp.

AvgSpa(AvgTemp, 5, Temp(6)**)**

AvgRun (Dest, Reps, Source, Number)

Calculates a running average of a measurement or calculated value.

Syntax
AvgRun ***(****Dest, Reps, Source, Number****)***

Remarks
AvgRun is used to create a running average. A running average is the average of the last N values where N is the number of values.

$$Dest = \frac{\sum_{i=1}^{i=N} X_i}{N}$$

Where X_N is the most recent value of the source variable and X_{N-1} is the previous value (X_1 is the oldest value included in the average, i.e., N-1 values back from the most recent).

Parameter & Data Type	Enter
Dest *Variable or Array*	The variable or array in which to store the average(s).
Reps *Constant*	When the source is an array, this is the number of variables in the array to calculate averages for. When the source is not an array or only a single variable of the array is to be averaged, reps should be 1.
Number *Constant*	The number of values to include in the running average..
Source *Array*	The name of the variable or array that is to be averaged.

Example

```
BeginProg          'Program begins here
    Scan( RATE, RUNITS, 0, 0 )   'Scan 1(mSecs),
    '_____________________ Volt Blocks _____________________
    VoltDiff(HiVolts, VREP1, VRNG1, 5, 1, 0, VDLY1, VINT1, VMULT1,
        VOSET1)
    AvgRun(AvgOut,1,HiVolts,100  )  'Put the average of 100 HiVolts in
        AvgOut
    CallTable MAIN        'Go up and run Table MAIN
    Next Scan             'Loop up for the next scan
EndProg            'Program ends here
```

Cos (Source)

Returns the cosine of an angle specified in radians.

Syntax
x = Cos (*source***)**

Remarks
Source can be any valid numeric expression measured in radians.

The **Cos** function takes an angle and returns the ratio of two sides of a right triangle. The ratio is the length of the side adjacent to the *angle* divided by the length of the hypotenuse. The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Cos FunctionExample
The example uses Cos to calculate the cosine of an angle with a user-specified number of degrees.

```
Dim Degrees, Pi, Radians, Ans      'Declare variables.
BeginProg
Pi = 4 * Atn(1)                    'Calculate Pi.
Degrees = Volts(1)                 'Get value to convert.
Radians = Degrees * (Pi / 180)     'Convert to radians.
Ans = Cos(Radians)                 'The Cosine of Degrees.
EndProg
```

CosH (Source)

The COSH function returns the hyperbolic cosine of an expression or value.

Syntax
x = COSH *(source)*

Remarks
The COSH function takes a value and returns the hyperbolic cosine [COSH(x) = 0.5(ex + e-x)] for that value.

COSH Function Example
The example uses COSH to calculate the hyperbolic cosine of a voltage input and store the result in the Ans variable.

```
Public Volt1, Ans                    'Declare variables.
BeginProg
      Scan (1,Sec,3,0)
      VoltDiff (Volt1,1,mV5000,1,True ,200,500,1.0,0)  'Return voltage on
                                                        DiffChan1
      Ans = COSH( Volt1 )
      NextScan
EndProg
```

Spatial Covariance

The CovSpa instruction computes the covariance(s) of sets of data that are loaded into arrays.

Syntax

CovSpa(Dest, NumOfCov, SizeOfSets, CoreArray; DatArray)

CovSpa calculates the covariance(s) between the data in the CoreArray and one or more data sets in the DatArray. The covariance of the sets of data X and Y is calculated as:

$$Cov(X,Y) = \frac{\sum_{i=1}^{n} X_i \cdot Y_i}{n} - \frac{\sum_{i=1}^{n} X_i \sum_{i=1}^{n} Y_i}{n^2}$$

Where n is the number of values in each data set (**SizeofSets**). X_i and Y_i are the individual values of X and Y.

Parameter *& Data Type*	**Enter**
Dest *Variable or Array*	The Variable in which to store the results of the instruction. When multiple covariances are calculated, the results are stored in an array with the variable name. An array must be dimensioned to at least the value of NumOfCov.
NumOfCov *Constant*	The number of covariances to be calculated. If four data sets are to be compared against a fifth set, this would be set to four.
SizeOfSets *Constant*	The number of values in the data sets for the covariance calculations.
CoreArray Array	The array that holds the core data set. The covariance of core data with each of the other sets is calculated independently. The data need to be consecutive in the array. If the first data value is not the first point of the array, the first point of the data set must be specified in this parameter.
DatArray *Array*	The array that contains the data set(s) for calculating the covariance with the CoreSet. When multiple covariances are calculated, the data sets have to be loaded consecutively into one array. The array must be dimensioned to at least the value of NumOfCov multiplied by SizeOfSets. For example, if each set of data has 100 elements (SizeOfSets), and there are 4 covariances (NumOfCov) to be calculated, then the DatArray needs to be dimensioned to 4 x 100 = 400. If the first value of the first set is not the first point of the array, the first point of the data set must be specified in this parameter.

Exp

Returns e (the base of natural logarithms) raised to a power.

Syntax
x = Exp (*source***)**

Remarks
If the value of the source exceeds 709.782712893, an Overflow error occurs. The constant e is approximately 2.718282.

NOTE The Exp function complements the action of the Log function and is sometimes referred to as the antilogarithm.

Exp FunctionExample
The example uses Exp to calculate the value of e. Exp(1) is e raised to the power of 1.

'Exp(x) is e ^x so Exp(1) is e ^1 or e.

```
Dim ValueOfE          'Declare variables.
BeginProg
ValueOfE = Exp(1)     'Calculate value of e.
EndProg
```

FFTSpa (Dest, N, Source, Tau, Units, Option)

The FFTSpa performs a Fast Fourier Transform on a time series of measurements stored in an array and places the results in an array. It can also perform an inverse FFT, generating a time series from the results of an FFT. Depending on the output option chosen, the output can be: 0) The real and imaginary parts of the FFT; 1) Amplitude spectrum. 2) Amplitude and Phase Spectrum; 3) Power Spectrum; 4) Power Spectral Density (PSD); or 5) Inverse FFT.

The difference between the FFT instruction (Section 6) and FFTSpa is that FFT is an output instruction that stores the results in a data table and FFTSpa stores its results in an array.

Parameter *& Data Type*	**Enter**
Dest *Array*	The array in which to store the results of FFT.
Source *Variable*	The name of the Variable array that contains the input data for the FFT.
N *Constant*	Number of points in the original time series. The number of points must be a power of 2 (i.e., 512, 1024, 2048, etc.).
Tau *Constant*	The sampling interval of the time series.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>Units
Constant</td><td>The units for Tau.
<table>
<tr><th>Alpha Code</th><th>Numeric Code</th><th>Units</th></tr>
<tr><td>USEC</td><td>0</td><td>microseconds</td></tr>
<tr><td>MSEC</td><td>1</td><td>milliseconds</td></tr>
<tr><td>SEC</td><td>2</td><td>seconds</td></tr>
<tr><td>MIN</td><td>3</td><td>minutes</td></tr>
</table></td></tr>
<tr><td>Options
Constant</td><td>A code to indicate what values to calculate and output.
<table>
<tr><th>Code</th><th>Result</th></tr>
<tr><td>0</td><td>FFT. The output is N/2 complex data points, i.e., the real and imaginary parts of the FFT. The first pair is the DC component and the Niquist component. This first pair is an exception because the DC and niquist components have no imaginary part.</td></tr>
<tr><td>1</td><td>Amplitude spectrum. The output is N/2 magnitudes. With Acos(wt); A is magnitude.</td></tr>
<tr><td>2</td><td>Amplitude and Phase Spectrum. The output is N/2 pairs of magnitude and phase; with Acos(wt - ϕ); A is amplitude, ϕ is phase (-π,π).</td></tr>
<tr><td>3</td><td>Power Spectrum. The output is N/2 values normalized to give a power spectrum. With Acos(wt - ϕ), the power is A^2 / 2. The summation of the N/2 values yields the total power in the time series signal.</td></tr>
<tr><td>4</td><td>Power Spectral Density (PSD). The output is N/2 values normalized to give a power spectral density (power per herz). The Power Spectrum multiplied by T = N*tau yields the PSD. The integral of the PSD over a given bandwidth yields the total power in that band. Note that the bandwidth of each value is 1/T herz.</td></tr>
<tr><td>5</td><td>Inverse FFT. The input is N/2 complex numbers, organized as in the output of option 0, which is assumed to be the transform of some real time series. The output is the time series whose FFT would result in the input array.</td></tr>
</table></td></tr>
</table>

T = N*tau: the length, in seconds, of the time series.
Processing field: "FFT,N,tau,option". Tick marks on the x axis are 1/(N*tau) Herz. N/2 values, or pairs of values, are output, depending upon the option code.

Normalization details:

Complex FFT result i, i = 1 .. N/2: ai*cos(wi*t) + bi*sin(wi*t).
wi = 2π(i-1)/T.
ϕi = atan2(bi,ai) (4 quadrant arctan)
Power(1) = $(a1^2 + b1^2)/N^2$ (DC)
Power(i) = $2*(ai^2 + bi^2)/N^2$ (i = 2..N/2, AC)
PSD(i) = Power(i) * T = Power(i) * N * tau
A1 = $sqrt(a1^2 + b1^2)/N$ (DC)
Ai = $2*sqrt(ai^2 + bi^2)/N$ (AC)

Notes:

- Power is independent of the sampling rate (1/tau) and of the number of samples (N).
- The PSD is proportional to the length of the sampling period (T=N*tau), since the "width" of each bin is 1/T.
- The sum of the AC bins (excluding DC) of the Power Spectrum is the Variance (AC Power) of the time series.
- The factor of 2 in the Power(i) calculation is due to the power series being mirrored about the Niquist frequency N/(2*T); only half the power is represented in the FFT bins below N/2, with the exception of DC. Hence, DC does not have the factor of 2.
- The Inverse FFT option assumes that the data array input is the transform of a real time series. Filtering is performed by taking an FFT on a data set, zeroing certain frequency bins, and then taking the Inverse FFT. Interpolation is performed by taking an FFT, zero padding the result, and then taking the Inverse FFT of the larger array. The resolution in the time domain is increased by the ratio of the size of the padded FFT to the size of the unpadded FFT. This can be used to increase the resolution of a maximum or minimum, as long as aliasing is avoided.

Frac (Source)

Returns the fractional part of a number.

Syntax
x = Frac (*source***)**

Remarks
Returns the fractional portion of the *number* within the parentheses.

Frac FunctionExample
The example uses Frac function.

GetRecord (Dest, TableName, RecsBack)

Retrieves one record from a data table.

Syntax
GetRecord *(Dest, TableName, RecsBack)*

Remarks
The GetRecord instruction retrieves one entire record from a data table. The destination array must be dimensioned large enough to hold all the fields in the record.

Parameter *& Data Type*	**Enter**
Dest *Array*	The destination variable array in which to store the fields of the record. The array must be dimensioned large enough to hold all the fields in the record.
TableName *name*	The name of the data table to retrieve the record from.
RecsBack *Const. Or variable*	The number of records back from the most recent record stored to go to retrieve the record (1 record back is the most recent).

IfTime

The IfTime instruction is used to return a number indicating True (-1) or False (0) based on the datalogger's real-time clock.

Syntax

IfTime (TintoInt, Interval, Units)

The IfTime function returns True (-1) or False (0) based on the scan clock. Time is kept internally by the datalogger as the elapsed time since January 1, 1990, at 00:00:00 hours. The interval is synchronized with this elapsed time (i.e., the interval is true when the Interval divides evenly into this elapsed time). The time into interval allows an offset to the interval. The IfTime instruction can be used to set the value of a variable or it can be used as an expression for a condition.

The scan clock that the IfTime function checks has the time resolution of the scan interval (i.e., it remains fixed for an entire scan and increments for the next scan). IfTime must be within a scan to function.

The window of time in which the IfTime instruction is true is 1 of its specified **Units**. For example, if IfTime specifies 0 into a 10 minute interval, it could be true when the scan clock specified any time within the first minute of the ten minute interval. With 0 into a 600 second interval, the interval is still 10 minutes but it could only be true during the first 1 second of that interval.

IfTime will only return true once per interval. For example, a program with a 1 second scan that tests IfTime(0,10, min) -- 0 minutes into a 10 minute interval – each scan will execute the instruction 60 times during the minute that it could be true. It will only return true the first time that it is executed, it will not return true again until another interval has elapsed.

Parameter *& Data Type*	**Enter**
TintoInt *constant*	The time into interval sets an offset from the datalogger's clock to the interval at which the IfTime will be true. For example, if the Interval is set at 60 minutes, and TintoInt is set to 5, IfTime will be True at 5 minutes into the hour, every hour, based on the datalogger's real-time clock. If the TintoInt is set to 0, the IfTime statement is True at the top of the hour.
Interval *constant*	The Interval is how often IfTime will be True.
Units *Constant*	The time units for **TintoInt** and **Interval**

Alpha Code	**Numeric Code**	**Units**
Usec	0	microseconds
Msec	1	milliseconds
Sec	2	seconds
Min	3	minutes
Hr	4	hours
Day	5	days

IIF

The IIF function evaluates a variable or expression and returns one of two results based on the outcome of that evaluation.

Syntax

Result = **IIF**(Expression, TrueValue, FalseValue)

Parameter *& Data Type*	**Enter**
Expression *Expression or Variable*	The Variable or expression to test.

Value	Result
≠0	True: return **TrueValue**
0	False: return **FalseValue**

TrueValue *Constant Variable or Expression*	The Value (or expression determining the value) to return if the test condition is true
FalseValue *Constant Variable or Expression*	The Value (or expression determining the value) to return if the test condition is False

IMP

The IMP function is used to perform a logical implication on two expressions.

Syntax
result = expression1 IMP expression2

Remarks
The following table illustrates how Result is determined:

If expression1 is	And expression2 is	The result is
True	True	True
True	False	False
True	Null	Null
False	True	True
False	False	True
False	Null	True
Null	True	True
Null	False	Null
Null	Null	Null

The IMP operator performs a bitwise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following table:

If bit in expression1 is	And bit in expression2 is	The result is
0	0	1
0	1	1
1	0	0
1	1	1

Int, Fix

Return the integer portion of a number.

Syntax
Int (*source***)**

Fix (*source***)**

Remarks
The source can be any valid numeric expression. Both **Int** and **Fix** remove the fractional part of *source* and return the resulting integer value.

If the numeric expression results in a Not-a-Number, **Int** and *Fix* return a Not-a-Number.

The difference between **Int** and **Fix** is that if *number* is negative, **Int** returns the first negative integer less than or equal to *number*, whereas **Fix** returns the first negative integer greater than or equal to *number*. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Fix(*number***)** is equivalent to:
Int and Fix Function Example

This example illustrates the use of Int and Fix.

```
Dim A, B, C, D          'Declare variables.
BeginProg
A = Int(-99.8)          'Returns -100
B = Fix(-99.8)          'Returns -99
C = Int(99.8)           'Returns 99
D = Fix(99.8)           'Returns 99
EndProg
```

Log (Source)

Returns the natural logarithm of a number.

Syntax
x = Log (*source***)**

Remarks
The source can be any valid numeric expression that results in a value greater than 0. The natural logarithm is the logarithm to the base e. The constant e is approximately 2.718282.

You can calculate base-n logarithms for any *number* x by dividing the natural logarithm of x by the natural logarithm of n as follows:

Logn(x) = **Log**(x) / **Log**(n)

The following example illustrates a procedure that calculates base-10 logarithms:

Log10 = **Log**(X) / **Log**(10)

Log Function Example

The example calculates the value of e, then uses the Log function to calculate the natural logarithm of e to the first, second, and third powers.

```
Dim I, M                    'Declare variables.
BeginProg
M = Exp(1)
For I = 1 To 3              'Do three times.
      M =Log(Exp(1) ^ I)
Next I
EndProg
```

LOG10 (number)

The LOG10 function returns the base 10 logarithm of a number.

Syntax
LOG10(number)

Remarks
The LOG10 function returns the logarithm base 10 of a number.

The Number argument can be any valid numeric expression that has a value greater than 0. You can calculate base-n logarithms for any number x by dividing the logarithm base 10 of x by the logarithm base 10 of n as follows:

LOGN(x) = LOG10(x) / LOG10(n)

LOG10 Function Example

This example uses the LOG10 instruction to calculate the log base 2 of 1000.

```
Dim LOG2_1000              'Declare variables.
LOG2_1000 = LOG10(1000)/ LOG10(2)
```

MemoryTest (Dest)

Stores the results of the most recent memory test in a variable.

Syntax
MemoryTest (Dest**)**

Remarks
The CR5000 tests CPU RAM and Task Sequencer memory when it compiles and runs a program. MemoryTest stores the results of this compile test in a variable

Parameter *& Data Type*	**Enter**
Dest *Variable*	The variable in which to store the results of the memory test

Result	**Meaning**
0	No problems found
1	Error in CPU RAM
2	Error in Task Memory
3	Error in both CPU RAM and Task Memory

MaxSpa (Dest, Swath, Source)

Finds the maximum value in an array.

Syntax
MaxSpa(Dest, Swath, Source**)**

Remarks
Find the maximum value in the given array and place the result in the variable named in Dest. The Source must be a particular element in an array (e.g., Temp(1)); it is the first element in the array

Parameter *& Data Type*	**Enter**
Dest *Variable*	The variable in which to store the maximum.
Swath *Constant*	The number of values of the source array in which to search for the maximum.
Source *Array*	The name of the variable array that is the input for the instruction.

MaxSpa Function Example

This example uses MaxSpa to find the maximum value of the five elements Temp(6) through Temp(10) and store the result in the variable MaxTemp.

```
MaxSpa(MaxTemp, 5, Temp(6))
```

MinSpa (Dest, Swath, Source)

Finds the minimum value in an array.

Syntax
MinSpa(Dest, Swath, Source**)**

Remarks
Find the minimum value in the given array and place the result in the variable named in Dest. The Source must be a particular element in an array (e.g., Temp(1)); it is the first element in the array to check for the minimum. The Swath is the number of elements to compare for the minimum.

Parameter *& Data Type*	**Enter**
Dest *Variable*	The variable in which to store the results of the instruction.
Swath *Constant*	The number of values of the source array in which to search of the minimum.
Source *Array*	The name of the variable array that is the input for the instruction.

MinSpa Function Example

This example uses MinSpa to find the minimum value of the five elements Temp(6) through Temp(10) and store the result in the variable MinTemp.

```
MinSpa(MinTemp, 5, Temp(6))
```

Mod

Divides two numbers and returns only the remainder.

Syntax
result = operand1 **Mod** *operand2*

Remarks
The modulus, or remainder, operator divides *operand1* by *operand2* (rounding floating-point numbers to integers) and returns only the remainder as *result*. For example, in the expression A = 19 **Mod** 6.7, A (which is result) equals 5. The operands can be any numeric expression.

Mod Operator Example

The example uses the Mod operator to determine if a 4-digit year is a leap year.

```
Dim TestYr, LeapStatus                                  'Declare variables.
TestYr = 1995
If TestYr Mod 4 = 0 And TestYr Mod 100 = 0 Then         'Divisible by 4?
      If TestYr Mod 400 = 0 Then                        'Divisible by 400?
            LeapStatus = True
      Else
            LeapStatus = False
      End If
ElseIf TestYr Mod 4 = 0 Then
      LeapStatus = True
Else
      LeapStatus = False
End If
```

Move (Dest, Swath, Source)

Moves a block or fills an array.

Syntax
Move(Dest, Reps, Source, Reps**)**

Remarks
Block Move or fill array.

Parameters: Dest array, destination reps; Source array or expression; Source reps. If source reps is less than destination reps, the remainder of destination is filled with that last value of source.

Parameter *& Data Type*	**Enter**
Dest *Variable or Array*	The variable in which to store the results of the instruction.
Reps *Constant*	The number of repetitions for the measurement or instruction.
Source *Array*	The name of the variable array that is the input for the instruction.
Reps *Constant*	The number of repetitions for the measurement or instruction.

Move Function Example

The example uses the Move function.

```
Move(x, 20, y, 20)        'move array y into array x
Move(x, 20, 0.0, 1)       'fill x with 0.0.
```

NOT

The NOT function is used to perform a logical negation on an expression.

Syntax
result = NOT expression

Remarks
The following table illustrates how Result is determined:

If expr is	The result is
True	False
False	True
Null	Null

The NOT operator also inverts the bit values of any variable and sets the corresponding bit in result according to the following truth table:

If bit in expr1 is	The result is
0	1
1	0

NOT Operator Example

The example sets the value of the variable Msg depending on the state of Flag(1).

```
Dim A, B, C, Flag(8)          'Declare variables.
Public Msg
If NOT Flag(1) Then           'Evaluate expressions.
      Msg = 10
Else
      Msg = 100.
End If
```

Or

Used to perform a logical disjunction on two expressions.

Syntax
result **=** *expr1* **Or** *expr2*

Remarks
If either or both expressions evaluate True, result is True. The following table illustrates how result is determined:

If *expr1* **is**	**And** *expr2* **is**	**The** *result* **is**
True	True	True
True	False	True
False	True	True
False	False	False

The **Or** operator also performs a bit-wise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following truth table:

If bit in *expr1* **is**	**And bit in** *expr2* **is**	**The** *result* **is**
0	0	0
0	1	1
1	0	1
1	1	1

Or Operator Example

The example sets Msg that depends on the value of variables A, B, and C, assuming that no variable is a Null. If A = 10, B = 8, and C = 11, the left expression is True and the right expression is False. Because at least one comparison expression is True, the Or expression evaluates True

```
Dim A, B, C                     'Declare variables.
A = 10: B = 8: C = 11           'Assign values.
If A > B Or B > C Then          'Evaluate expressions.
      Msg = True
Else
      Msg = False.
End If
```

PeakValley (DestPV, DestChange, Reps, Source, Hysteresis)

PeakValley is used to detect peaks and valleys (local maxima and minima) in a signal. When a new peak or valley is detected, the new peak or valley and the change from the previous peak or valley are stored in variables.

Parameter *& Data Type*	**Enter**
DestPV *Variable or array*	Variable or array in which to store the new peak or valley. When a new peak or valley is detected, the value of the peak or valley is loaded in the destination. PeakValley will continue to load the previous peak or valley until the next peak or valley is detected.
DestChange *Variable or array*	Variable or array in which to store the change from the previous peak or valley. When a new peak or valley is detected, the change from the previous peak or valley is loaded in the destination. When a new peak or valley has not yet been reached, 0 is stored in the destination. When Reps are greater than 1, the array must be dimensioned to Reps+1. The additional element is used to flag when a new peak or valley is detected in any of the source inputs. The flag element is stored after the changes [e.g., *changevar*(Reps+1)] and is set to -1 (true) when a new peak or valley is detected and set to 0 (false) when none are detected.
Reps *Constant*	The number inputs to track the peaks and valleys for. Each input is tracked independently. When reps are greater than 1 the source and DestPV arrays must be dimensioned to at least the number of repetitions; DestChange must be dimensioned to Reps+1.
Source *Variable or Array*	The variable or array containing the inputs to check for peaks and valleys.
Hysteresis *Constant Variable or expression*	The minimum amount the input has to change to be considered a new peak or valley. This would usually be entered as a constant.

The following example uses sine and cosine signal inputs to illustrate the use of PeakValley with two repetitions. Data Table PV1 stores the peaks and valleys from the cosine wave. PV2 stores the peaks and valleys from the sine wave. PV3 stores the peaks and valleys from both.

```
Public PeakV(2), Change(3),Deg
Public Dim XY(2)

Const Pi=4*ATN(1)                    'Define Pi for converting degrees to
                                     radians

DataTable(PV1,Change(1),500)         'Peaks and valleys for first signal,
                                     triggered when 'Change(1) is not 0.
    Sample(1,PeakV(1),IEEE4)         'DataTable PV1 holds the peaks and
                                     valleys for XY(1)
EndTable

DataTable(PV2,Change(2),500)         'Peaks and valleys for second signal,
                                     triggered when 'Change(2) is not 0.
    Sample(1,PeakV(2),IEEE4)         'DataTable PV2 holds the peaks and
                                     valleys for XY(2)
EndTable

'The Following table is an alternative to using separate tables for each signal.
'It stores both signals whenever there is a new peak or valley in either signal.
'The value stored for the signal that does not have a new peak will be a repeat
'of its last peak or valley.  Normally a program would not have a table storing
'peaks and valleys for several signals, it would use individual tables for the
'signals.

DataTable(PVBoth,Change(3),500)
    Sample(2,PeakV(1),IEEE4)
EndTable

BeginProg
    Scan(500,mSec,0,0)
        Deg=Deg+5
        XY(1)=Cos(Deg*Pi/180)        'Compute the cosine as input XY(1)
        XY(2)=Sin(Deg*Pi/180)        'Compute the sine as input XY(2)

        PeakValley(PeakV(1),Change(1),2,XY(1),0.1)   'Find the peaks and
                                                     'valleys for both
                                                     'inputs. Hysteresis
                                                     '= 0.1
        CallTable PV1
        CallTable PV2
        CallTable PVBoth
    Next Scan
EndProg
```

PRT (Dest, Reps, Source, Mult, Offset)

Used to calculate temperature from the resistance of an RTD.

Syntax
PRT ***(****Dest, Reps, Source, Mult, Offset****)***

Remarks
This instruction uses the result of a previous RTD bridge measurement to calculate the temperature. The input (Source) must be the ratio Rs/R0, where Rs is the RTD resistance and R0 the resistance of the RTD at 0° C.

The temperature is calculated according to the DIN 43760 specification adjusted (1980) to the International Electrotechnical Commission standard. The range of linearization is -200° C to 850° C. The error in the linearization is less than 0.001° C between -200 and +300° C, and is less than 0.003° C between -180 and +830° C. The error (T calculated - T standard) is +0.006° at -200° C and -0.006° at +850° C.

Parameter *& Data Type*	**Enter**
Dest *Variable or Array*	The variable in which to store the temperature in degrees C.
Reps *Constant*	The number of repetitions for the measurement or instruction.
Source *Variable or Array*	The name of the variable or array that contains the Rs/RO value(s).
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

Randomize (Source)

Initializes the random-number generator.

Syntax
Randomize (*source*)

Remarks
The argument *number* can be any valid numeric expression. *Number* is used to initialize the random-number generator by giving it a new seed value. If you omit *number*, the value returned by the Timer function is used as the new seed value.

If **Randomize** is not used, the Rnd function returns the same sequence of random numbers every time the program is run. To have the sequence of random numbers change each time the program is run, place a **Randomize** statement with no argument at the beginning of the program.

RealTime

Used to pick out year, month, day, hour, minute, second, usecond, day of week, and/or day of year from the CR5000 clock.

Syntax
RealTime(Dest**)**

Remarks
RealTime Example

This example uses **RealTime** to place all time segments in the Destination array. If the remark (') is removed from the first 8 Sample statements and the last Sample statement is remarked, the results will be exactly the same.

```
Public rTime(9)                     'declare as public and dimension rTime to 9
Alias rTime(1) = Year               'assign the alias Year to rTime(1)
Alias rTime(2) = Month              'assign the alias Month to rTime(2)
Alias rTime(3) = Day                'assign the alias Day to rTime(3)
Alias rTime(4) = Hour               'assign the alias Hour to rTime(4)
Alias rTime(5) = Minute             'assign the alias Minute to rTime(5)
Alias rTime(6) = Second             'assign the alias Second to rTime(6)
Alias rTime(7) = uSecond            'assign the alias uSecond to rTime(7)
Alias rTime(8) = WeekDay            'assign the alias WeekDay to rTime(8)
Alias rTime(9) = Day_of_Year        'assign the alias Day_of_Year to rTime(9)

DataTable (VALUES, 1, 100)          'set up data table
DataInterval(0, 1, mSec, 0)         'set up data table
'  Sample(1, Year, IEEE4)           'place Year in VALUES table
'  Sample(1, Month, IEEE4)          'place Month in VALUES table
'  Sample(1, Day, IEEE4)            'place Day in VALUES table
'  Sample(1, Hour, IEEE4)           'place Hour in VALUES table
'  Sample(1, Minute, IEEE4)         'place Minute in VALUES table
'  Sample(1, Second, IEEE4)         'place Second in VALUES table
'  Sample(1, uSecond, IEEE4)        'place uSecond in VALUES table
'  Sample(1, WeekDay, IEEE4)        'place WeekDay in VALUES table
'  Sample(1, Day_of_Year, IEEE4)    'place Day_of_Year in VALUES table
  Sample(9, rTime(), IEEE4)         'place all 9 segments in VALUES table
EndTable

BeginProg
  Scan (1, mSec, 0, 0)
    RealTime(rTime())
    CallTable VALUES
  Next Scan
EndProg
```

RectPolar (Dest, Source)

Converts from rectangular to polar coordinates.

Parameter *& Data Type*	**Enter**
Dest *Variable array*	Variable array in which to store the 2 resultant values. The length of the vector is stored in the specified destination element and the angle, in radians(± π), in the next element of the array
Source *Variable Array*	The variable array containing the X and Y coordinates to convert to Polar coordinates. The X value must be in the specified array element and the Y value in the next element of the array.

Example: In the following example, a counter (Deg) is incremented from 0 to 360 degrees. The cosine and sine of the angle are taken to get X and Y in rectangular coordinates. RectPolar is then used to convert to polar coordinates.

```
Dim XY(2),Polar(2),Deg,AnglDeg
Const Pi=4*ATN(1)

Alias XY(1)=X
Alias XY(2)=Y
Alias Polar(1)=Length
Alias Polar(2)=AnglRad

DataTable(RtoP,1,500)
 Sample(1,Deg,IEEE4)
 Sample(2,XY,IEEE4)
 Sample(2,Polar,IEEE4)
 Sample(1,AnglDeg,IEEE4)
EndTable

BeginProg
 For Deg=0 to 360
        XY(1)=Cos(Deg*Pi/180)      'Cos and Sin operate on radians
        XY(2)=Sin(Deg*Pi/180)
        RectPolar(Polar,XY)
        AnglDeg=Polar(2)*180/Pi    'Convert angle to degrees for
                                   comparison w/Deg
        CallTable RtoP
    Next Deg
EndProg
```

RMSSpa (Dest, Swath, Source)

Used to compute the RMS value of an array.

Syntax
RMSSpa(Dest, Swath, Source**)**

Remarks
Spatial RMS, Calculate the root mean square of values in an array.

$$Dest = \sqrt{\frac{\sum_{i=j}^{i=j+swath} \left(X(i)\right)^2}{swath}}$$

Where *X(j)* = Source

Parameter *& Data Type*	**Enter**
Dest *Variable*	The variable in which to store the RMS value.
Swath *Constant*	The number of values of the array to include in the RMS calculation.
Source *Array*	The name of the variable array that is the input for the instruction.

RmsSpa Function Example

The example uses the RmsSpa function to

RND (Source)

The RND function is used to generate a random number.

Syntax
RND(source)

Remarks
The RND function returns a single value less than 1 but greater than or equal to 0.

The same random-number sequence is generated each time the instruction is encountered because each successive call to the RND function uses the previous random number as a seed for the next number in the random-number sequence.

The value of the Number argument determines how the random number will be generated:

Value	**Description**
< 0	The same number each time, as determined by Number
> 0	The next random number in the sequence
= 0	The number most recently generated
Number omitted	The next random number in the sequence

To have the program generate a different random-number sequence each time it is run, use the Randomize statement with no Number argument to initialize the random-number generator before RND is called.

To produce random integers in a given range, use this formula:

INT((upperbound - lowerbound + 1) * RND + lowerbound)

Here, upperbound is the highest number in the range, and lowerbound is the lowest number in the range.

StrainCalc (Dest, Reps, Source, BrZero, BrConfig, GF, *v*)

Converts the output of a bridge measurement instruction to microstrain.

Syntax
StrainCalc (Dest, Reps, Source, BrZero, BrConfig, GF, *v*)

Remarks
Calculates microstrain, $\mu\in$, from the appropriate formula for the bridge configuration. All are electrically full bridges , the quarter bridge, half bridge and full bridge strain gages refer to the number of active elements (i.e., strain gages), 1,2, or 4 respectively.

Parameter *& Data Type*	**Enter**
Dest	Variable to store strain in.
Reps	Number of strains to calculate, Destination, source, and zero variables must be dimensioned accordingly.
BrConfig	Bridge configuration code for strain gages The bridge configuration code can be entered as a positive or negative number: + code: $V_r = 0.001(Source - Zero)$; bridge configured so its output decreases with increasing strain. - code: $V_r = -0.001(Source - Zero)$; bridge configured so output increases with strain. This is the configuration for a quarter bridge using CSI's 4WFB350 Terminal Input Module (i.e., enter the bridge configuration code as -1 for 1/4 bridge with TIM.)

Code	**Configuration**
1	Quarter bridge strain gauge $\mu\varepsilon = \dfrac{-4 \cdot 10^6 V_r}{GF(1+2V_r)}$
2	Half bridge strain gauge, one gage parallel to strain, the other at 90° to strain: $\mu\varepsilon = \dfrac{-4 \cdot 10^6 V_r}{GF[(1+\nu) - 2V_r(\nu - 1)]}$
3	Half bridge strain gauge, one gage parallel to $+\varepsilon$, the other parallel to $-\varepsilon$: $\mu\varepsilon = \dfrac{-2 \cdot 10^6 V_r}{GF}$

	4	Full bridge strain gage, 2 gages parallel to $+\varepsilon$, the other 2 parallel to $-\varepsilon$: $\mu\varepsilon = \frac{-10^6 V_r}{GF}$
	5	Full bridge strain gage, half the bridge has 2 gages parallel to $+\varepsilon$ and $-\varepsilon$: the other half $+\nu\varepsilon$ and $-\nu\varepsilon$: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF(\nu+1)}$
	Code	**Configuration**
	6	Full bridge strain gage, one half $+\varepsilon$ and $-\nu\varepsilon$, the other half $-\nu\varepsilon$ and $+\varepsilon$.: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF[(\nu+1) - V_r(\nu-1)]}$
Source	The source variable array for the measurement(s), the input is expected as millivolts out per volt in (the result of the full bridge instruction with a multiplier of 1 and an offset of 0.	
BrZero	The variable array that holds the unstrained reading(s) in millivolts out per volt in.	
GF	Gage Factor. The gage factor can be entered as a constant used for all repetitions or a variable array can be loaded with individual gage factors which are automatically used with each rep. To use an array enter the parameter as *arrayname*(), with no element number in the parentheses.	
v	Poisson ratio, enter 0 if it does not apply to configuration.	

StrainCalc Example

This example uses StrainCalc to find the microstrain value of a bridge output.

```
'      Program name: STRAIN.DLD

Public Count, ZStrain, StMeas, Strain, Flag(8)     ' Declare all variables as
                                                   public

'Data Table STRAINS samples every measurement when user Sets Flag(1)
High

DataTable(STRAINS,Flag(1),-1)
      DataInterval(0,0,0,100)                      'Interval = Scan, 100 lapses
      Sample (1,Strain,Ieee4)
EndTable

'DataTable ZERO_1 stores the "zero" measurements

DataTable(ZERO_1,Count>99,100)                     'Trigger on Count 100
      Average(1,ZStrain,IEEE4,0)
EndTable

'Subroutine to measure Zero, Called on first pass or when user sets Flag(2)low
```

```
Sub Zero
      Count = 0                                    'Reset Count
      Scan(10,mSec,0,100)                          'Scan 100 times
           BrFull(ZStrain,1,mV50,5,1,6,7,1,5000,1,0,0,100,1,0)
           Count = Count + 1                       'Increment Counter used By
                                                   DataTable
           CallTable ZERO_1                        'Zero_1 outputs on last scan
                                                   (Count=100)
      Next Scan
      ZStrain = ZERO_1.ZStrain_Avg(1,1)            'Set ZStrain = averaged
                                                   value
      Flag(2) = True
End Sub

BeginProg
      Scan(10,mSec,0,0)                                'Scan 10(mSecs)
           If Not Flag(2) Then Zero
           BrFull(StMeas,1,mV50,5,1,6,7,1,5000,1,0,0,100,1,0)
           StrainCalc(Strain,1,StMeas,ZStrain,-1,2,0)
           CallTable STRAINS                       'Strains outputs only when
Flag(1)=True
      Next Scan
EndProg
```

Rnd

Returns a random number.

Syntax
Rnd [**(***number***)**]

Remarks
The argument *number* can be any valid numeric expression.

The **Rnd** function returns a Single value less than 1 but greater than or equal to 0.

The value of *number* determines how **Rnd** generates a random number:

Value of number	Number returned
< 0	The same number every time, as determined by number.
> 0	The next random number in the sequence.
= 0	The number most recently generated.
number omitted	The next random number in the sequence.

The same random-number sequence is generated each time the instruction is encountered because each successive call to the **Rnd** function uses the previous random number as a seed for the next number in the random-number sequence.

To have the program generate a different random-number sequence each time it is run, use the Randomize statement without an argument to initialize the random-number generator before **Rnd** is called.

To produce random integers in a given range, use this formula:

Int((upperbound - lowerbound + 1) * **Rnd** + lowerbound)

Here, upperbound is the highest number in the range, and lowerbound is the lowest number in the range.

Rnd Function Example
The example uses the Rnd function to generate random integer values from 1 to 9. Each time this program is run, Randomize generates a new random-number sequence.

```
Dim Wild1, Wild2                'Declare variables.
Randomize                       'Seed random number generator.
Wild1 = Int(9 * Rnd + 1)        'Generate first random value.
Wild2 = Int(9 * Rnd + 1)        'Generate second random value.
```

Sgn (Source)

Used to find the sign value of a number.

Syntax
x = Sgn (*source***)**

Remarks
Returns an integer indicating the sign of a number.

The argument number can be any valid numeric expression. Its sign determines the value returned by the Sgn function:

If X > 0, then Sgn(X) = 1.

If X = 0, then Sgn(X) = 0.

If X < 0, then Sgn(X) = -1.

Sgn Function Example

The example uses Sgn to determine the sign of a number.

```
Dim Msg, Number                 'Declare variables.
Number = Volt(1)                'Get user input.
Select Case Sgn(Number)         'Evaluate Number.
    Case 0                      'Zero.
        Msg = 0
    Case 1                      'Positive.
        Msg = 1
    Case -1                     'Negative.
        Msg = -1
End Select
```

Sin (Source)

Returns the sine of an angle.

Syntax
x = Sin (*source*)

Remarks
Source can be any valid numeric expression measured in radians.

The **Sin** function takes an *angle* and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite the angle divided by the length of the hypotenuse.

The result lies in the range -1 to 1.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Returns the sine of the value in parentheses. The input must be in radians.

Sin Function Example

The example uses Sin to calculate the sine of an angle from a Volt input.

```
Dim Degrees, Pi, Radians, Ans      'Declare variables.
Pi = 4 * Atn(1)                    'Calculate Pi.
Degrees = Volt(1)                  'Get input.
Radians = Degrees * (Pi / 180)     'Convert to radians.
Ans = Sin(Radians)                 ‘The Sine of Degrees.
```

SinH (Source)

The SINH function returns the hyperbolic sine of an expression or value.

Syntax
x = SINH(Expr)

Remarks
The SINH function returns the hyperbolic sine [SINH(x) = 0.5(ex - e-x)] for the value contained in the Expr argument.

The example uses SINH to calculate the hyperbolic sine of a voltage input.

```
Public Volt1, Ans              'Declare variables.
BeginProg
Scan ( 1, min, 3, 0)
      VoltDiff(Volt1,1,mV5000,1,True,100,500,1,0)
      'Returns voltage on Channel(1) to Volt(1)
      Ans = SINH( Volt1 )  'The Hyperbolic Sine of Volt1.
      NextScan
EndProg
```

Sqr (Source)

Returns the square root of a *number*.

Syntax
x = Sqr (*number***)**

Remarks
The argument *number* can be any valid numeric expression that results in a value greater than or equal to 0.

Returns the square root of the value in parentheses.

Sqr Function Example

The example uses Sqr to calculate the square root of Volt(1) value.

```
Dim Msg, Number                 'Declare variables.
Number = Volt(1)                'Get input.
If Number < 0 Then
Msg = 0                         'Cannot determine the square root of a
                                 negative number.
Else
 Msg = Sqr(Number)
End If
```

StdDevSpa

Used to find the standard deviation of an array.

Syntax
StdDevSpa(Dest, Swath, Source**)**

Remarks
Spatial standard deviation.

$$Dest = \left(\left(\sum_{i=j}^{i=j+swath} X(i)^2 - \left(\sum_{i=j}^{i=j+swath} X(i) \right)^2 / \, swath \right) / \, swath \right)^{\frac{1}{2}}$$

Where *X(j)* = Source

Parameter *& Data Type*	**Enter**
Dest *Variable or Array*	The variable in which to store the results of the instruction.
Swath *Constant*	The number of values of the array over which to perform the specified operation.
Source *Array*	The name of the variable array that is the input for the instruction.

Tan (Source)

Returns the tangent of an angle.

Syntax
x = Tan (*source***)**

Remarks
Source can be any valid numeric expression measured in radians.

Tan takes an *angle* and returns the ratio of two sides of a right triangle. The ratio is the length of the side opposite an angle divided by the length of the side adjacent to the angle.

To convert degrees to radians, multiply degrees by π/180. To convert radians to degrees, multiply radians by 180/π.

Tan Function Example

The example uses Tan to calculate the tangent of an angle from a Volt(1) input.

```
Dim Degrees, Pi, Radians, Ans          'Declare variables.
Pi = 4 * Atn(1)                        'Calculate Pi.
Degrees = Volt(1)                      'Get user input.
Radians = Degrees * (Pi / 180)         'Convert to radians.
Ans = Tan(Radians)                     'The Tangent of Degrees.
```

TANH (Source)

The TANH function returns the hyperbolic tangent of an expression or value.

Syntax
x = TANH *(Source)*

Remarks
The TANH function returns the hyperbolic tangent [tanh(x) = sinh(x)/cosh(h)] for the value defined in Source.

TANH Function Example

The example uses TANH to calculate the hyperbolic tangent of a voltage input.

```
Public Volt1, Ans                  'Declare variables.
VoltDiff(Volt1,1,mV5000,1,True,100,500,1,0)
'Returns voltage on Channel(1) to Volt(1)
Ans = TANH( Volt1 )                'The Hyperbolic Tangent of Volt1.
```

XOR

The XOR function is used to perform a logical exclusion on two expressions.

Syntax
result = expr1 XOR expr2

Remarks
If only one of the expressions evaluates True, result is True. If either expression is a Null, result is also a Null. When neither expression is a Null, result is determined according to the following table:

If expr1 is	And expr2 is	The result is
True	True	False
True	False	True
False	True	True
False	False	False

The XOR operator also performs a bit-wise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following truth table:

If bit in expr1 is	And bit in expr2 is	The result is
0	0	0
0	1	1
1	0	1
1	1	0

XOR Operator Example

The example sets the variable Msg based on the value of variables A, B, and C, assuming that no variable is a Null. If A = 10, B = 8, and C = 11, the left expression is True and the right expression is False. Because only one comparison expression is True, the XOR expression evaluates True.

```
Dim A, B, C                    'Declare variables.
A = 10: B = 8: C = 11          'Assign values.
If A > B XOR B > C Then         'Evaluate expressions.
      Msg = True
Else
      Msg = False.
End If
```

Derived Math Functions

The following is a list of nonintrinsic mathematical functions that can be derived from the intrinsic math functions provided with CRBasic:

Function	CRBasic equivalent
Secant	Sec = 1 / Cos(X)
Cosecant	Cosec = 1 / Sin(X)
Cotangent	Cotan = 1 / Tan(X)
Inverse Secant	Arcsec = Atn(X / Sqr(X * X - 1)) + Sgn(Sgn(X) -1) * 1.5708
Inverse Cosecant	Arccosec = Atn(X/Sqr(X * X - 1)) + (Sgn(X) - 1) * 1.5708
Inverse Cotangent	Arccotan = Atn(X) + 1.5708
Hyperbolic Secant	HSec = 2 / (Exp(X) + Exp(-X))
Hyperbolic Cosecant	HCosec = 2 / (Exp(X) - Exp(-X))
Hyperbolic Cotangent	HCotan = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X))
Inverse Hyperbolic Sine	HArcsin = Log(X + Sqr(X * X + 1))
Inverse Hyperbolic Cosine	HArccos = Log(X + Sqr(X * X - 1))
Inverse Hyperbolic Tangent	HArctan = Log((1 + X) / (1 - X)) / 2
Inverse Hyperbolic Secant	HArcsec = Log((Sqr(-X * X + 1) + 1) / X)
Inverse Hyperbolic Cosecant	HArccosec = Log((Sgn(X) * Sqr(X * X + 1) +1) / X)
Inverse Hyperbolic Cotangent	HArccotan = Log((X + 1) / (X - 1)) / 2
Logarithm	LogN = Log(X) / Log(N)

Section 9. Program Control Instructions

BeginProg, EndProg

The BeginProg instruction is used to mark the beginning of a program. EndProg marks the end of a program.

Syntax

```
BeginProg
  ...
  ...
EndProg
```

Remarks

All of the instructions for the main program fall between the BeginProg and EndProg statements. Program Variables, DataTables, and Subroutines must be defined before the main program. The BeginProg/EndProg statements are not necessary if there are no subroutines or DataTables.

BeginProg Example

The following code shows the layout of a typical datalogger program and the use of the BeginProg/EndProg statements. Program variables and the DataTable are defined, followed by the code for the main program.

```
'Define Variables for WindSpeed and Rain
'Dimension the RealTime array
PUBLIC WINDSP
PUBLIC RAIN
DIM TIME(9)
ALIAS TIME(1)=YEAR
ALIAS TIME(2)=MONTH
ALIAS TIME(3)=DAY
ALIAS TIME(4)=HOUR
ALIAS TIME(5)=MINUTES
ALIAS TIME(6)=SECONDS
ALIAS TIME(7)=mSECONDS
ALIAS TIME(8)=DAY_OF_WEEK
ALIAS TIME(9)=DAY_OF_YEAR

'Define the DataTable, METDATA
DataTable (METDATA,1,1000)
      DataInterval (0,1,Min,10)
      Sample (1,WINDSP,FP2)
      Totalize (1,RAIN,FP2,False )
EndTable

'Main program - Read datalogger real-time clock
'Measure 2 pulse count channels and Call DataTable
BeginProg
```

```
    Scan (1,Sec,3,0)
    RealTime (TIME)
    PulseCount (WINDSP,1,1 ,1,1,1.0,0)
    PulseCount (RAIN,1,2,2,0,1.0,0)
    CallTable METDATA
    NextScan
EndProg
```

Call

The Call statement is used to transfer program control from the main program to a subroutine.

Syntax
Call Name(list of variables)

Remarks
Use of the Call keyword when calling a subroutine is optional.

The Call statement has these parts:

Call	Call is an optional keyword used to transfer program control to a subroutine.
Name	The Name parameter is the name of the subroutine to call.
List of Variables or Constants	The list may contain variables, constants, or expressions that evaluate to a constant (i.e., do not contain a variable) that should be passed into the variables declared in the subroutine. Values of variables passed can be altered by the subroutine. If the subroutine changes the value of the subroutine declared variable, it changes the value in the variable that was passed in. If a constant is passed to one of the subroutine declared "variables", that "variable" becomes a constant and its value cannot be changed by the subroutine.

Call Statement Example
See Sub description in Section 5.

CallTable

Used to call a data table.

Syntax
CallTable Name

Remarks
CallTable is used in the main program to call a DataTable. DataTables are listed in the declaration section of the program prior to BeginProg. When the

DataTable is called, it will process data as programmed and check the output condition.

CallTable Example

This example uses CallTable to call the ACCEL table.

CallTable ACCEL

Data, Read, Restore

Used to mark the beginning of a data list.

Syntax
Data *list* of constants

Read [VarExpr]

Restore

Remarks
Data function: A *list* of floating point constants that can be read (using **Read**) into an Array Variable.

Parameter: A *list* of floating point constants.

Reads **Data** from **Data** declaration into an array. Subsequent **Read** picks up where current **Read** leaves off.

Parameter: Variable destination.

Restore pointer to **Data** to beginning. Used in conjunction with **Data** and **Read**.

Data Statement Example

This example uses Data to hold the data values and Read to transfer the values to variables.

```
Data 1, 2, 3, 4, 5                    'data for x
Data 6, 7, 8, 9, 10                   'data for y
For I = 1 To 5
   Read x( I )
Next I
For I = 1 To 5
   Read y( I )
Next I
```

This next example uses Restore to read 1, 2, 3, 4 into both X( ) and Y( ) variables.

```
Data 1, 2, 3, 4
For I = 1 To 4
   Read X( I )
Next I
Restore
For I = 1 To 4
   Read Y( I )
Next I
```

ClockSet (Source)

Sets the CR5000 clock from the values in an array. The most likely use for this is where the CR5000 can input the time from a more accurate clock than

its own (e.g., a GPS receiver). The input time would periodically or conditionally be converted into the required variable array and ClockSet would be used to set the CR5000 clock.

Source *Array*	The source must be a seven element array . *array*(1)..*array*(7) should hold respectively year, month, day, hours, minutes, seconds, and microseconds..

Delay (Option, Delay, Units)

Used to delay the program.

Syntax
Delay Option
Delay(Delay, Units**)**

Remarks
The Delay instruction is used to delay the measurement task sequence or the processing instructions for the time period specified by the Delay and Units arguments, before progressing to the next measurement or processing instruction.

The Scan Interval should be sufficiently long to process all measurements plus the delay period. If the delay is applied to the measurement task sequence and the scan interval is not long enough to process all measurements plus the delay, the program will not compile when downloaded to the datalogger. If the delay is applied to the processing task sequence, the program will compile but scans will be skipped.

<table>
<tr><td>Parameter
& Data Type</td><td colspan="3">Enter</td></tr>
<tr><td>DelayOption
Constant</td><td colspan="3"><table><tr><th>Code</th><th>Result</th></tr><tr><td>0</td><td>Delay will affect the measurement task sequence. Processing will continue to take place as needed in the background. When this option is chosen, the Delay instruction must not be placed in a conditional statement.</td></tr><tr><td>1</td><td>Delay will affect processing. Measurements will continue as called for by the task sequencer.</td></tr></table></td></tr>
<tr><td>Delay
Constant</td><td colspan="3">The numeric value for the time delay.</td></tr>
<tr><td>Units
Constant</td><td colspan="3">The units for the delay.<table><tr><th>Alpha Code</th><th>Numeric Code</th><th>Units</th></tr><tr><td>USEC</td><td>0</td><td>microseconds</td></tr><tr><td>MSEC</td><td>1</td><td>milliseconds</td></tr><tr><td>SEC</td><td>2</td><td>seconds</td></tr><tr><td>MIN</td><td>3</td><td>minutes</td></tr></table></td></tr>
</table>

Do

Repeats a block of statements while a condition is true or until a condition becomes true.

Syntax 1
Do [{**While** | **Until**} *condition*]
[*statementblock*]
[**Exit Do**]
[*statementblock*]
Loop

Syntax 2
Do
[*statementblock*]
[**Exit Do**]
[*statementblock*]
Loop [{**While** | **Until**} *condition*]

The **Do...Loop** statement has these parts:

Part	Description
Do	Must be the first statement in a **Do...Loop** control structure.
While	Indicates that the loop is executed while *condition* is true.
Until	Indicates that the loop is executed until *condition* is true.
condition	Numeric expression that evaluates true (nonzero) or false (0 or Null).
statementblock	Program lines between the **Do** and **Loop** statements that are repeated while or until *condition* is true.
Exit Do	Only used within a **Do...Loop** control structure to provide an alternate way to exit a **Do...Loop**. Any number of **Exit Do** statements may be placed anywhere in the **Do...Loop**. Often used with the evaluation of some condition (for example, If...Then), **Exit Do** transfers control to the statement immediately following the **Loop**. When **Do...Loop** statements are nested, control is transferred to the **Do...Loop** that is one nested level above the loop in which the **Exit Do** occurs.
Loop	Ends a **Do...Loop**.

Do...Loop Statement Example

The example creates an infinite Do...Loop that can be exited only if Volt(1) is within a range.

```
Dim Reply                              'Declare variable.
Do
    Reply = Volt(1)
    If Reply > 1 And Reply < 9 Then    'Check range.
        Exit Do                        'Exit Do Loop.
    End If
Loop
```

Alternatively, the same thing can be accomplished by incorporating the range test in the Do...Loop as follows:

```
Dim Reply                       'Declare variable.
Do
    Reply = Volt(1)
Loop Until Reply > 1 And Reply < 9
The next example show the use of Wend.
While X > Y                     'Old fashioned way of looping.
  .........
  .........
Wend

Do While X > Y                  'Much better
  .........
  .........
Loop
```

FileManage

The FileManage instruction is used to manage files from within a running datalogger program.

Syntax
FileManage("Device: FileName", Attribute)

Remarks
FileManage is a function that allows the active datalogger program to manipulate program files that are stored in the datalogger.

The FileManage instruction has the following parameters:

<table>
<tr><th>Parameter
& Data Type</th><th colspan="3">Enter</th></tr>
<tr><td>Device;
Filename
Text</td><td colspan="3">The "Device:Filename" argument is the file that should be manipulated. The Device on which the file is stored must be specified and the entire string must be enclosed in quotation marks. Device = CPU, the file is stored in datalogger memory. Device = CRD, the file is stored on a PCMCIA card..</td></tr>
<tr><td>Attribute
Constant</td><td colspan="3">The Attribute is a numeric code to determine what should happen to the file affected by the FileManage instruction. The Attribute codes are actually a bit field. The codes are as follows:</td></tr>
<tr><td></td><td>Bit</td><td>Decimal</td><td>Description</td></tr>
<tr><td></td><td>bit 0</td><td>1</td><td>Program not active</td></tr>
<tr><td></td><td>bit 1</td><td>2</td><td>Run on power up</td></tr>
<tr><td></td><td>bit 2</td><td>4</td><td>Run now</td></tr>
<tr><td></td><td>bits 1 & 2</td><td>6</td><td>Run now and on power up</td></tr>
<tr><td></td><td>bit 3</td><td>8</td><td>Delete</td></tr>
<tr><td></td><td>bit 4</td><td>16</td><td>Delete all</td></tr>
</table>

FileManage Example
The statement below uses FileManage to run TEMPS.CR5, which is stored on the datalogger's CPU, when Flag(2) becomes high.

```
If Flag(2) then FileManage( "CPU:TEMPS.CR5" 4 )   '4 means Run Now
```

FileMark (TableName)

Parameter *& Data Type*	**Enter**
TableName *name*	The name of the data table in which to insert the filemark..

FileMark is used to insert a filemark into a data file. The filemark can be used by the decoding software to indicate that a new file should be started at the mark. This capability to create multiple files only exists in the binary to ASCII converter. To make use of it files must be stored to a PCMCIA card and retrieved from the logger files screen or by removing the card and transferring the file directly to the computer.

FileMark is placed within a conditional statement in order to write the filemark at the desired time.

For ... Next Statement

Repeats a group of instructions a specified number of times.

Syntax
For *counter* = *start* **To** *end* [**Step** *increment*]
[*statementblock*]
[**Exit For**]
[*statementblock*]
Next [*counter* [, *counter*][, ...]]

The **For...Next** statement has these parts:

Part	**Description**
For	Begins a **For...Next** loop control structure. Must appear before any other part of the structure.
counter	Numeric variable used as the loop counter. The variable cannot be an array element or a record element.
start	Initial value of *counter*.
To	Separates *start* and *end* values.
end	Final value of *counter*.
Step	Indicates that *increment* is explicitly stated.
increment	Amount *counter* is changed each time through the loop. If you do not specify **Step**, *increment* defaults to one.
statementblock	Program lines between **For** and **Next** that are executed the specified number of times.

Exit For Only used within a **For...Next** control structure to provide an alternate way to exit. Any number of **Exit For** statements may be placed anywhere in the **For...Next** loop. Often used with the evaluation of some condition (for example, If...Then), **Exit For** transfers control to the statement immediately following the **Next**.

Next Ends a **For...Next** loop. Causes *increment* to be added to *counter*.

The *Step* value controls loop execution as follows:

When *Step* is	Loop executes if
Positive or 0	*counter* <= *end*
Negative	*counter* >= *end*

Once the loop has been entered and all the statements in the loop have executed, *Step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute in the first place), or the loop is exited and execution continues with the statement following the **Next** statement.

Tip Changing the value of *counter* while inside a loop can make the program more difficult to read and debug.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its *counter*. The following construction is correct:

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
            ...
        Next K
    Next J
Next I
```

Note If you omit the variable in a **Next** statement, the value of **Step** increment is added to the variable associated with the most recent **For** statement. If a **Next** statement is encountered before its corresponding **For** statement, an error occurs.

For...Next Statement Example

The example runs one For..Next loop inside another.

```
Dim I, J  'Declare variables.
For J = 5 To 1 Step -1      'Loop 5 times backwards.
    For I = 1 To 12                 'Loop 12 times.
        . . . .                     'Run some code.
    Next I
    . . . .                         'Run some code.
Next J
. . . .                             'Run some code.
```

This next example fills odd elements of X up to 40 * Y with odd numbers.

```
For I = 1 To 40 * Y Step 2
  X( I ) = I
Next I
```

If ... Then ... Else Statement

Allows conditional execution, based on the evaluation of an expression.

Syntax 1
If *condition* **Then** *thenpart* [**Else** *elsepart*]

Syntax 2
If *condition1* **Then**
[*statementblock-1*]
[**ElseIf** *condition2* **Then**
[*statementblock-2*]]
[**Else**
[*statementblock-n*]]
End If

Syntax 1 Description
The single-line form is often useful for short, simple conditional tests. Syntax 1 has these parts:

Part	Description
If	Begins the simple **If...Then** control structure.
condition	An expression that evaluates true (nonzero) or false (0 and Null).
Then	Identifies actions to be taken if *condition* is satisfied.
thenpart	Statements or branches performed when *condition* is true.
Else	Identifies actions taken if *condition* is not satisfied. If the **Else** clause is not present, control passes to the next statement in the program.
elsepart	Statements or branches performed when *condition* is false.

The *thenpart* and the *elsepart* fields both have this syntax:

{statements | [GoTo] linenumber | GoTo linelabel }

The *thenpart* and *elsepart* syntax has these parts:

Part	Description
statements	One or more CRBasic statements, separated by colons.
Note	You can have multiple statements with a *condition*, but they must be on the same line and separated by colons, as in the following statement: **If** A > 10 **Then** A = A + 1 : B = B + A : C = C + B

Syntax 2 Description

The block form of **If...Then...Else** provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug. Syntax 2 has these parts:

Part	Description
If	Keyword that begins the block **If...Then** decision control structure.
condition1	Same as *condition* used in the single-line form shown above.
Then	Keyword used to identify the actions to be taken if a condition is satisfied.
statementblock-1	One or more CRBasic statements executed if *condition1* is true.
ElseIf	Keyword indicating that alternative conditions must be evaluated if *condition1* is not satisfied.
condition2	Same as *condition* used in the single-line form shown above.
statementblock-2	One or more CRBasic statements executed if *condition2* is true.
Else	Keyword used to identify the actions taken if none of the previous conditions are satisfied.
statementblock-n	One or more CRBasic statements executed if *condition1* and *condition2* are both false.
End If	Keyword that ends the block form of the **If...Then**.

In executing a block If, CRBasic tests *condition1*, the first numeric expression. If the expression is true, the statements following **Then** are executed.

If the first expression is false, CRBasic begins evaluating each **ElseIf** condition in turn. When CRBasic finds a true condition, the statements immediately following the associated **Then** are executed. If none of the **ElseIf** conditions is true, the statements following the **Else** are executed. After executing the statements following **Then** or **Else**, the program continues with the statement following **End If**.

The **Else** and **ElseIf** clauses are both optional. You can have as many **ElseIf** clauses as you like in a block **If**, but none can appear after an **Else** clause. Any of the statement blocks can contain nested block **If** statements.

CRBasic looks at what appears after the **Then** keyword to determine whether or not an **If** statement is a block **If**. If anything other than a comment appears after **Then**, the statement is treated as a single-line If statement.

A block **If** statement must be the first statement on a line. The **Else**, **ElseIf**, and **End If** parts of the statement can have nothing but spaces in front of them. The block **If** must end with an **End If** statement.

For Example

```
If a > 1 And a <= 100 Then

    ...

ElseIf a = 200 Then

    ...

End If
```

Tip Select Case may be more useful when evaluating a single expression that has several possible actions.

If...Then ... Else Statement Example

The example illustrates the various forms of the If...Then...Else syntax.

```
Dim X, Y, Temp( 5 )              'Declare variables.
X = Temp( 1 )
If X < 10 Then
    Y = 1                        '1 digit.
ElseIf X < 100 Then
    Y = 2                        '2 digits.
Else
    Y = 3                        '3 digits.
End If
. . . .                          'Run some code
. . . .                          'Run some code
```

Power Off

Used to turn the CR5000 off until a designated time.

Syntax
PowerOff(StartTime, Interval, Units**)**

Parameter *& Data Type*	**Enter**
Start Time *Array*	The name of a six element array that contains the start time: Year, month, day, hour, minutes, and seconds, respectively.
Interval *Constant*	Enter the time interval on which the CR5000 is to be powered up.
Units *Constant*	The units for the time parameters.

Alpha Code	**Numeric Code**	**Units**
SEC	2	seconds
MIN	3	minutes
HR	4	hours
DAY	5	days

Remarks
This instruction sets a time to power up and then shuts off CR5000 power. Only the clock continues running while the CR5000 is powered down. When the time to power up arrives, the power is restored, the CR5000 reloads its program from Flash memory and begins running.

The interval allows the CR5000 to periodically power up and execute a program. StartTime is a time value. If StartTime is in the future when PowerOff is executed, it is the time the CR5000 will be programmed to power up. If StartTime is in the past when PowerOff is executed, The CR5000 will set the time to power up to the next occurrence of the interval (using StartTime as the start of the first interval)

The units for the interval are days, hours, minutes, or seconds.

When the CR5000 is in this power off state the ON Off switch is in the on position just like when the CR5000 is turned off from the keyboard's configure menu. The CR5000 will wake up if a key is pressed on the keyboard or in response to communication on the CSI/O or RS232 ports.

Power off can also be used in conjunction with the power-up digital inputs to set up the CR5000 to power up in response to external trigger, make a series of measurements, and then power off.

If the "<0.5 " input is switched to ground or if the ">2" input has a voltage greater than 2 volts applied, the CR5000 will awake, load the program in memory and run. If the "< 0.5" input is held at ground and off/on switch turned off then on, the CR5000 will not run the program in memory. This is extremely useful if the program executes the PowerOff instruction immediately or after a short measurement period.

The following example is a good one to play with to become familiar with the PowerOff instruction. The CR5000 "scans" once a second for two minutes. At the end of that time it powers down. It is programmed to wake up on a 4 minute interval. After the first PowerOff, it will wake up every four minutes, count for 2 minutes and turn itself off. You can load this program and use the Power On inputs to wake the CR5000 before the interval is up. A program for an actual application would have measurements within the scan.

```
Public Start(6), count          'Declare the start time array and count
    'Start() is initialized to 0 at compile time.  0 time is Midnight the start of 1990
    'count is initialized to 0 at compile time
BeginProg
    Scan(1,SEC,0,120)       'Scan once per second for 2 minutes
        Count=count+1       'Increment counter
    NextScan
    PowerOff(Start,4,min)   'Power off, wake up on 4 minute interval
EndProg
```

Print *list of variables or quoted text*

Print is used as a tool in debugging a program to print text or the value of variables at different points in the program. "Printing" occurs over the active link and can be observed from Tools | Diagnostics | Terminal Mode in PC9000.

RunDLDFile

The RunDLDFile instruction is used to run a datalogger program file from the active program file.

Syntax
RunDLDFile("Device:FileName", Attrib)

Remarks
The RunDLDFile has the following parameters:

"Device:FileName"	The "Device:Filename" argument is the file that should be executed. The Device on which the file is stored must be specified and the entire string must be enclosed in quotation marks. Device = CPU, the file is stored in datalogger memory. Device = CRD, the file is stored on a PCMCIA card.
Attribute	The Attribute is a numeric code to determine what should happen to the file called by the RunDLDFile

instruction. The Attribute codes are actually a bit field. The codes are as follows:

Bit	Decimal	Description
bit 1	2	Run on power up
bit 2	4	Run now

RunDLDFile Example

The statement below uses RunDLDFile to run TEMPS.DLD, which is stored on the datalogger's CPU, when Flag(2) becomes high.

```
If Flag(2) then RunDLDFile( "CPU:TEMPS.DLD" 4 )      '4 means Run Now
```

Reset Table

Used to reset a data table under program control.

Syntax

ResetTable(TableName **)**

Remarks

ResetTable is a function that allows a running program to erase and restart a data table. TableName is the name of the table to reset.

ResetTable Example

The example program line uses ResetTable to reset table MAIN when Flag(2) is high.

```
If Flag(2) then ResetTable( MAIN )          'resets table MAIN
```

Scan

Used to establish the program scan rate.

Syntax
Scan(Interval, Units, Option, Count**)**
...
...**[Exit Scan]**
...
Next Scan

The measurements, processing, and calls to output tables bracketed by the Scan…NextScan instructions determine the sequence and timing of the datalogger program.

The Scan instruction determines how frequently the measurements within the Scan…NextScan structure are made, controls the buffering capabilities, and sets the number of times to loop through the scan.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter</td></tr>
<tr><td>Interval
Constant</td><td>Enter the time interval at which the scan is to be executed. The interval may be in µs, ms, s, or minutes, whichever is selected with the Units parameter. The maximum scan interval is one minute.</td></tr>
<tr><td>Units
Constant</td><td>The units for the time parameters.
<table>
<tr><th>Alpha Code</th><th>Numeric Code</th><th>Units</th></tr>
<tr><td>USEC</td><td>0</td><td>microseconds</td></tr>
<tr><td>MSEC</td><td>1</td><td>milliseconds</td></tr>
<tr><td>SEC</td><td>2</td><td>seconds</td></tr>
<tr><td>MIN</td><td>3</td><td>minutes</td></tr>
</table></td></tr>
<tr><td>Option
Constant</td><td>The Option parameter determines how data will be buffered during the Scan…NextScan process. The options are:
<table>
<tr><th>Option</th><th>Result</th></tr>
<tr><td>0, 1, or 2</td><td>The datalogger uses two buffers when processing measurements. When a measurement begins on a scan, the values of the previous scan are loaded into a buffer. This allows processing to finish on the previous scan during measurement of the current scan.</td></tr>
<tr><td>>3</td><td>The datalogger uses three or more buffers when processing measurements, based on the number of scans defined by this Constant.</td></tr>
</table>
Larger buffers can be used for a Scan that has occasional large processing requirements such as FFTs or Histograms, and/or when processing may be interrupted by communications. If a value of 1000 is inserted into the BufferSize argument of a scan having 10 thermocouple measurements, 40,000 bytes of SRAM will be allocated for the buffer [(4 bytes) / (measurement) x (10 measurements)/(buffered scan) x 1000 buffered scans)]. The buffer size plus the size of any Output Tables stored in SRAM should not exceed 2 megabytes.

If the processing ever lags behind by more than the buffer allocated, the datalogger will discard the buffered values and synchronize back up to the current measurement

The SlowSequence instruction does not allow for this buffering scheme even though Scan is used to signify the start of a scan in a slow sequence. In SlowSequence, the measurements are stored in a single buffer. Processing of this buffer is completed before the NextScan measurements are made.</td></tr>
<tr><td>Count
Integer</td><td>The number of times to execute the Scan/NextScan loop. Enter 0 for infinite looping.</td></tr>
</table>

Select Case Statement

Executes one of several statement blocks depending on the value of an expression.

Syntax
Select Case *testexpression*
[**Case** *expressionlist1*
    [*statementblock-1*]]
[**Case** *expressionlist2*
    [*statementblock-2*]]
[**Case Else**
    [*statementblock-n*]]
End Select

The Select Case syntax has these parts:

Part	Description
Select Case	Begins the **Select Case** decision control structure. Must appear before any other part of the **Select Case** structure.
testexpression	Any numeric or string expression. If *testexpression* matches the *expressionlist* associated with a **Case** clause, the *statementblock* following that **Case** clause is executed up to the next **Case** clause, or for the final one, up to the **End Select**. Control then passes to the statement following **End Select**. If *testexpression* matches more than one **Case** clause, only the statements following the first match are executed.
Case	Sets apart a group of CRBasic statements to be executed if an expression in *expressionlist* matches *testexpression*.
expressionlist	The *expressionlist* consists of a comma-delimited list of one or more of the following forms. expression expression To expression Is compare-operator expression statementblock Elements *statementblock-1* to *statementblock-n* consist of any number of CRBasic statements on one or more lines.
Case Else	Keyword indicating the *statementblock* to be executed if no match is found between the *testexpression* and an *expressionlist* in any of the other **Case** selections. When there is no **Case Else** statement and no expression listed in the **Case** clauses matches *testexpression*, program execution continues at the statement following **End Select**.
End Select	Ends the **Select Case**. Must appear after all other statements in the **Select Case** control structure.

The argument expressionlist has these parts:

Part	Description
expression	Any numeric expression.
To	Keyword used to specify a range of values. If you use the To keyword to indicate a range of values, the smaller value must precede To.

Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen *testexpression* values.

You can use multiple expressions or ranges in each **Case** clause. For example, the following line is valid:

```
Case 1 To 4, 7 To 9, 11, 13
```

Select Case statements can be nested. Each **Select Case** statement must have a matching **End Select** statement.

Select Case Statement Example

The example uses Select Case to decide what action to take based on user input.

```
Dim X, Y                        'Declare variables.
If Not X = Y Then               'Are they equal
    If X > Y Then
        Select Case X           'What is X.
        Case 0 To 9             'Must be less than 10.
            . . . .             'Run some code.
            . . . .             'Run some code.
        Case 10 To 99           'Must be less than 100.
            . . . .             'Run some code.
            . . . .             'Run some code.
        Case Else               'Must be something else.
            . . . .             'Run some code.
        End Select
    End If
Else
    Select Case Y               'What is Y.
    Case 1, 3, 5, 7, 9          'It's odd.
        . . . .                 'Run some code.
    Case 0, 2, 4, 6, 8          'It's even.
        . . . .                 'Run some code.
    Case Else                   'Out of range.
        . . . .                 'Run some code.
        . . . .                 'Run some code.
    End Select
End If
. . . .                         'Run some code.
. . . .                         'Run some code.
```

Sleep

The Sleep instruction is used to put the datalogger in a quiescent mode between program scans.

Syntax
Sleep

Remarks
This instruction allows the datalogger to go into its lowest current state between program scans. The Sleep instruction affects only the scan in which it resides.

Sleep Program Example
The following program has two separate scans. One scan uses the Sleep instruction to put the datalogger in low power mode between scans.

```
DIM DiffVolt1                    'Declare variable for diff measurement
DIM SeVolt2                      'Declare variable for se measurement

BeginProg
   Scan (1,Sec,3,0)              'Scan interval = 1 second
      VoltDiff (DiffVolt1,1,mV5000,1,True ,100,250,1.0,0)     'Scan diff channel
      NextScan
   Scan (10,Min,3,0)             'Scan interval = 10 minutes
      Sleep                      'Low power between scans
      VoltSe (SeVolt2,1,mV5000,1,1,100,250,1.0,0)         'Scan se channel
      NextScan
EndProg
```

Slow Sequence

The SlowSequence instruction allows a group of measurements to be executed at a rate slower than the main program.

Syntax
SlowSequence

Remarks
The SlowSequence statement marks the end of the main program and begins a low priority program. The instructions for this program are executed when the main program is not running as time allows.

It is possible to have up to four different SlowSequence scans for measurements that are not needed at the rate of the primary scan interval. The datalogger tags on measurement instructions from the SlowSequence scans to the normal scan as time allows. At least one rep of a measurement instruction from the SlowSequence scan is added to each normal scan (the requested settling time occurs before the measurement). Thus, the primary scan interval must be long enough to make the primary scan measurements plus the longest single measurement (including settling time) from the scan in the SlowSequence. In the case where the primary scan interval is only long enough to allow one measurement fragment from the SlowSequence per primary scan,

the minimum time for the SlowSequence scan interval is the product of the number of SlowSequence measurement segments and the primary scan interval. One function of a SlowSequence is to update the calibration table.

The measurements in a single "scan" of the SlowSequence may be spread out over a longer time period because the measurements can be parceled into multiple primary scans.

When more than one SlowSequence is used in a program, certain combinations of the main program scan rate and the SlowSequence scan rates may result in a lower priority SlowSequence never being executed. If the interval for a higher priority SlowSequence arrives before the lower priority SlowSequence can be sliced in, the lower priority SlowSequence will never be performed. Each combination of main program and SlowSequence program segments must be evaluated to determine if there is sufficient time to allow lower priority scans to be executed.

Low priority DataTables can be included in the SlowSequence scan by listing them after the SlowSequence instruction. It should be noted that time stamped data written to SlowSequence DataTables will be stamped with the start time of the last SlowSequence scan.

SlowSequence Example
The example uses SlowSequence to calibrate the datalogger every sixty seconds.

```
'Calibrate

Public Temp1
Public Calib1(60)

DataTable(Table1,1,600)
    DataInterval(0,1,sec,1)
    Sample(1,Temp1,FP2)
    EndTable

BeginProg
    Scan(20,mSec,0,0)
        PanelTemp (Temp1,250)
        CallTable Table1
    Next scan

    SlowSequence
        Scan (60,Sec,0,0)
            Calibrate(Calib1)
    Next scan
EndProg
```

SubScan (SubInterval, Units, SubRatio)

The SubScan instruction is used to control an AM16/32 multiplexer or to measure some analog inputs at a faster rate than the program scan.

Syntax

SubScan (SubInterval, Units, SubRatio)

Measurements and processing

Next SubScan

Remarks

The SubScan/NextSubScan instructions are placed within the Scan/NextScan instructions of a program.

NOTE SubScans cannot be nested or placed in a SlowSequence. Pulse Count or SDM measurements cannot be used within a SubScan.

Parameter *& Data Type*	**Enter**		
SubInterval *Constant*	The time interval between subscans. Enter 0 for no delay between subscans.		
Units *Constant*	The unit of time for the SubInterval.		
	Alpha Code	**Numeric Code**	**Units**
	usec	0	microseconds
	msec	1	milliseconds
	sec	2	Seconds
SubRatio *Constant*	The number of times to loop through the subscan each time the scan runs. The maximum number is 65,535.		

Example program:

```
'CR5000

'Example Program showing the use of the SubScan instruction to measure
'16 100 ohm Platinum Resistance Thermometers connected to an AM16/32 multiplexer
' used in the 4x16 configuration. The program also measures 6 copper constantan
'thermocouples.

'Wiring:
'CR5000            AM16/32              PRT(4 Wires)
'                     Control/Common    Sensor Terminals
'C1  ------------------------- Reset              1H ----------------Excitation
'C2  ------------------------- Clock              1L  ---------------- Excitation Return
'IX1 ------------------------- COM 1H             2H ---------------- Sense wire excitation side
'IXR ------------------------- COM 1L             2L ----------------Sense wire return side
'7H  ------------------------- COM 2H
'7L  -------------------------  COM2L
```

```
'The Thermocouples are connected to differential channels 1-6.

'Declare Variables:
Public TRef, TCTemp(6), PRTResist(16), PRTTemp(16)
Dim I  'Counter for setting Array element to correct value for mux measurement

'Declare Output Table for 15 minute averages:
DataTable (Avg15Min,1,-1)
  DataInterval (0,5,Min,10)
  Average (1,TRef,IEEE4,0)
  Average (6,TCTemp(),IEEE4,0)
  Average (16,PRTTemp(),IEEE4,0)
EndTable

BeginProg
  Scan (1,Sec,3,0)
     PanelTemp (TRef,250)
    TCDiff (TCTemp(),6,mV20C ,1,TypeT,TRef,True ,0,250,1.0,0)
    Portset (1 ,1 )    'Enable Multiplexer
    I=0
    SubScan(0,sec,16)
      'Pulse port (Set High, Delay, Set Low) to clock multiplexer
      Portset (2,1 )
      Delay (0,20,mSec)
      Portset (2,0)
      I=I+1
      'The Resistance measurement measures the PRT resistance:
      Resistance (PRTResist(I),1,mV50,7,Ix1,1,500,True ,True ,0,250,0.01,0)
      'With a multiplier of 0.01 (1/100) the value returned is the Resistance/Resistance @ 0 deg,
      'the required input for the PRT temperature calculation instruction.
    NextSubScan
    PRT (PRTTemp(1),16,PRTResist(1),1.0,0)
    CallTable Avg15Min
  NextScan
EndProg
```

Timer

Used to return the value of a timer.

Remarks

Timer is a function that returns the value of a timer. TimOpt is used to start, stop, reset and start, stop and reset, or read without altering the state (running or stopped). Multiple timers, each identified by a different number (TimNo), may be active at the same time.

Syntax

variable = Timer(1,usec,2)

Parameter *& Data Type*	**Enter**
TimNo *Constant, Variable, or Expression*	An integer number for the timer (e.g., 0, 1, 2, . . .) Use low numbers to conserve memory; using TimNo 100 will allocate space for 100 timers even if it is the only timer in the program.
Units *Constant*	The units in which to return the timer value. **Alpha Code** / **Numeric Code** / **Units** USEC / 0 / microseconds MSEC / 1 / milliseconds SEC / 2 / seconds MIN / 3 / minutes
TimOpt *Constant*	The action on the timer. The timer function returns the value of the timer after the action is performed **Code** / **Result** 0 / start 1 / stop 2 / reset and start 3 / stop and reset 4 / read only

Timer Example
The example uses Timer.

While…Wend

The While…Wend instructions are used to executes a series of statements in a loop as long as a given condition is true.

Syntax
While Condition
 [StatementBlock]
Wend

Remarks
While…Wend loops can be nested.

The While...Wend statement has the following parameters:

While	The While statement begins the While...Wend loop control structure.
Condition	The Condition is any expression that can be evaluated True (nonzero) or False (0 and Null). If Condition is true, all statements in StatementBlock are executed until the Wend statement is encountered. Control then returns to the While statement and Condition is again checked. If Condition is still true, the process is repeated. If Condition it is not True, execution resumes with the statement following the Wend statement.

StatementBlock	The StatementBlock is the portion of the program that should be repeated until the loop is terminated. These instructions lie between the While and Wend statements.
Wend	The Wend statement ends the While...Wend control structure.

NOTE The Do...Loop statement provides more structure and a more flexible way to perform looping.

While...Wend Statement Example
This example creates a While...Wend that is exited only if Reply is within a range.

```
Dim Reply                    'Declare variable.
While Reply < 90
      Reply = Reply + 1
Wend
```

Appendix A. CR5000 Status Table

The CR5000 status table contains current system operating status information that can be accessed from the running CR5000 program or monitored by PC software. There is also a way to view the status information from the keyboard. Table 1 shows the variables in the status table and a brief explanation of each follows.

Table 1. CR5000 Status Table

Name	Type
ROMVersion	String
OSVersion	String
OSItem	Float
OSDate	String
StationName	String
ProgName	String
StartTime	Time
Battery	Float
PanelTemp	Float
LithiumBattery	Integer(Boolean)
CPUSignature	Integer
DLDSignature	Integer
ProgSignature	Integer
PC-CardBytesFree	Integer
MemoryFree	Integer
DLDBytesFree	Integer
ProcessTime	Integer
MaxProcTime	Integer
MeasureTime	Integer
SkippedScan	Integer
SlowProcTime	Integer(array)
MaxSlowProcTime	Integer(array)
LastSlowScan	Integer(array)
SkippedSlowScan	Integer(array)
MeasureOps	Integer
WatchdogErrors	Integer
Low12VCount	Integer
StartUpCode	Integer
CommActive	Integer(Boolean)
ProgErrors	Integer
ErrorCalib	Integer
VarOutOfBound	Integer
SkippedRecord	Integer(array)
SecsPerRecord	Integer(array)
SrlNbr	integer
Rev	integer
CalVolts	Float(array)
CalGain	Float(array)

CalSeOffset	Integer(array)
CalDiffOffset	Integer(array)
CardStatus	string
CompileResults	string

ROMVersion - Version of the ROM code. This value is stored in the ROM and read by the OS at compile time.

OSVersion - Current version of the operating system.

OSItem - The CSI item number for the operating system.

OSDate - Date that the Operating System was compiled.

StationName - String stored as the Station Name of the CR5000.

ProgName - The Name of the currently running program.

StartTime - Time that the program began running.

Battery - Current value of the battery voltage. This measurement is made in the background calibration.

PanelTemp - Current Panel temperature measurement. This measurement is made in the background to temp compensate the display contrast. The measurement is made with a zero integration conversion and minimal settling time and therefore should not be used as a reference temperature for thermocouple measurements.

LithiumBattery - A Boolean variable signaling "True" (-1) if the lithium battery is OK and "False" (0) if not. The lithium battery is loaded and a comparator checked every 4 seconds to verify that the battery is charged.

CPUSignature - The Operating System signature. The value should match the value obtained by running the CSI sig program on the *name*.obj operating system file.

DLDSignature - Signature of the current running program file.

ProgSignature - Signature of the compiled binary data structure for the current program. This value is independent of comments added or non-functional changes to the program file.

PC-CardBytesFree - Gives the number of bytes free on the PC-Card.

MemoryFree - Amount (in bytes) of unallocated memory on the CPU (SRAM). The user may not be able to allocate all of free memory for data tables as final storage must be contiguous. As memory is allocated and freed there may be holes that are unusable for final storage, but that will show up as free bytes.

DLDBytesFree - Amount of free space in the CPU RAM disk that is used to store program files.

ProcessTime - Time in microseconds that it took to run through processing on the last scan. Time is measured from the end of the EndScan instruction (after the measurement event is set) to the beginning of the EndScan (before the wait for the measurement event begins) for the subsequent scan.

MaxProcTime - The maximum time required to run through processing for the current scan. This value is reset when the scan exits.

MeasureTime - The time required by the hardware to make the measurements in this scan. The sum of all integration times and settling times. Processing will occur concurrent with this time so the sum of measure time and process time is not the time required in the scan instruction.

SkippedScan - Number of scans that have occurred while running the current program.

SlowProcTime - Time required to process the current slow scan. If the user has slow scans then this variable becomes an array with a value for the system slow scan and each of the users scans.

MaxSlowProcTime - The maximum Time required to process the current slow scan. If the user has slow scans then this variable becomes an array with a value for the system slow scan and each of the users scans.

LastSlowScan - The last time that this slow scan executed. If the user has slow scans then this variable becomes an array with a value for the system slow scan and each of the users scans.

SkippedSlowScan - The number of scans that have been skipped in this slow sequence. If the user has slow scans then this variable becomes an array with a value for the system slow scan and each of the users scans.

MeasureOps - This is the number of task sequencer opcodes required to do all measurements in the system. This value includes the Calibration opcodes (compile time) and the system slow sequence opcodes.

WatchdogErrors - The number of Watchdog errors that have occurred while running this program. This value can be reset from the keyboard by going to status and scrolling down to the variable and pressing the DEL key. It is also reset upon compiling a new program.

Low12VCount - Keeps a running count of the number of occurrences of the 12VLow signal being asserted. When this condition is detected the logger ceases making measurements and goes into a low power mode until the system voltage is up to a safe level. This value can be reset from the keyboard by going to status and scrolling down to the variable and pressing the DEL key. It is also reset upon compiling a new program.

StartUpCode - A code variable that allows the user to know how the system woke up from poweroff.

CommActive - A variable signaling whether or not communications is currently active (increments each time the autobaud detect code is executed).

ProgErrors - The number of compile (or runtime) errors for the current program.

ErrorCalib - A counter that is incremented each time a bad calibration value is measured. The value is discarded (not included in the filter update) and this var is incremented.

VarOutOfBound - Flags whether an array was accessed out of bounds.

SkippedRecord - Variable that tells how many records have been shipped for a given table. Each table has its own entry in this array.

SecsPerRecord - Output interval for a given table. Each table has its own entry in this array.

SrlNbr - Machine specific serial number. Stored in FLASH memory.

Rev - Hardware revision number. Stored in FLASH memory.

CalVolts - Factory calibration numbers. This array contains twenty values corresponding to the 20 integration / range combinations. These numbers are loaded by the Factory Calibration and are stored in FLASH.

CalGain - Calibration table Gain values. Each integration / range combination has a gain associated with it. These numbers are updated by the background slow sequence if the running program uses the integration / range.

CalSeOffset - Calibration table single ended offset values. Each integration / range combination has a single ended offset associated with it. These numbers are updated by the background slow sequence if the running program uses the integration / range.

CalDiffOffset - Calibration table differential offset values. Each integration / range combination has a differential offset associated with it. These numbers are updated by the background slow sequence if the running program uses the integration / range.

CardStatus - Contains a string with the most recent card status information.

CompileResults - Contains any error messages that were generated by compilation or during run time.

Index

I

L

M

N

O

P

R

S

T

U

V

W