

OPERATOR'S MANUAL



CR3000 Micrologger

Revision: 9/07



Copyright © 2000-2007
Campbell Scientific, Inc.

Warranty and Assistance

The **CR3000 MICROLOGGER** is warranted by CAMPBELL SCIENTIFIC, INC. to be free from defects in materials and workmanship under normal use and service for thirty-six (36) months from date of shipment unless specified otherwise. Batteries have no warranty. CAMPBELL SCIENTIFIC, INC.'s obligation under this warranty is limited to repairing or replacing (at CAMPBELL SCIENTIFIC, INC.'s option) defective products. The customer shall assume all costs of removing, reinstalling, and shipping defective products to CAMPBELL SCIENTIFIC, INC. CAMPBELL SCIENTIFIC, INC. will return such products by surface carrier prepaid. This warranty shall not apply to any CAMPBELL SCIENTIFIC, INC. products which have been subjected to modification, misuse, neglect, accidents of nature, or shipping damage. This warranty is in lieu of all other warranties, expressed or implied, including warranties of merchantability or fitness for a particular purpose. CAMPBELL SCIENTIFIC, INC. is not liable for special, indirect, incidental, or consequential damages.

Products may not be returned without prior authorization. The following contact information is for US and International customers residing in countries served by Campbell Scientific, Inc. directly. Affiliate companies handle repairs for customers within their territories. Please visit www.campbellsci.com to determine which Campbell Scientific company serves your country. To obtain a Returned Materials Authorization (RMA), contact CAMPBELL SCIENTIFIC, INC., phone (435) 753-2342. After an applications engineer determines the nature of the problem, an RMA number will be issued. Please write this number clearly on the outside of the shipping container. CAMPBELL SCIENTIFIC's shipping address is:

CAMPBELL SCIENTIFIC, INC.

RMA# _____
815 West 1800 North
Logan, Utah 84321-1784

CAMPBELL SCIENTIFIC, INC. does not accept collect calls.

CR3000 Table of Contents

PDF viewers note: These page numbers refer to the printed version of this document. Use the Adobe Acrobat® bookmarks tab for links to specific sections.

CR3000 Overview	OV-1
OV1. Physical Description	OV-2
OV1.1 Measurement Inputs	OV-3
OV1.1.1 Analog Inputs (SE 1-28, DIFF 1-14)	OV-3
OV1.1.2 Signal Grounds ($\frac{\oplus}{\ominus}$)	OV-3
OV1.1.3 Power Grounds (G)	OV-3
OV1.1.4 Ground Lug ($\frac{\oplus}{\ominus}$)	OV-3
OV1.1.5 Power In (G and 12V)	OV-3
OV1.1.6 Switched 12 Volts (SW-12)	OV-3
OV1.1.7 12 Volt Outputs	OV-4
OV1.1.8 5V Output	OV-4
OV1.1.9 Switched Voltage Excitation	OV-4
OV1.1.10 Switched Current Excitation	OV-4
OV1.1.11 Continuous Analog Outputs	OV-4
OV1.1.12 Digital I/O	OV-4
OV1.1.13 Pulse Inputs	OV-4
OV1.2 Communication and Data Storage	OV-5
OV1.2.1 Peripheral Port	OV-5
OV1.2.2 CS I/O	OV-5
OV1.2.3 Computer RS-232	OV-6
OV1.3 Power Supply and AC Adapter	OV-7
OV2. Memory and Operating Concepts	OV-7
OV2.1 Memory	OV-7
OV2.2 Programming	OV-7
OV2.3 Instruction Execution within the Datalogger	OV-8
OV2.3.1 Pipeline Mode	OV-8
OV2.3.2 Sequential Mode	OV-9
OV2.3.3 Slow Sequence Scans	OV-9
OV2.3.4 Task Priority	OV-9
OV2.4 Data Tables	OV-10
OV2.5 PakBus Communication with the CR3000	OV-10
OV2.6 Set up: Device Configuration Utility or Keyboard Display	OV-11
OV3. Device Configurator	OV-11
OV3.1 Main DevConfig Screen	OV-12
OV3.2 Deployment Tab	OV-13
OV3.2.1 Datalogger	OV-13
OV3.2.2 Ports Settings	OV-14
OV3.2.3 TCP/IP	OV-16
OV3.2.4 Advanced	OV-17
OV3.3 Logger Control Tab	OV-18
OV3.4 Data Monitor Tab	OV-19
OV3.5 Send OS Tab - Downloading an Operating System	OV-19
OV3.6 Settings Editor Tab	OV-21
OV4. Quick Start Tutorial	OV-23
OV4.1 Software Products for the CR3000	OV-23

- OV4.1.1 Options for Creating CR3000 Programs OV-24
- OV4.2 Connections to the CR3000 OV-24
- OV4.3 Setting the CR3000 PakBus Address OV-24
- OV4.4 PC200W Software OV-24
 - OV4.4.1 Creating a CR3000 Program using Short Cut OV-25
 - OV4.4.2 Configuring the Setup Tab OV-30
 - OV4.4.3 Synchronize the Clocks OV-30
 - OV4.4.4 Send the Program OV-30
 - OV4.4.5 Monitor Data Tables OV-30
 - OV4.4.6 Collect Data OV-31
 - OV4.4.7 View Data OV-32
- OV4.5 Programming using the CRBasic Program Editor OV-33
- OV5. Keyboard Display OV-34
 - OV5.1 Data Display OV-36
 - OV5.1.1 Real Time Tables OV-37
 - OV5.1.2 Real Time Custom OV-38
 - OV5.1.3 Final Storage Tables OV-39
 - OV5.2 Run/Stop Program OV-40
 - OV5.3 File Display OV-41
 - OV5.3.1 File: Edit OV-42
 - OV5.4 PCCard Display OV-43
 - OV5.5 Ports and Status OV-44
 - OV5.6 Settings OV-45
 - OV5.6.1. Set Time/Date OV-45
 - OV5.6.2 PakBus Settings OV-45
 - OV5.6.3 Configure Display OV-46
- OV6. Specifications OV-47

1. Installation and Maintenance 1-1

- 1.1 Protection from the Environment 1-1
- 1.2 Power Requirements 1-2
- 1.3 Campbell Scientific Power Supplies 1-3
 - 1.3.1 CR3000 Alkaline Battery Base (Part Number 10519) 1-4
 - 1.3.2 CR3000 Lead Acid Battery Base (Part Number 10518) 1-5
 - 1.3.3 CR3000 Low Profile Base (Part Number 10695) 1-7
 - 1.3.3.1 BPALK Alkaline Power Supply 1-7
 - 1.3.3.2 PS100 Lead Acid Power Supply 1-8
 - 1.3.3.3 CH100 1-9
 - 1.3.3.4 A100 Null Modem Adapter 1-10
- 1.4 Solar Panels 1-10
- 1.5 Direct Battery Connection to the CR3000 Wiring Panel 1-10
- 1.6 Vehicle Power Supply Connections 1-11
- 1.7 CR3000 Grounding 1-11
 - 1.7.1 ESD Protection 1-11
 - 1.7.2 Effect of Grounding on Measurements: Common Mode Range 1-13
 - 1.7.3 Effect of Grounding on Single-Ended Measurements 1-14
- 1.8 Powering Sensors and Peripherals 1-14
- 1.9 Controlling Power to Sensors and Peripherals 1-15
 - 1.9.1 Use of Digital I/O Ports for Switching Relays 1-16
- 1.10 Maintenance 1-17
 - 1.10.1 Desiccant 1-17
 - 1.10.2 Replacing the Internal Battery 1-17

2. Data Storage and Retrieval	2-1
2.1 Data Storage in CR3000	2-1
2.1.1 Internal SRAM.....	2-1
2.1.2 CFM100 or NL115.....	2-1
2.2 Internal Data Format	2-2
2.3 Data Collection	2-3
2.3.1 Via a Communications Link	2-4
2.3.2 Via CF Card	2-4
2.3.2.1 Inserting a CF Card	2-4
2.3.2.2 Removing Card from CR3000.....	2-5
2.3.2.3. Converting File Format	2-5
2.4 Data Format on Computer	2-5
2.4.1 Header Information	2-5
2.4.2 TOA5 ASCII File Format	2-7
2.4.3 TOB1 Binary File Format	2-7
2.4.4 TOB3 Binary File Format	2-8
3. CR3000 Measurement Details	3-1
3.1 Analog Voltage Measurement Sequence	3-1
3.1.1 Voltage Range.....	3-1
3.1.2 Reversing Excitation or the Differential Input	3-3
3.1.3 Measuring Single-Ended Offset	3-3
3.1.4 Settling Time.....	3-3
3.1.5 Integration	3-4
3.2 Single Ended and Differential Voltage Measurements	3-4
3.3 Signal Settling Time	3-6
3.3.1 Minimizing Settling Errors	3-6
3.3.2 Measuring the Necessary Settling Time.....	3-7
3.4 Thermocouple Measurements	3-8
3.4.1 Error Analysis	3-9
3.4.2 Use of External Reference Junction or Junction Box.....	3-16
3.5 Bridge Resistance Measurements	3-17
3.6 Measurements Requiring AC Excitation	3-19
3.7 Pulse Count Measurements.....	3-20
3.8 Self Calibration	3-21
3.9 Measurement Accuracy	3-22
4. CRBASIC - Native Language Programming	4-1
4.1 Format Introduction	4-1
4.1.1 Mathematical Operations	4-1
4.1.2 Measurement and Output Processing Instructions	4-1
4.1.3 Inserting Comments Into Program	4-2
4.2 Programming Sequence	4-2
4.3 Example Program	4-4
4.3.1 Data Tables	4-4
4.3.2 The Scan -- Measurement Timing and Processing.....	4-6
4.4 Variable Data Types	4-7
4.4.1 FLOAT.....	4-7
4.4.2 LONG	4-7
4.4.3 BOOLEAN	4-7
4.4.4 STRING	4-7
4.4.5 Numerical Expressions with Floats, Longs and Booleans	4-8

4.5 Numerical Entries	4-9
4.6 Logical Expression Evaluation.....	4-10
4.6.1 What is True?	4-10
4.6.2 Expression Evaluation	4-10
4.6.3 Numeric Results of Expression Evaluation	4-10
4.7 Flags.....	4-11
4.8 Parameter Types.....	4-11
4.8.1 Expressions in Parameters.....	4-12
4.8.2 Arrays of Multipliers Offsets for Sensor Calibration	4-12
4.9 Program Access to Data Tables	4-13
5. Program Declarations.....	5-1
6. Data Table Declarations and Output Processing Instructions	6-1
6.1 Data Table Declaration	6-1
6.2 Trigger Modifiers	6-2
6.3 Export Data Instructions	6-8
6.4 Output Processing Instructions.....	6-12
7. Measurement Instructions	7-1
7.1 Voltage Measurements.....	7-3
7.2 Thermocouple Measurements	7-3
7.3 Half Bridges	7-5
7.4 Full Bridges.....	7-8
7.5 Excitation	7-10
7.6 Self Measurements	7-12
7.7 Digital I/O	7-16
7.8 Specific Sensors	7-27
7.9 Peripheral Devices	7-30
8. Processing and Math Instructions	8-1
9. Program Control Instructions	9-1
10. Custom Keyboard Display Menus	10-1
11. String Functions	11-1
11.1 Expressions with Strings	11-1
11.1.1 Constant Strings	11-1
11.1.2 Add Strings.....	11-1
11.1.3 Subtraction of Strings.....	11-1
11.1.4 String Conversion to/from Numeric	11-1
11.1.5 String Comparison Operators	11-2
11.1.6 Sample () Type Conversions and Other Output Processing Instructions	11-2
11.2 String Manipulation Functions.....	11-2

12. Serial Input and Output Functions 12-1**13. PakBus Communication Instructions 13-1****Appendix****A. CR3000 Status Table..... A-1****Figures**

OV1-1. CR3000 Measurement and Control System.....	OV-1
OV1-2. CR3000 Wiring Panel and Associated Instructions	OV-2
1.2-1. CR3000 Battery Pack and Panel.....	1-3
1.3-1. Alkaline Battery Orientation	1-4
1.3-2. Lead Acid Battery Wiring	1-6
1.3-3. BPALK Power Supply	1-7
1.6-1. Connecting CR3000 to Vehicle Power Supply.....	1-11
1.7-1. Schematic of CR3000 Grounds	1-12
1.9-1. Relay Driver Circuit with Relay	1-16
1.9-2. Power Switching without Relay	1-17
1.10-1. CR3000 with wiring panel.....	1-19
1.10-2. Loosen thumbscrews to remove CR3000 wiring panel from base	1-19
1.10-3. Disconnect internal battery from wiring panel.	1-20
1.10-4. Remove plate to expose lithium battery	1-20
1.10-5. Remove battery cover and replace battery.....	1-21
3.3-1. Settling Time for Pressure Transducer	3-8
3.4-1. Panel Temperature Errors.....	3-10
3.4-2. Panel Temperature Gradients during -65 to 70 °C Change	3-11
3.4-3. Panel Temperature Gradients during 75 to 25 °C Change.....	3-11
3.4-4. Diagram of Junction Box.....	3-17
3.5-1. Circuits Used with Bridge Measurement Instructions	3-19
3.6-1. Model of Resistive Sensor with Ground Loop	3-20
3.7-1. Varying counts within pulse interval.....	3-21
3.9-1. Instrument voltage measurement error performance illustrating dependence upon input voltage.....	3-23
3.9-2. Comparison of the CR10X \pm (0.X% of Full-Scale Range) and CR1000 \pm (0.X% of reading + Offset) voltage measurement accuracy specifications for the (0 to 40) °C temperature range.....	3-24
6.4-1. Example Crossing Data	6-22
6.4-2. Crossing Data with Second Dimension Value.....	6-23
6.4-3. Input Sample Vectors	6-32
6.4-4. Mean Wind Vector	6-33
6.4-5. Standard Deviation of Direction.....	6-34
7.7-1. Input conditioning circuit for low-level and high level period averaging	7-18
7.7-2. Conditioning Large Voltage Pulses	7-22
8-1. Dew Point Temperature over the RH Range for Selected Air Temperatures	8-11
8.2 Effect of RH Errors on Calculated Dew Point.....	8-11

Tables

OV1-2. Computer RS-232 Pin-Out	OV-6
OV2-1. Typical Data Table	OV-10
1.3-1. Typical Alkaline Battery Service and Temperature	1-5
1.3-2. CR3000 Rechargeable Battery and AC Transformer Specifications	1-6
1.3-3. PS100, Battery, and AC Transformer Specifications	1-9
1.8-1. Current Sourcing Limits	1-15
1.8-2. Typical Current Drain for Some CR3000 Peripherals	1-15
1.10-1. CR3000 Lithium Battery Specifications	1-18
2.2-1. CR3000 Data Types	2-3
2.2-2. Resolution and Range Limits of FP2 Data	2-3
2.2-3. FP2 Decimal Location	2-3
3.3-1. First Six Values of Settling Time Data	3-8
3.4-1. Limits of Error for Thermocouple Wire	3-12
3.4-2. Voltage Range for Maximum Thermocouple Resolution	3-13
3.4-3. Limits of Error on CR3000 Thermocouple Polynomials	3-14
3.4-4. Reference Temperature Compensation Range and Polynomial Error Relative to NIST Standards	3-15
3.4-5. Example of Errors in Thermocouple Temperature	3-16
4.3-1. Formats for Output Data	4-6
4.5-1. Formats for Entering Numbers in CRBasic	4-9
4.6-1. Synonyms for True and False	4-10
4.8-1. Rules for Names	4-12
7.7-1. Calibrate Return Value Decode	7-14
13.1-1. ComPort Parameter Codes	13-1
13.1-2. ResultCode Error Codes	13-2

CR3000 Overview

The CR3000 provides precision measurement capabilities with processing and control capability in a rugged, battery-operated package. The CR3000 includes CPU and analog and digital inputs and outputs. The on-board, BASIC-like programming language includes data processing and analysis routines. PC200, PC400, or LoggerNet software provides program generation and editing, data retrieval, and realtime monitoring.

Campbell Scientific, Inc. provides the following documents to aid in understanding and operating the CR3000:

1. This Overview
2. The CR3000 Operator's Manual

The online help found in PC400 and LoggerNet software provides valuable information regarding the programming of the CR3000 as well as program examples.

This Overview introduces the concepts required to take advantage of the CR3000's capabilities. Hands-on programming examples start in Section OV4. Working with a CR3000 will help the learning process, so don't just read the examples, turn on the CR3000 and do them. If you want to start this minute, go ahead and try the examples, then come back and read the rest of the Overview.



FIGURE OV1-1. CR3000 Measurement and Control System

OV1. Physical Description

Figure OV1-2 shows the CR3000 panel and the associated program instructions. The details of the measurement instructions can be found in Section 7.

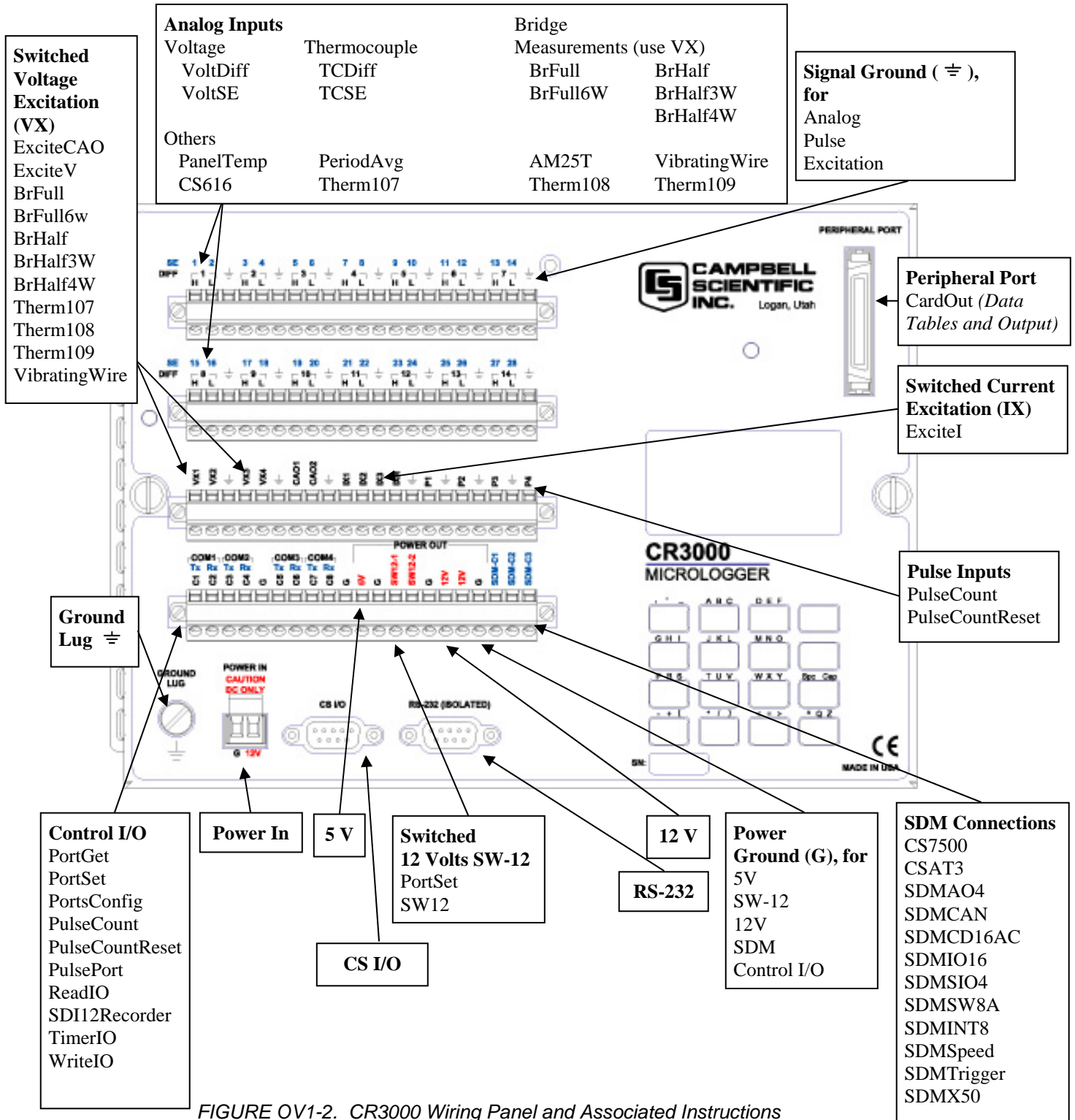


FIGURE OV1-2. CR3000 Wiring Panel and Associated Instructions

OV1.1 Measurement Inputs

OV1.1.1 Analog Inputs (SE 1-28, DIFF 1-14)

There are 14 differential or 28 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 $0.67 \mu\text{V}$

OV1.1.2 Signal Grounds (\equiv)

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

Signal returns from the CAO and Pulse channels should use the \equiv terminals located on the CAO and Pulse terminal strip to minimize current flow through the \equiv grounds on the analog 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 analog section, which can cause Single-ended voltage measurement errors.

OV1.1.4 Ground Lug (\equiv)

The large ground lug is used to tie the ground potential of the datalogger to earth ground. A conductive connection, using a heavy gage wire, is necessary to ensure equivalent ground potentials. This path to ground is also used to shunt incoming electrical transients to ground; these transients may be induced on the shield wire of the connected sensor leads.

OV1.1.5 Power In (G and 12V)

The G and 12V terminals on the Power In connector plug are for connecting power from an external battery to the CR3000. These are the only terminals that can be used to input battery power; the other 12V and SW-12V terminals are output only.

The green power connector on the wiring panel is a plug in connector that allows the power supply to be easily disconnected. The power connection is reverse polarity protected.

For a CR3000 with an alkaline battery base or a rechargeable battery base, it is not necessary to connect anything to the Power In terminals.

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 12 Volt Outputs

The 12V terminals provide a constant unregulated 12 volts for powering external devices such as multiplexers and SDM modules.

The 12V is common with pin 8 on the CS I/O pin connector.

OV1.1.8 5V Output

The 5 V ($\pm 4.0\%$) output is commonly used to power peripherals such as the QD1 Incremental Encoder Interface, AVW1 or AVW4 Vibrating Wire Interface.

The 5 V output is common with pin 1 on the CS I/O 9 pin connector; 200 mA is the maximum combined current output.

OV1.1.9 Switched Voltage Excitation (VX)

Four switched excitation channels provide precision programmable voltages within the ± 5 Volt range for bridge measurements. Each analog output will provide up to 25 mA between ± 5.0 V.

OV1.1.10 Switched Current Excitation (IX)

Three switched current channels provide precision programmable current output within the ± 2.5 mA range for bridge measurements.

OV1.1.11 Continuous Analog Outputs (CAO)

Two CAO channels supply continuous output voltages in the ± 5 Volt range under program control, for use with strip charts, x-y plotters, or proportional controllers.

OV1.1.12 Digital I/O (C1-8)

There are 8 digital Input/Output channels (0 V low, 5 V high) for frequency measurement, pulse counting, digital control, triggering, and SDI-12 sensors. In addition to the individual channel digital I/O functions, there are several groups of channels that can be used for other functions.

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

The COM groupings can be used for serial I/O communication and Intelligent Sensor input.

OV1.1.13 Pulse Inputs (P1-4)

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

OV1.2 Communication and Data Storage

OV1.2.1 Peripheral Port

The peripheral port is for attaching data storage or communication peripherals. Both the CFM100 and NL115 modules plug onto the peripheral port and have a slot for a Type I or Type II CompactFlash[®] card (Section 2.1.2). The NL115 also supports Ethernet communications.

CAUTION

Removing a card from the CFM100 or NL115 while the card is active can cause garbled data and can actually damage the card. Always press the button to disable the card for removal before switching off the CR3000 power.

OV1.2.2 CS I/O

All Campbell Scientific communication peripherals connect to the CR3000 through the 9-pin subminiature D-type socket connector located on the front of the Wiring Panel labeled “CS I/O” (Figure OV1-3). Table OV1-1 gives a brief description of each pin.

TABLE OV1-1. Pin Description

TABLE OV1-1. Pin Description			
ABR = Abbreviation for the function name. PIN = Pin number. O = Signal Out of the CR3000 to a peripheral. I = Signal Into the CR3000 from a peripheral.			
PIN	ABR	I/O	Description
1	5 V	O	5V: Sources 5 VDC, used to power peripherals.
2	SG		Signal Ground: Provides a power return for pin 1 (5V), and is used as a reference for voltage levels.
3	RING	I	Ring: Raised by a peripheral to put the CR3000 in the telecommunications mode.
4	RXD	I	Receive Data: Serial data transmitted by a peripheral are received on pin 4.
5	ME	O	Modem Enable: Raised when the CR3000 determines that a modem raised the ring line.
6	SDE	O	Synchronous Device Enable: Used to address Synchronous Devices (SDs), and can be used as an enable line for printers.
7	CLK/HS	I/O	Clock/Handshake: Used with the SDE and TXD lines to address and transfer data to SDs. When not used as a clock, pin 7 can be used as a handshake line (during printer output, high enables, low disables).
8	+12 VDC		
9	TXD	O	Transmit Data: Serial data are transmitted from the CR10X to peripherals on pin 9; logic low marking (0V) logic high spacing (5V) standard asynchronous ASCII, 8 data bits, no parity, 1 start bit, 1 stop bit, 300, 1200, 2400, 4800, 9600, 19,200, 38,400, 115,200 baud (user selectable).

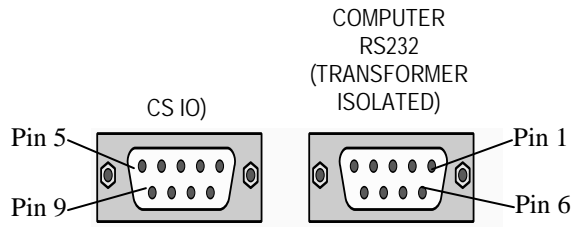


FIGURE OV1-3. Serial Communication Interfaces

OV1.2.3 Computer RS-232

The CR3000 has an isolated RS-232 port.

Direct connection of the CR3000 to a PC is most conveniently done through the "Computer RS232" port (Figure OV1-3). Table OV1-2 gives a brief description of each "Computer RS232" pin.

The Computer RS-232 port is a DCE device when connected to a PC with a serial cable. It also doubles as a DTE device when connected to a modem device through a null-modem cable. (DTR function is on pin I, Ring is an input).

Maximum input = ± 25V

Minimum Output = ± 5V

Typical Output = ± 7V

NOTE

Serial communications is not reliable over cable greater than 50 feet in length.

TABLE OV1-2. Computer RS-232 Pin-Out			
ABR = Abbreviation for the function name			
PIN = Pin number			
O = Signal Out of the CR3000 to a RS-232 device			
I = Signal Into the CR3000 from a RS-232 device			
PIN	ABR	I/O	Description
1	DTR	O	data terminal ready
2	TX	O	asynchronous transmit
3	RX	I	asynchronous receive
4			not connected
5	GND		isolated ground
6		O	connected to pin
7	CTS	I	clear to send
8	RTS	O	request to send
9	RING	I	ring

The CR3000 is supplied with a six foot 9-pin to 9-pin serial cable and a 9- to 25-pin adapter to facilitate connection to a PC RS-232 port.

OV1.3 Power Supply and AC Adapter

The CR3000 should be powered by any clean, battery backed 12 VDC power supply. For internal power supplies, a 10 Ahr alkaline battery base and a 7 Ahr rechargeable battery base are available. If internal batteries are not used, an external power supply such as the PS100 power supply should be used. The PS100 has a 7 amp hour battery with built in charging regulator. Optional adapters for AC power are available. Charging power can also come from a 17-28 VDC input such as a solar panel.

The datalogger should be earth or chassis ground during routine operation. See Section 1 for details on power supply connections and grounding.

When primary power falls below 10.0 VDC, the CR3000 stops executing its programs. The Low12VCount field in the Status table is incremented by one each time the primary power falls below 10.0 VDC.

The datalogger program and stored data remain in memory, and the clock continues to keep time when power is disconnected. The clock and SRAM are powered by an internal lithium battery.

OV2. Memory and Operating Concepts

OV2.1 Memory

The CR3000 has one MB Flash EEPROM that is used to store the Operating System. Another 128 K of Flash is used to store configuration settings. Four Megabytes of SRAM are available for program storage (16K), operating system use, and data storage. The size of available memory may be seen in the status file. Additional data storage is available by using a compact flash card in the optional CFM100 Compact Flash Module or NL115 Ethernet Interface and Compact Flash Module (Section 2.1.2).

OV2.2 Programming

The CR3000 program directs how and when the sensors are measured and data are stored. The program is created on a computer and sent to the CR3000. The CR3000 can store a number of programs in memory. Campbell Scientific has two software applications that create CR3000 programs: ShortCut and the CRBasic Editor.

For many applications ShortCut is a good place to start. With ShortCut you select the sensors to measure, the units to report the measurements in, and the data to output. ShortCut supports most of the sensors sold by Campbell Scientific as well as generic measurements. The CR3000 programs created by ShortCut are generally clear and provide a good example of CRBasic code for those who wish to write CR3000 programs themselves.

For those that have the need or inclination to tackle more complex programs, the CRBasic Editor is used to create and edit the CRBasic programs that the CR3000 runs. Section 4 provides an introduction to CRBasic Programming. The CRBasic Editor has syntax highlighting and online help for the CR3000 instruction set described in Sections 5-12.

ShortCut is included with PC200, PC400 and LoggerNet and is available for free from the Campbell Scientific web site. The CRBasic Editor is included in PC400 and LoggerNet.

OV2.3 Instruction Execution within the Datalogger

The execution of instructions within the datalogger is accomplished using three separate task types: measurement, SDM, and processing. As it is named, the measurement task handles measuring the signals received on the datalogger's wiring panel, as well as outputting signals for control of other devices. The measurement and control hardware is manipulated on a rigidly timed sequence. The SDM task handles the measurement and control of most SDM devices. The processing task converts the raw signals read by the datalogger into numbers representing engineering units, performs calculations, stores data, makes the decisions to actuate controls, and performs serial I/O communication.

<p>Measurement Task</p> <ul style="list-style-type: none"> • Analog Measurements • Excitation • Read Pulse Counters • Read Control Ports (GetPort) • Set Control Ports (SetPort) • VibratingWire • PeriodAvg • CS616 • Calibrate 	<p>SDM Task</p> <ul style="list-style-type: none"> • All SDM instructions, except SMDSIO4 and SCMIO16 	<p>Processing Task</p> <ul style="list-style-type: none"> • Processing • Output • Serial I/O • SDMSIO4 • SDMIO16 • ReadIO • WriteIO • Expression evaluation and variable setting in measurement and SDM instructions
--	---	---

The datalogger can execute these tasks in either pipeline or sequential mode. When a program is compiled the datalogger evaluates the program and determines which mode to use. This information is included in a message returned by the datalogger and is displayed by the support software. CRBasic's precompiler returns a similar message. A program can be forced to run in sequential mode by placing the SequentialMode instruction in the declarations section of the program.

OV2.3.1 Pipeline Mode

In pipeline mode, the measurement task, SDM task, and processing task are three separate functions. In this mode the three tasks may operate simultaneously. The measurement tasks are scheduled to take place at exact times and with the highest priority when the datalogger starts each scan. This results in a more precise timing of measurements, and may be more efficient with processing and power consumption. However, this prescheduling of measurements means measurement instructions must be executed every scan, and because multiple tasks are taking place at the same time, the sequence in which the instructions are executed may not be in the exact order in which they appear in the program. For these reasons, conditional measurements are not allowed in pipeline mode. Also note that because of the precise execution of measurement instructions, processing for the measurements in the current scan (including update of public variables and output to data tables) is delayed until all measurements are completed.

OV2.3.2 Sequential Mode

In sequential mode the instructions are executed in the sequence they appear in the program. Sequential mode can be slower than pipeline mode since it does only one step of the program at a time. After a measurement is made the result is converted to a value determined by the processing included in the instruction, and then the datalogger proceeds to the next instruction. Because of this step-by-step instruction execution, conditional measurements are allowed in sequential mode. The exact time at which measurements are made may vary if other measurements or processing are made conditionally, if there is heavy communications activity or other interrupts (e.g., inserting a CF card).

OV2.3.3 Slow Sequence Scans

The datalogger allows for one or more scans that are run outside of the instructions placed between the Scan/NextScan instructions in the main program. These scans, referred to as slow sequence scans, typically run at a slower rate than the main scan. Up to four slow sequences can be defined in a program (slow sequences are declared with the SlowSequence instruction).

Instructions in a slow sequence scan are executed whenever the main scan is not active. When running in pipeline mode, slow sequence measurements will be spliced in after measurements in the main program, as time allows. Because of this splicing, the measurements in a slow sequence may actually span across multiple main program scan intervals. In sequential mode, all instructions in the slow sequences are executed as they occur in the program (see Task Priority, below).

OV2.3.4 Task Priority

When considering the information above regarding pipeline and sequential mode, you must also consider that some sequences in the program may have higher priorities than other sequences in the program, and that measurement tasks generally take precedence over all others. In addition, the priority of sequences is different for pipeline mode and sequential mode.

When running in pipeline mode, measurement tasks have priority over all other tasks. Measurements in the main program have the highest priority, then background calibration, followed by any measurements in slow sequences that may be defined. The execution of processing tasks are handled by a task sequencer, and all tasks are given the same priority. When a condition is true for a task to start running it is put in a queue (this true condition can be based on time, the triggering of WaitDigTrig, the expiration of a Delay instruction, or a ring on a COM port triggering communication). Because all tasks are given the same priority, the task is put at the back of the queue. Every 10 msec (or faster if a new task is triggered) the task currently running is paused and put at the back of the queue, and the next task in the queue begins running. In this way, all tasks are given equal processing time by the datalogger. The only exception to this task switching queue is when a measurement task is triggered. In most instances the processing task and the measurement task should be able to run in parallel. However, if the datalogger is unable to complete a measurement when the task sequencer is executing, the task will be interrupted until the measurement is made.

When running in sequential mode, the datalogger uses a queuing system for processing tasks similar to the one used in the pipeline mode. The main difference when running a program in sequential mode is that there is no prescheduled timing of measurements; instead, all of the instructions are run in the order they occur in the program. A priority scheme is used to avoid conflicting use of measurement hardware. In this scheme the main scan has the highest priority and prevents other sequences from using measurement hardware until the main scan is completed (including processing). Other tasks, such as processing from other sequences and communications, can occur while the main sequence is running. Once the main scan has finished other sequences have access to measurement hardware with the order of priority being the background calibration sequence followed by the slow sequences in the order they are declared in the program. Note that Measurement tasks have priority over other tasks such as processing and communication to allow accurate timing needed within most measurement instructions, e.g. integrations.

OV2.4 Data Tables

The CR3000 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 such as listed in Table OV2-1. The values to output are selected when running ShortCut or when writing a datalogger program directly.

TOA5	Fritz	CR3000	1079	CR3000.Std.1.0	CPU:TCTemp.CR1	51399	Temp	
TIMESTAMP	RECORD	RefT_Avg	TC_Avg(1)	TC_Avg(2)	TC_Avg(3)	TC_Avg(4)	TC_Avg(5)	TC_Avg(6)
TS	RN	degC	DegC	DegC	DegC	DegC	DegC	DegC
		Avg	Avg	Avg	Avg	Avg	Avg	Avg
10/28/2004 12:10	119	23.52	23.49	23.49	23.5	23.49	23.5	23.5
10/28/2004 12:20	120	23.55	23.51	23.51	23.51	23.51	23.51	23.52
10/28/2004 12:30	121	23.58	23.52	23.53	23.53	23.53	23.53	23.53
10/28/2004 12:40	122	23.58	23.53	23.54	23.54	23.54	23.54	23.54

OV2.5 PakBus[®] Communication with the CR3000

The CR3000 uses the PakBus network communications protocol. PakBus increases the number of communications and networking options available to the datalogger. In addition to communicating via its RS-232 and/or CS I/O ports, the CR3000 can also communicate via the digital I/O COM ports.

Some of the advantages of PakBus are:

- Routing – the CR3000 can act as a router, passing on messages intended for another logger. PakBus supports automatic route detection and selection.
- Short distance networks with no extra hardware – A CR3000 can talk to another CR3000 or CR1000 over distances up to 30 feet by connecting 3 wires between the dataloggers: transmit, receive, and ground. A PC communicating with one of these loggers (e.g. via a phone modem or RF

to the CS I/O port) can be routed through that datalogger to the other datalogger.

- Datalogger to datalogger communications – Special PakBus instructions simplify transferring data between dataloggers for distributed decision making or control.

All devices that send or receive messages in a PakBus network must have a unique PakBus Address. The CR3000 default PakBus address is 1. In a PakBus Network each datalogger must be set to a unique address before it is installed in the network. To communicate with the CR3000, the PC software (e.g., LoggerNet) must know the CR3000's PakBus address.

OV2.6 Set up: Device Configuration Utility or Keyboard Display

When you receive a new CR3000 from Campbell Scientific it should be set to the default PakBus address, 1. If you only have one PakBus datalogger, or will only communicate with the CR3000 with a direct RS-232 or telephone modem connection, there may be no need to change the address.

However, if a CR3000 has been in use or someone has borrowed it, you may need to check what the address is or to set it or some other setting. While there are a number of ways to do this, the two most basic are to use the Device Configuration Utility (DevConfig) or the Keyboard display (see section OV5.6). DevConfig comes bundled with LoggerNet and PC400 software and is also available from the Campbell Scientific Inc. website (www.campbellsci.com).

OV3. Device Configurator

The Device Configuration Utility (DevConfig) sets up dataloggers and intelligent peripherals before those devices are deployed in the field and before these devices are added to networks in Campbell Scientific datalogger support software such as LoggerNet or PC400. Some key features of DevConfig include:

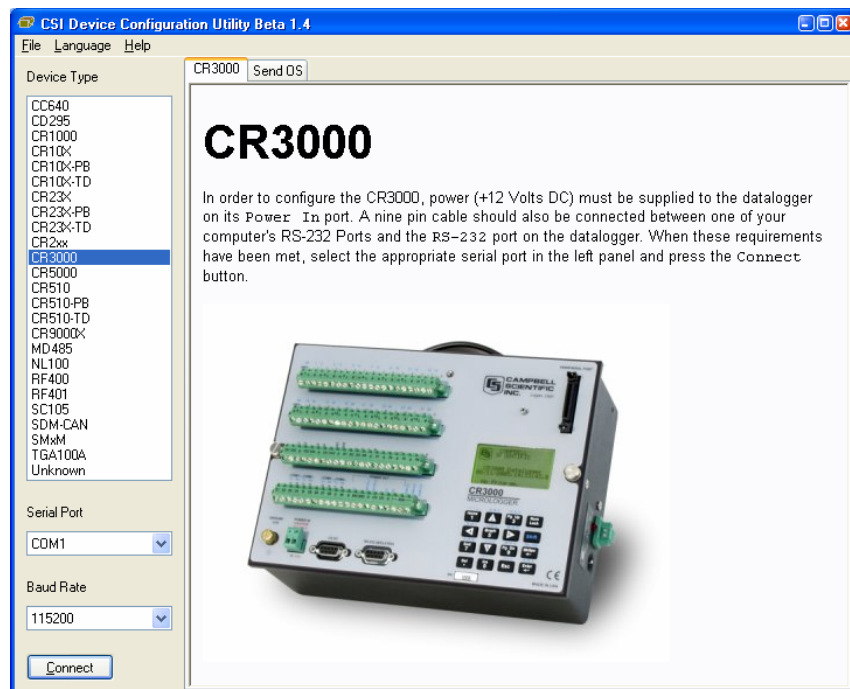
- DevConfig only supports direct serial connections between the PC and devices.
- DevConfig can send operating systems to supported device types.
- DevConfig can set datalogger clocks and send program files to dataloggers.
- DevConfig allows you to determine operating system types and versions
- DevConfig provides a reporting facility where a summary of the current configuration of a device can be shown on the screen and printed. This configuration can also be saved to a file and used to restore the settings in the same or a replacement device.

- Some devices may not support the configuration protocol in DevConfig, but do allow configurations to be edited through the terminal emulation screen.
- Help for DevConfig is shown as prompts and explanations on its main screen. Help for the appropriate settings for a particular device can also be found in the user's manual for that device.
- Updates to DevConfig are available from Campbell Scientific's web site. These may be installed over top of older versions.

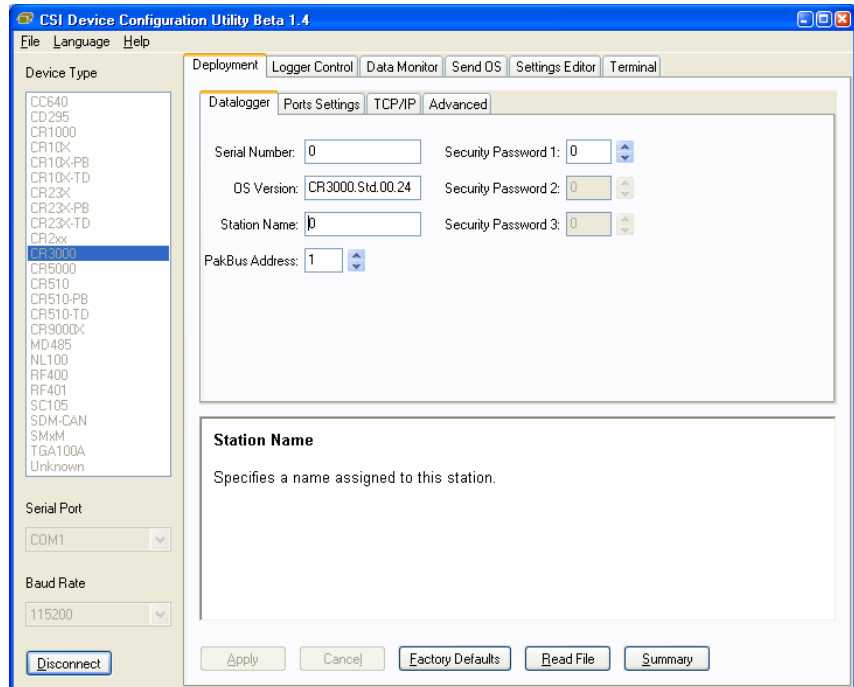
Note: Before opening DevConfig, make sure that you do not have other software open on the computer that uses the computer's serial (COM) ports (LoggerNet, PC400, PC200W, for example).

OV3.1 Main DevConfig Screen

The DevConfig window is divided into two main sections: the device selection panel on the left side and tabs on the right side. After choosing a device on the left, you will then have a list of the serial ports (COM1, COM2, etc.) installed on your PC. You'll be offered a choice of baud rates only if the device supports more than one baud rate in its configuration protocol. The page for each device presents instructions about how to set up the device to communicate with DevConfig. Different device types will offer one or more tabs on the right.



When the user presses the Connect button, the device type, serial port, and baud rate selector controls become disabled and, if DevConfig is able to connect to the CR3000, the button will change from "Connect" to "Disconnect". The Display will change to:



OV3.2 Deployment Tab

The Deployment Tab allows the user to configure the datalogger prior to deploying it.

OV3.2.1 Datalogger

Serial Number displays the CR3000 serial number. This setting is set at the factory and cannot be edited.

OS Version displays the operating system version that is in the CR3000.

Station Name displays the name that is set for this station.

PakBus Address allows you to set the PakBus address of the datalogger. The allowable range is between 1 and 4094. Each PakBus device should have a unique PakBus address. Addresses >3999 force other PakBus devices to respond regardless of their respective PakBus settings. See the [PakBus Networking Guide](#) for more information.

Security:

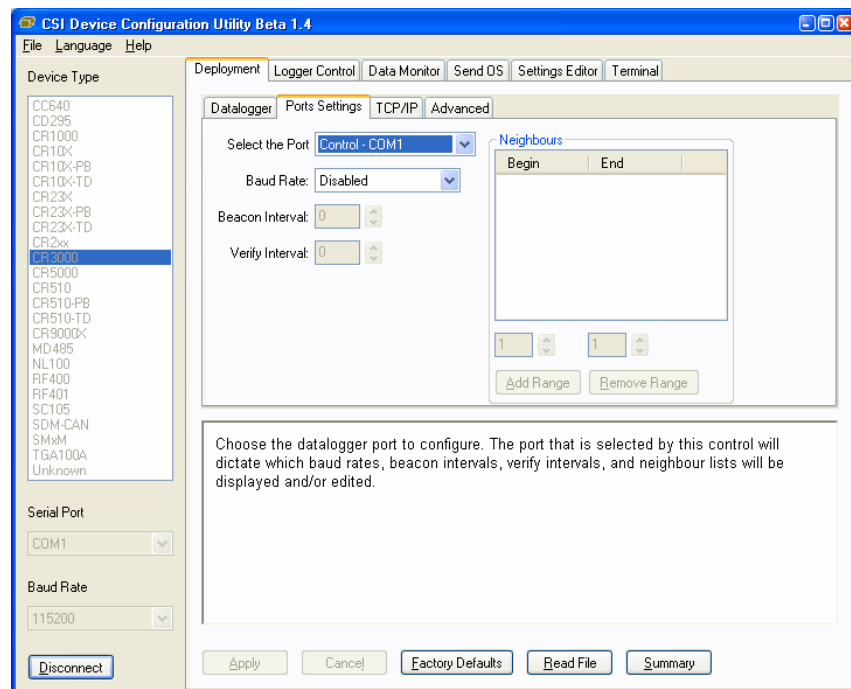
Up to three levels of security can be set in the datalogger. Level 1 must be set before Level 2 can be set, and Level 2 must be set before Level 3 can be set. If a level is set to 0, any level greater than it will also be set to 0 (e.g., if Level 2 is 0, Level 3 is 0). Valid security codes are 1 through 65535 (0 is no security). Each level must have a unique code. Functions affected by each level of security are:

Security Password 1 When this level is set, collecting data, setting the clock, and setting variables in the Public table are unrestricted, requiring no security code. If the user enters the Security1 code, the datalogger program can be changed or retrieved or variables can be set in the Status table.

Security Password 2 When this level is set, data collection is unrestricted, requiring no security code. If the user enters the Security2 code, the datalogger clock can be changed and variables in the public table can be changed. If the user enters the Security1 code, non-read-only values in the status table can be changed and the datalogger program can be changed or retrieved.

Security Password 3 When this level is set, all communication with the datalogger is prohibited if no security code is entered. If the user enters the Security3 code, data can be collected from the datalogger. If the user enters the Security2 code, data can be collected, public variables can be set, and the clock can be set. If the user enters the Security 1 code, all functions are unrestricted.

OV3.2.2 Ports Settings



Selected Port specifies the datalogger serial port to which the beacon interval and hello setting values will be applied.

Beacon Interval sets the interval (in seconds) on which the datalogger will broadcast beacon messages on the port specified by Selected Port.

Verify Interval specifies the interval (in seconds) at which the datalogger will expect to have received packets from neighbors on the port specified by Selected Port. A value of zero (default) indicates that the datalogger has no neighbor list for this port.

Neighbors List, or perhaps more appropriately thought of as the “expected neighbors list”, displays the list of addresses that this datalogger expects to find as neighbors on the port specified by Selected Port. As you select items in this list, the values of the **Begin** and **End** range controls will change to reflect the selected range. You can add multiple lists of neighbors on the same port.

Begin and End Range are used to enter a range of addresses that can either be added to or removed from the neighbors list for the port specified by Selected Port. As you manipulate these controls, the Add range and Remove Range buttons will be enabled or disabled depending on the relative values in the controls and whether the range is present in or overlaps with the list of address ranges already set up. These controls will be disabled if the **Verify Interval** value is set to zero.

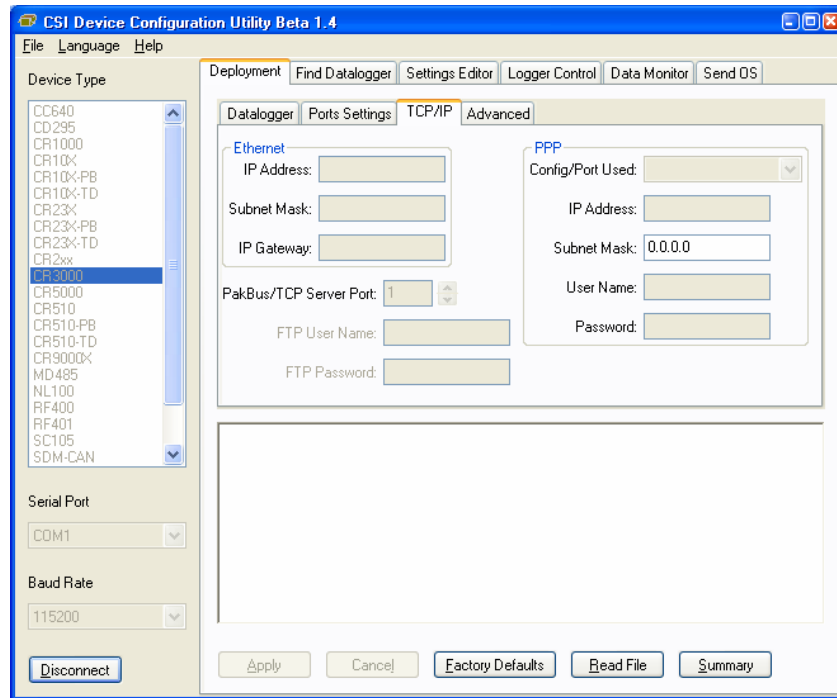
Add Range will cause the range specified in the **Begin** and **End** range to be added to the list of neighbors to the datalogger on the port specified by Selected Port. This control will be disabled if the value of the **Verify Interval** is zero or if the end range value is less than the begin range value.

Remove Range will remove the range specified by the values of the **Begin** and **End** controls from the list of neighbors to the datalogger on the port specified by Selected Port. This control will be disabled if the range specified is not present in the list or if the value of **Verify Interval** is set to zero.

Help is displayed at the bottom of the Deployment tab. When you’re finished, you must **Apply** to send the settings to the datalogger. The Summary window will appear and you can **Save** or **Print** the settings for your records or to use them as a template for another datalogger.

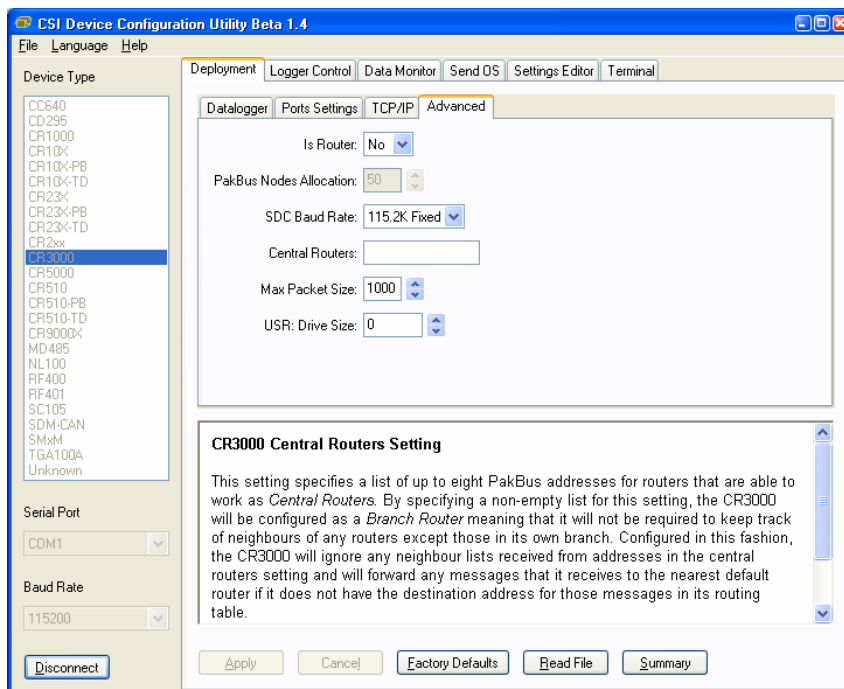
Cancel causes the datalogger to ignore the changes. **Read File** gives you the opportunity to load settings saved previously from this or another similar datalogger. If you load settings from a file, the changes will not actually be written to the datalogger until you click **Apply**.

OV3.2.3 TCP/IP



The TCP/IP tab is used to configure the CR3000 to communicate via Ethernet or PPP on any serial COM port. The Ethernet connection is provided by the NL115 parallel port module. The PPP protocol on a serial port requires no additional hardware but rather a device such as an IP modem or a computer that is set up to talk PPP on its serial port.

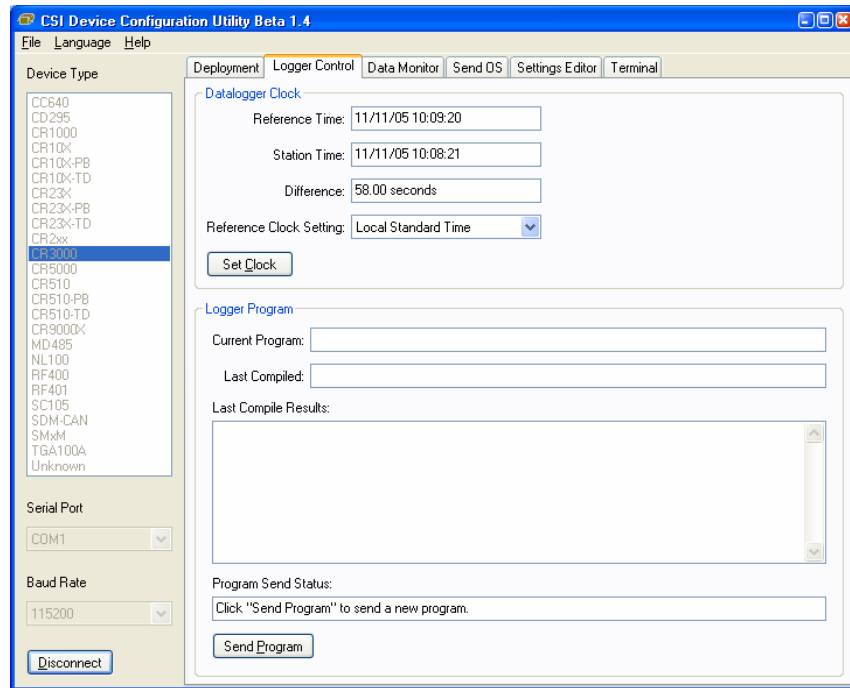
OV3.2.4 Advanced



Is Router allows you to control whether the datalogger will act as a PakBus router.

PakBus Nodes Allocation Specifies the amount of memory that the CR3000 allocates for maintaining PakBus Routing information. This value represents roughly the maximum number of PakBus Nodes that the CR3000 will be able to track in its routing tables.

OV3.3 Logger Control Tab



The clock in the PC and the datalogger will be checked every second and the difference displayed. The **System Clock Setting** allows you to configure what offset, if any, should be used with respect to standard time (Local Daylight Time or UTC, Greenwich mean time). The value selected for this control will be remembered between sessions. Clicking the **Set Clock Button** will synchronize the station clock to the current computer system time.

Current Program displays the current program known to be running in the datalogger. This value will be empty if there is no current program.

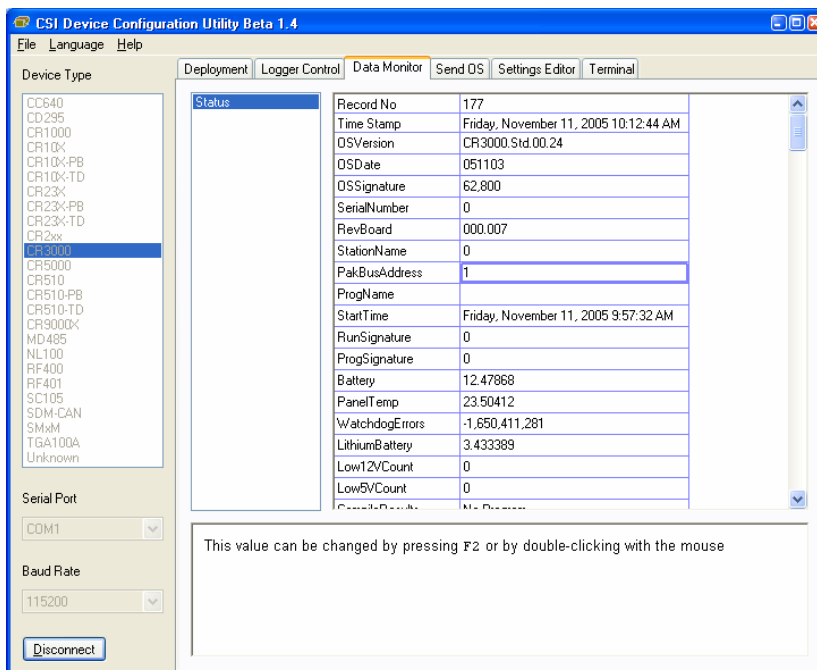
The **Last Compiled** field displays the time when the currently running program was last compiled by the datalogger. As with the Current Program field, this value will be read from the datalogger if it is available.

Last Compile Results shows the compile results string as reported by the datalogger.

The **Send Program** button presents an open file dialogue from which you can select a program file to be sent to the datalogger. The field above the button will be updated as the send operation progresses. When the program has been sent the Current Program, Last Compiled, and Last Compile Results fields will be filled in.

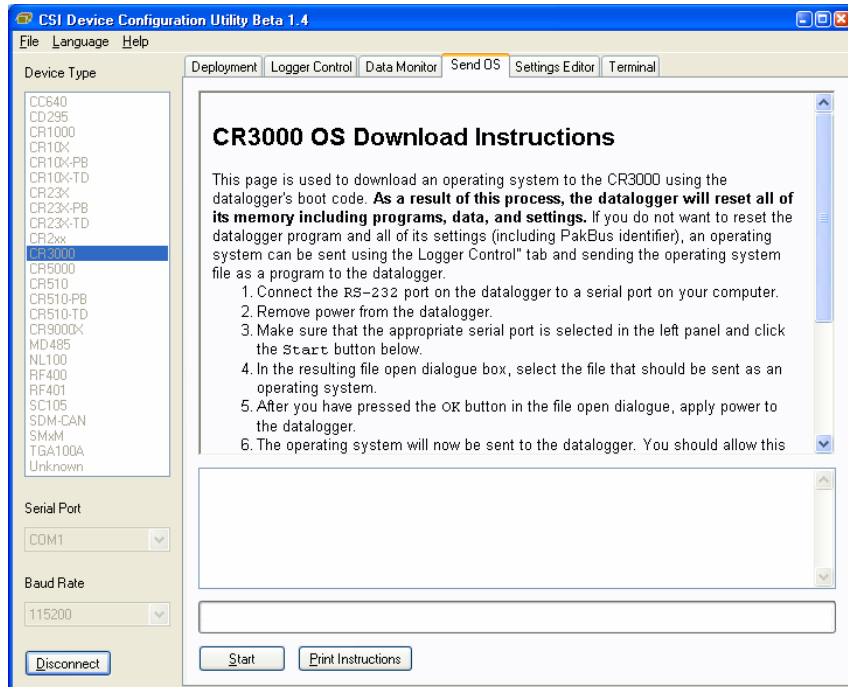
OV3.4 Data Monitor Tab

The Data Monitor tab shows the latest record in the tables stored in the CR3000.



OV3.5 Send OS Tab - Downloading an Operating System

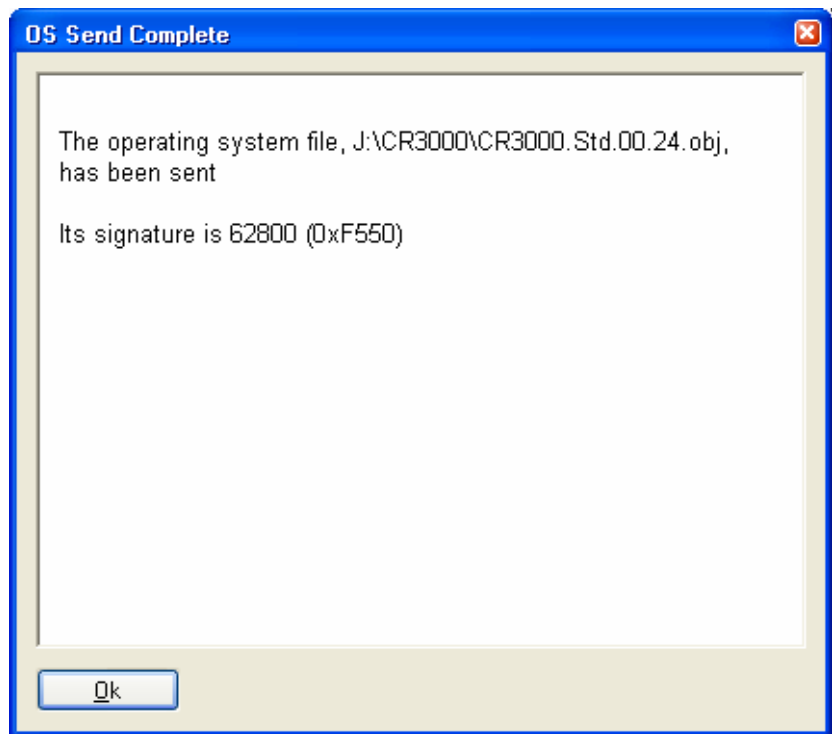
DevConfig can send operating systems to all Campbell Scientific devices with flash replaceable operating systems. Current operating systems are available from the Campbell Scientific Inc. website, www.campbellsci.com. An example for the CR3000 is shown below:



The text at right gives the instructions for downloading the OS. Follow these instructions.

When you click the **Start** button, DevConfig offers a file open dialog box to prompt you for the operating system file (*.obj file). When the CR3000 is then powered-up, DevConfig starts to send the operating system:

When the operating system has been sent, a message dialog will appear similar to the one shown below:

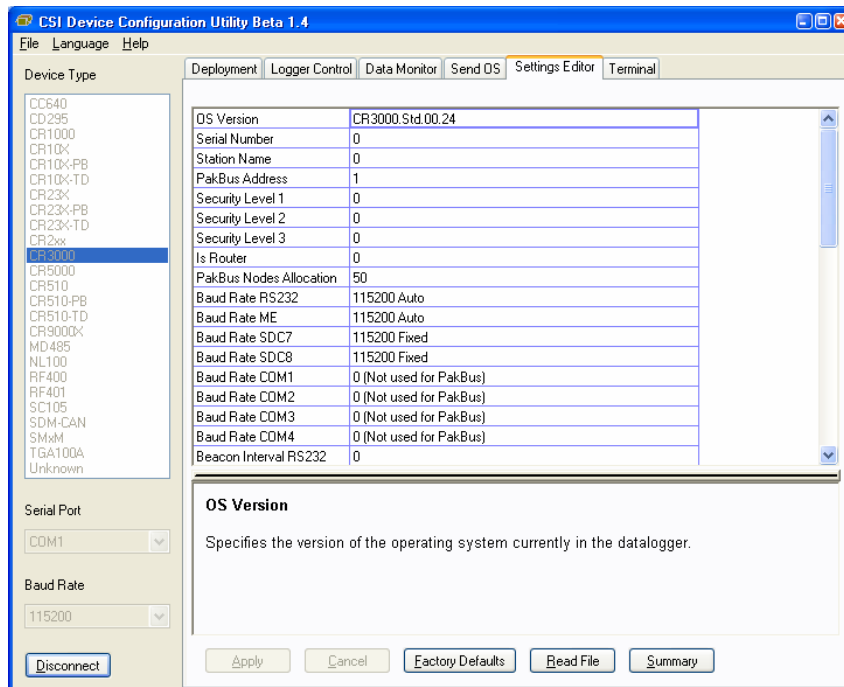


The information in the dialog helps to corroborate the signature of the operating system sent. For devices such as the CR10X (especially those with extended memory) that can take a long time to reset following an OS download, text warns you against interrupting the memory test.

OV3.6 Settings Editor Tab

The CR3000 has a number of properties, referred to as “settings”, some of which are specific to the PakBus protocol. PakBus is discussed in more detail in the [PakBus Networking Guide](#) available from the Campbell Scientific website (www.campbellsci.com).

The **Settings Editor** tab provides access to most of the PakBus settings, however, the **Deployment** tab makes configuring some of these settings a bit easier.



The top of the Settings Editor is a grid that allows the user to view and edit the settings for the device. The grid is divided into two columns with the setting name appearing in the left hand column and the setting value appearing in the right hand column. You can change the currently selected cell with the mouse or by using the up arrow and down arrow keys as well as the Page Up and Page Down keys. If you click in the setting names column, the value cell associated with that name will automatically be made active. You can edit a setting by selecting the value, pressing the F2 key or by double clicking on a value cell with the mouse. The grid will not allow read-only settings to be edited.

The bottom of the Settings Editor displays help for the setting that has focus on the top of the screen.

Once you have changed a setting, you can **Apply** them to the device or **Cancel**. These buttons will only become enabled after a setting has been changed. If the device accepts the settings, a configuration summary dialogue will be shown that will give the user a chance to save and/or print the settings for the device:

Setting Name	Setting Value
OS Version	CR3000.Std.00.24
Serial Number	0
Station Name	0
PakBus Address	1
Security Level 1	0
Security Level 2	0
Security Level 3	0
Is Router	0
PakBus Nodes Allocation	50

Clicking the **Factory Defaults** button on the Settings Editor will send a command to the device to revert to its factory default settings. The reverted values will not take effect until the final changes have been applied. This button will remain disabled if the device does not support the DevConfig protocol messages.

If, after changing a setting or clicking the **Summary** button, you clicked **Save** on the summary screen to save the configuration, you can use the **Read File** button to load those settings. The settings from the saved file are immediately sent to the device and, if they're accepted, you can then **Apply** them.

OV4. Quick Start Tutorial

OV4.1 Software Products for the CR3000

PC200W Starter Software supports a direct connection between the PC and the CR3000, and includes Short Cut for Windows (Short Cut) for creating CR3000 programs. PC200W provides basic tools for setting the datalogger's clock, sending a program, monitoring sensors, and manually collecting and viewing data. CR3000 support was added to PC200W in Version 3.1. PC200W is available at no charge from the Campbell Scientific website.

PC400 Datalogger Support Software (mid-level software) supports a variety of telecommunication options, manual data collection, and data display. PC400 includes Short Cut and the CRBasic Program Editor for creating CR3000 programs. PC400 does not support combined communication options (e.g., phone-to-RF), PakBus® routing, or scheduled data collection. CR3000 support was added to PC400 in Version 1.3.

LoggerNet Datalogger Support Software (full-featured software) supports combined telecommunication options, data display, and scheduled data collection. The software includes Short Cut and CRBasic for creating CR3000 programs, and tools for configuring, trouble-shooting, and managing datalogger networks. CR3000 support was added to LoggerNet in Version 3.2.

OV4.1.1 Options for Creating CR3000 Programs

1. Short Cut is a program generator that creates a datalogger program in four easy steps, and a wiring diagram for the sensors. Short Cut supports the majority of sensors sold by Campbell Scientific, and is recommended for creating straightforward programs that measure the sensors and store data.
2. The CRBasic Editor is a program editor used to create more complex CR3000 programs. Short Cut generated programs can be imported into the CRBasic Editor for adding instructions, or for functionality not supported by Short Cut.

For those users of CR23X dataloggers who are switching to CR3000 dataloggers, the Transformer Utility can be used to convert a CR23X program to a CR3000 program, which can be imported into the CRBasic Editor. Because of differences in program code, not all CR23X programs can be fully converted by the Transformer. The Transformer Utility is included with PC400 and LoggerNet software.

OV4.2 Connections to the CR3000

Campbell Scientific Power Supplies are described in Section 1.3. When connecting an external power supply to the CR3000, first remove the green power connector from the CR3000 front panel. Insert the positive 12V lead into the terminal labeled “12V”, and the ground lead into the terminal labeled “G”. Double-check the polarity before plugging the green connector into the panel. For a CR3000 with an internal alkaline battery pack or rechargeable battery base, the green power plug is not used.

Connect the white serial cable (PN 10873, provided) between the port labeled “RS232” on the CR3000 and the serial port on the computer. For computers that have only a USB port, a USB Serial Adaptor (PN 17394 or equivalent) is required.

OV4.3 Setting the CR3000 PakBus Address

The CR3000 default PakBus address is 1 (Section OV2.5). Unless the CR3000 is used in a network, there is no need to change the Pakbus address, or any of the other default settings. To change settings, the Device Configuration Utility (DevConfig) is used, as described in Section OV3.

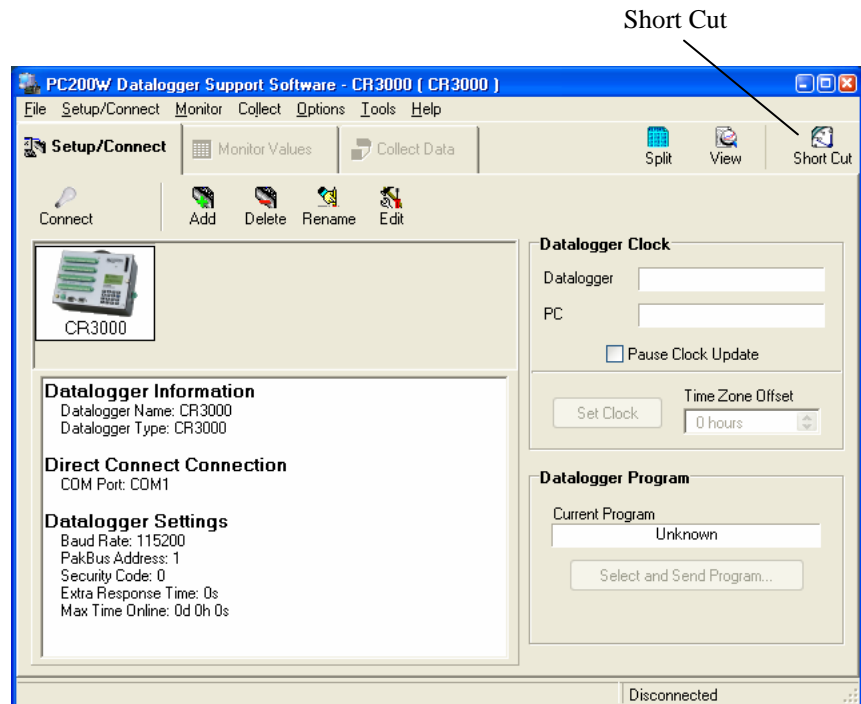
OV4.4 PC200W Software

This Quick-Start tutorial prompts the user through the process of programming the CR3000, monitoring sensor measurements, collecting data, and viewing data using the PC200W software.

When PC200W is first started, the EZSetup Wizard is launched. Click the **Next** button and follow the prompts to select the **CR3000**, the **COM** port on the computer that will be used for communications, **115200** baud, and **Pakbus Address 1**. When prompted with the option to **Test Communications** click the **Finish** button.

To change a setting in the datalogger setup, select that datalogger from the main window, and click the **Edit** button. If a datalogger was not added with the Wizard, click the **Add** button to invoke the Wizard.

After exiting the EZSetup wizard, the **Setup/Connect** window appears, as shown below. The Current Datalogger Profile, Datalogger Clock, and Datalogger Program features of PC200W are integrated into this window. Tabs to the right are used to select the **Monitor Values** and **Collect Data** windows. Buttons to the right of the tabs are used to run the **Split**, **View**, and **Short Cut** applications.



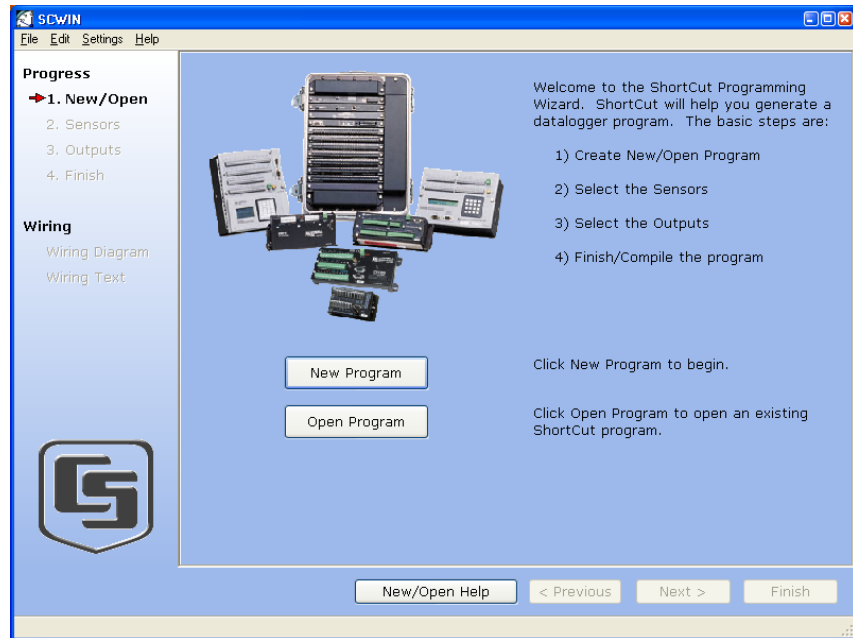
OV4.4.1 Creating a CR3000 Program using Short Cut

Objective: Every one second, measure air temperature in degrees C with a Type T thermocouple, and store one-minute average Battery Voltage, Panel Temperature, and Thermocouple temperature.

NOTE

A Type T Thermocouple is included with CR3000, packaged with the screwdriver. The thermocouple consists of a pair of 5-inch wires with blue/red insulation, soldered together at one end.

Click on the **Short Cut** button to display the **Home** screen, as shown below.



Each of the four steps has a button with a ? for accessing Help. Use the Help in conjunction with the steps outlined below:

Step 1: Create a New File

Step 1 is to open a new or existing file. From the **Home** page, click the **New Program** button. Use the drop-down list box to select the **CR3000** and click on OK. Enter a 1 second in the Scan Interval window and click **OK** to complete Step 1.

Step 2: Select the Sensors

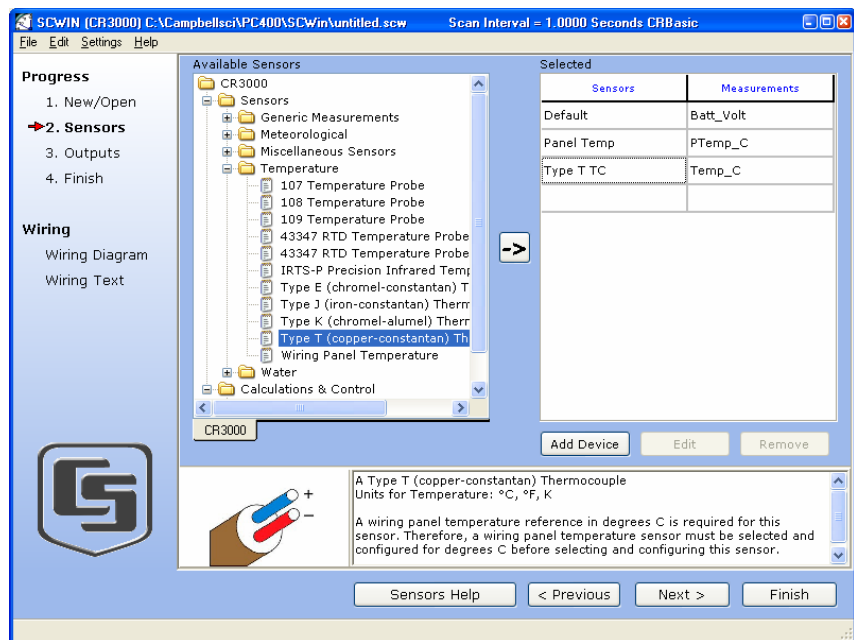
A Type T thermocouple consists of two wires of dissimilar metals (copper and constantan) soldered together at one end. The soldered end is the measurement junction; the junction that is created when the thermocouple is wired to the CR3000 is the reference junction.

When the two junctions are at different temperatures, a voltage proportional to the temperature difference is induced into the wires. The thermocouple measurement requires the reference junction temperature to calculate the measurement junction temperature.

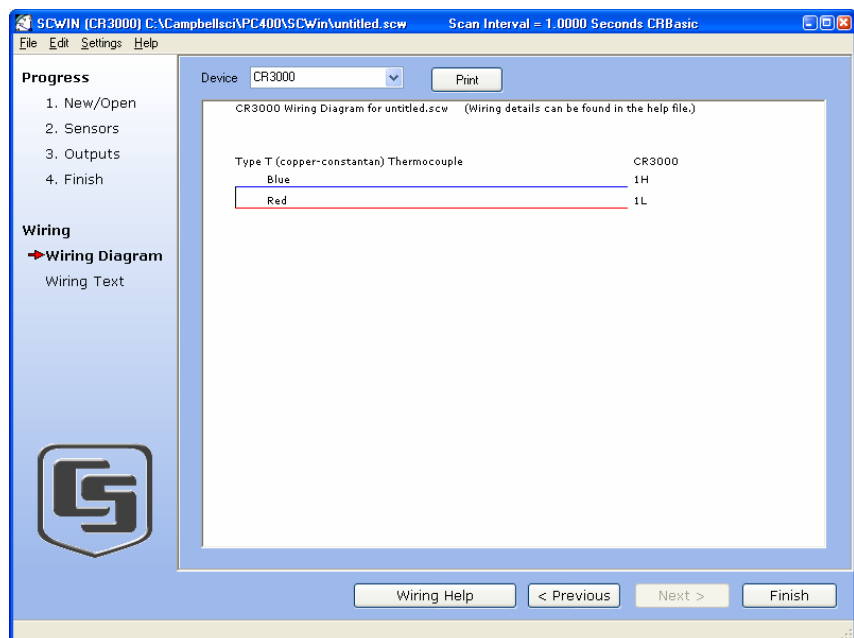
Step 2 is to select the sensors to be measured. The Sensors worksheet is divided into two sections: the Available sensors tree and the Selected sensors table, as shown below. The sensors you want to measure are chosen from the Available sensors tree.

Double click on the **Temperature** application group to display the available sensors. Double click on the **Wiring Panel Temperature** sensor to add it the selected sensors table. Click **OK** on the next screen to accept the PTemp_C label.

Double click on the **Type T thermocouple**, change the number to 1 and click **OK**. On the next screen, make sure Ptemp_C is selected for the Reference Temperature Measurement, and click **OK** to accept the Temp_C label.



Click on the **Wiring Diagram** link to view the sensor wiring diagram, as shown below. Wire the Type T Thermocouple (provided) to the CR3000 as shown on the diagram.



Step 3: Output Processing

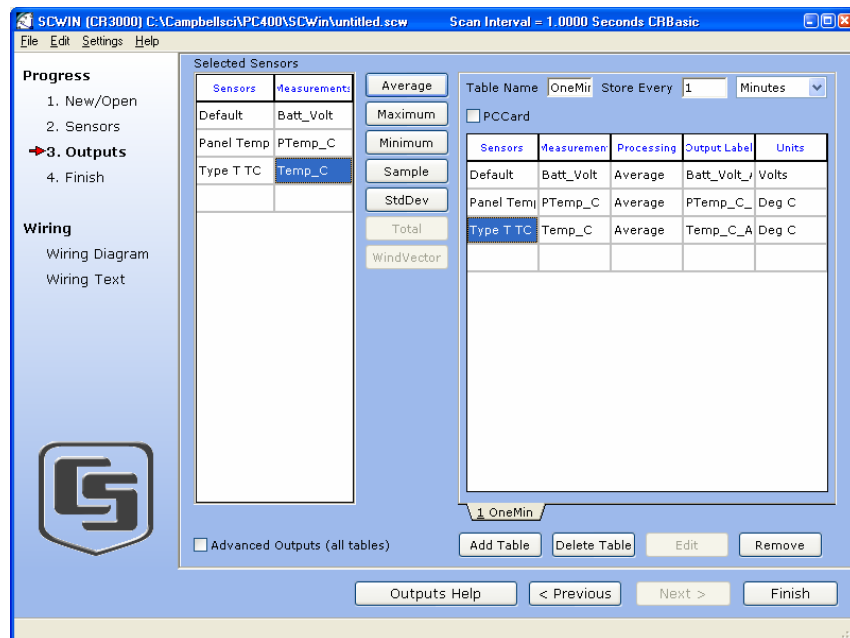
Step 3 is to define the output processing for the sensor measurements. Click the **Outputs** link in the **Progress** menu at the left.

The Outputs screen has a list of Selected Sensors on the left, and Output Tables on the right. The default is for two Tables, Table1 and Table2. Both Tables have a **Store Every** field and the drop-down list box that are used to set the interval at which data will be stored.

The objective for this exercise calls for a one-minute output processing. To remove Table2, Click on the **Table2** tab to activate it, and click the **Delete Table** button. When the **Confirm** window appears click **Yes**.

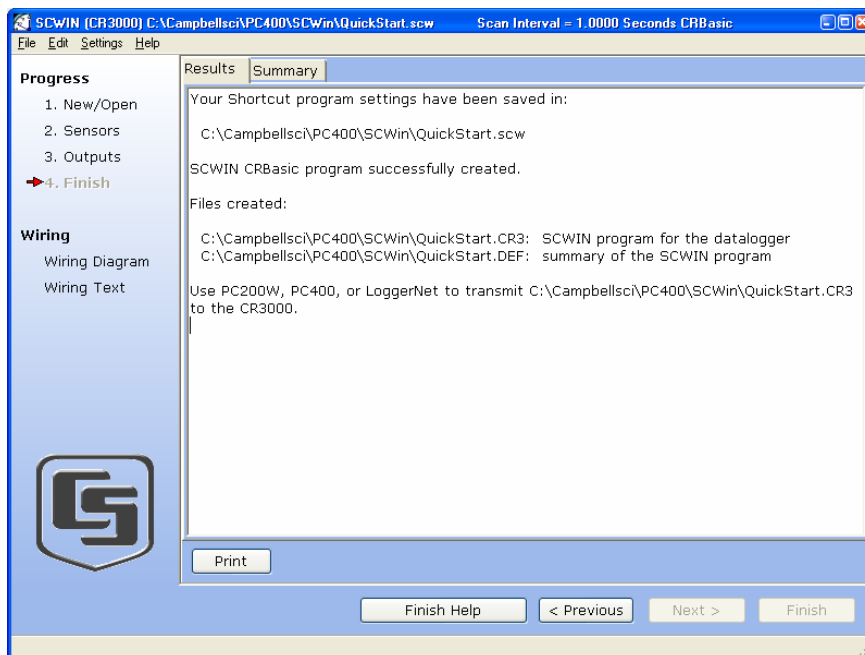
The **Table Name** field is the name that will be used for the Table in which the output will be stored. Change the default Name of Table1 to OneMin, and change the Store Every Field to 1 minute.

The Selected Sensors list is provided on the left side of the screen. To add a sensor measurement to the Output Table, highlight a measurement and click one of the output buttons; e.g., Average. Select the Default, Panel Temp, and Type T TC sensors and click the **Average** button to add them to the OneMin Table.



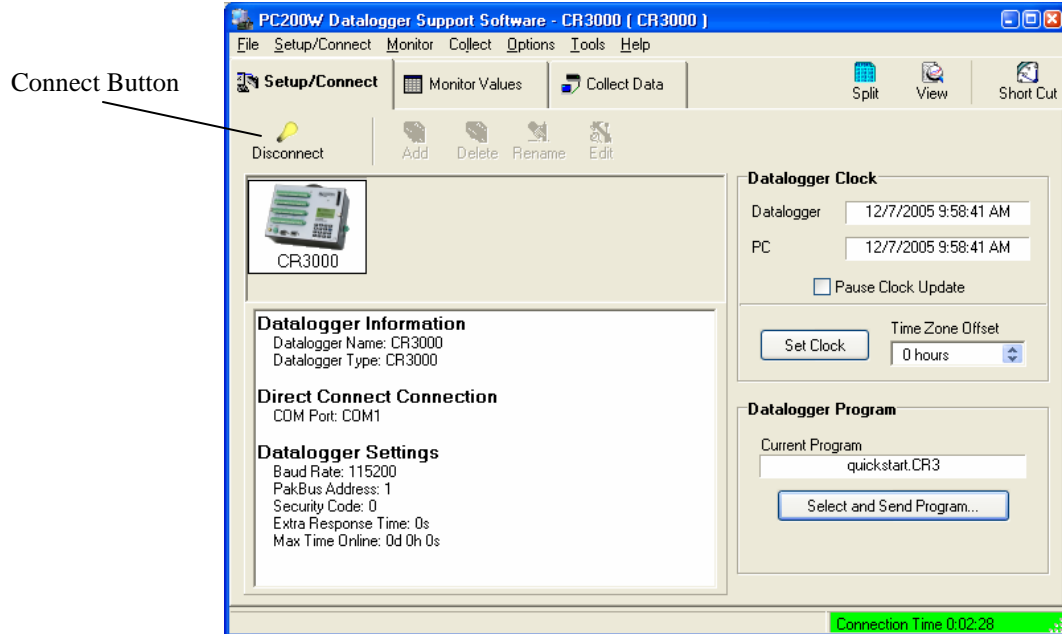
Step 4: Finish

Click the **Finish** link in the **Progress** menu on the left side to complete the program. Type in QuickStart for the file name. Any errors the compiler may have detected are displayed, along with the names of the files that were created. The file QuickStart.CR3 is the program file that will be sent to the CR3000, QuickStart.def is a summary of the sensor wiring and measurement labels (click the **Summary** tab or **Print** buttons to view or print the file). Now close Short Cut by clicking on the **X** box at the upper right, keying in **Alt+F4**, or clicking on **File | Exit**.



OV4.4.2 Configuring the Setup Tab

From the **Setup/Connect** screen, click on the **Connect** button to establish communications with the CR3000. When communications have been established, the text on the button will change to **Disconnect**.



OV4.4.3 Synchronize the Clocks

Click the **Set Clock** button to synchronize the datalogger's clock with the computer's clock.

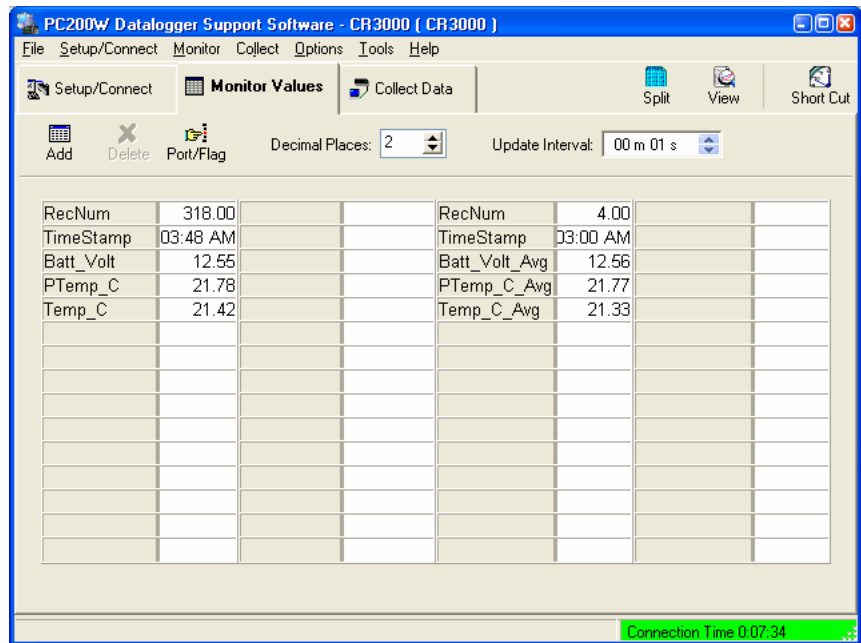
OV4.4.4 Send the Program

Click the **Select and Send Program** button. Navigate to the C:\CampbellSci\SCWin folder and select the file QuickStart.CR1 and click the **Open** button. A progress bar is displayed, followed by a message that the program was successfully sent.

OV4.4.5 Monitor Data Tables

The Monitor Values window is used to display the current sensor measurement values from the Public Table, and the most recent data from the OneMin Table.

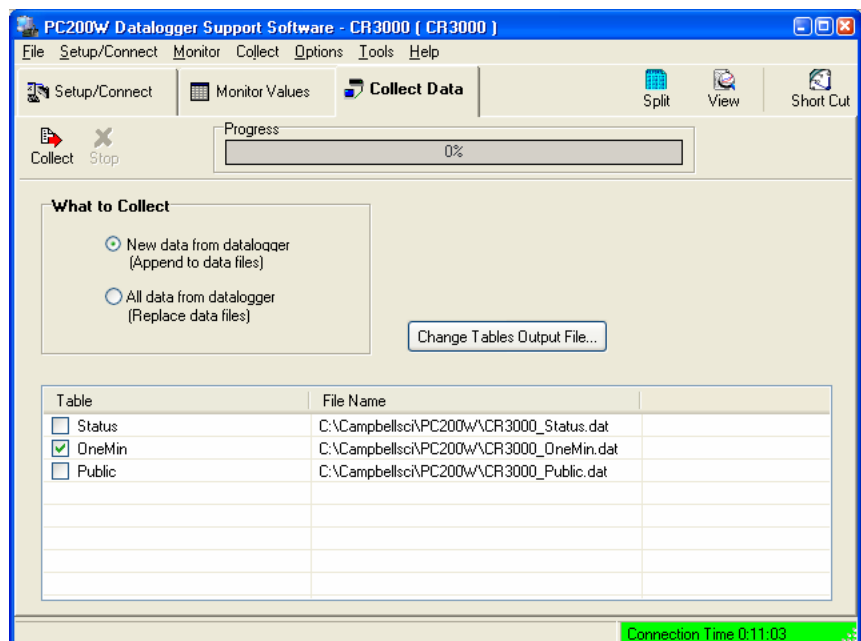
Click on the **Monitor Values** tab. The Public Table is automatically selected and displayed. To view the OneMin Table, click the **Add** button, select the **OneMin** Table, and click the **Paste** button.



OV4.4.6 Collect Data

Click on the **Collect Data** tab. From the Collect Data window you can choose what data to collect, and where to store the retrieved data.

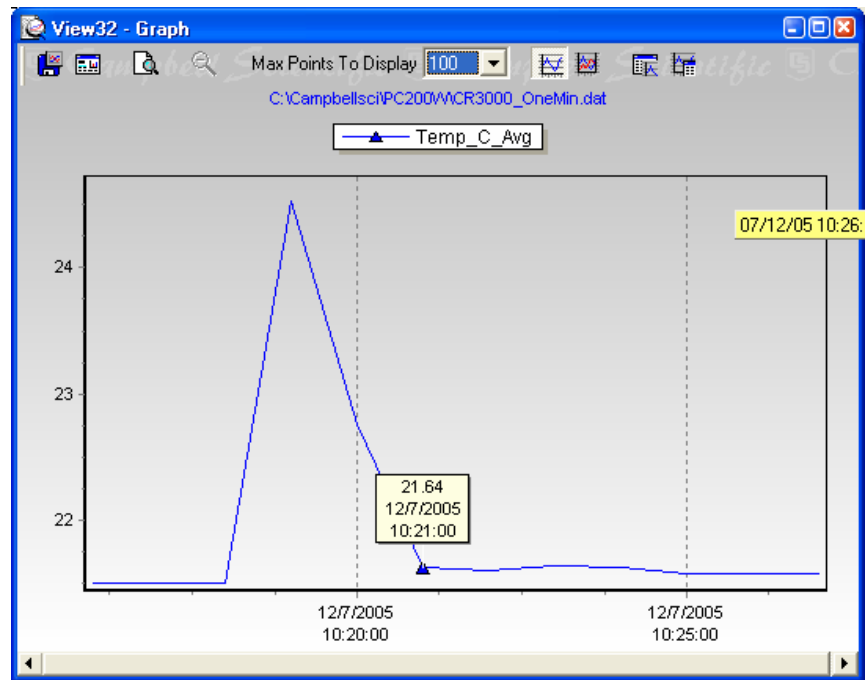
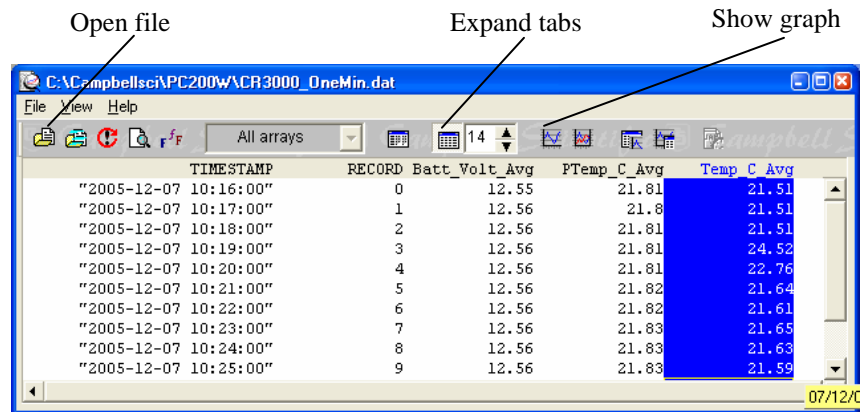
Click on the **OneMin** Table, with the Option **New data from datalogger** selected. Click the **Collect** button and a dialog box appears, prompting for a file name. Click the **Save** button to use the default file name CR3000_OneMin.dat. A progress bar, followed by the message **Collection Complete** is displayed.



OV4.4.7 View Data

To view the collected data, click on the **View** button (located in the upper right hand corner of the main screen). Options are accessed by using the menus or by selecting the toolbar icons. If you move and hold the mouse over a toolbar icon for a few seconds, a brief description of that icon's function will appear.

To open a data file, click the **Open file** icon, and double click on the file CR3000_OneMin.dat in the PC200W folder. Click the **Expand Tabs** icon to display the data in columns with column headings. To graph thermocouple temperature, click on the data column with the heading Temp_C, then click the **Show Graph, 1 Y axis** icon on the toolbar.



Close the graph and view screens, and close PC200W.

OV4.5 Programming using the CRBasic Program Editor

Those users who are moving from the Edlog Program Editor to the CRBasic Program Editor may find Short Cut to be an excellent way to learn CRBasic. First create a program using Short Cut, then open the file with CRBasic to see how Short Cut created the program. The program file listed below is the Short Cut file QuickStart.CR3 from the tutorial after being imported into the CRBasic editor.

See Section 4 for information on the CRBasic programming.

```
'CR3000
'Created by SCWIN (2.5)

'Declare Variables and Units
Public Batt_Volt
Public PTemp_C
Public Temp_C

Units Batt_Volt=Volts
Units PTemp_C=Deg C
Units Temp_C=Deg C

'Define Data Tables
DataTable(OneMin,True,-1)
DataInterval(0,1,Min,10)
Average(1,Batt_Volt,FP2,False)
Average(1,PTemp_C,FP2,False)
Average(1,Temp_C,FP2,False)
EndTable

'Main Program
BeginProg
Scan(1,Sec,1,0)
    'Default Datalogger Battery Voltage measurement Batt_Volt:
    Battery(Batt_Volt)
    'Wiring Panel Temperature measurement PTemp_C:
    PanelTemp(PTemp_C,_60Hz)
    'Type T (copper-constantan) Thermocouple measurements Temp_C:
    TCDiff(Temp_C,1,mV20C,1,TypeT,PTemp_C,True,0,_60Hz,1,0)
    'Call Data Tables and Store Data
    CallTable(OneMin)
NextScan
EndProg
```

OV5. Keyboard Display

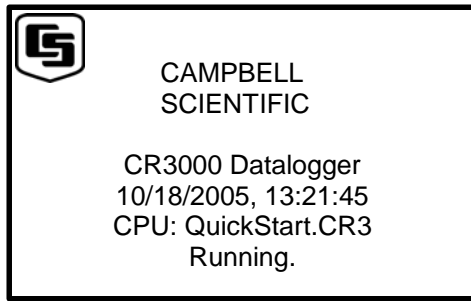
The CR3000 has an onboard keyboard display. This section illustrates the use of the keyboard display.

The CR3000KD has a few keys that have special functions which are listed below.

Key	Usage
[2] and [8]	To navigate up and down through the menu list one line at a time
[4] and [6]	To navigate left and right through a line. The left arrow [4] will display the full name (up to 20 characters) of any field name that won't fit into the 14 character space it normally uses. The right arrow [6] will display full data.
[Enter]	Selects the line or toggles the option of the line the cursor is on
[Esc]	Backs up one level in the menu
[Home]	Moves cursor to top of the list
[End]	Moves cursor to bottom of the list
[Pg Up]	Moves cursor up one screen
[Pg Dn]	Moves cursor down one screen
[BkSpc]	Delete character to the left
[Shift]	Change alpha character selected
[Num Lock]	Change to numeric entry
[Del]	Delete
[Ins]	Insert/change graph setup
[Graph]	Graph

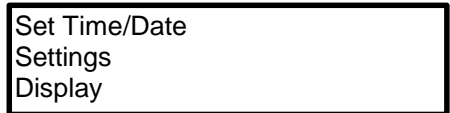
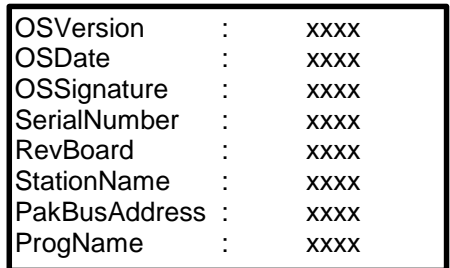
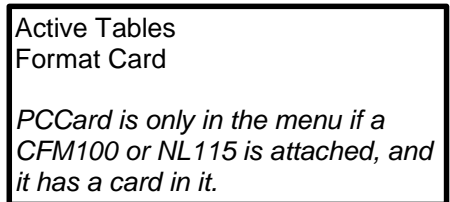
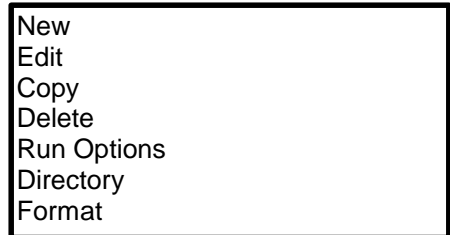
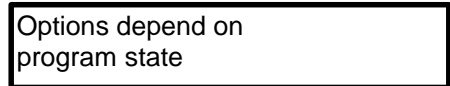
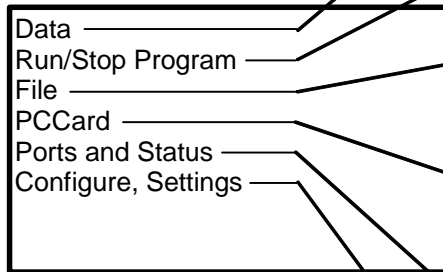
Power Up Screen

CR3000 Display

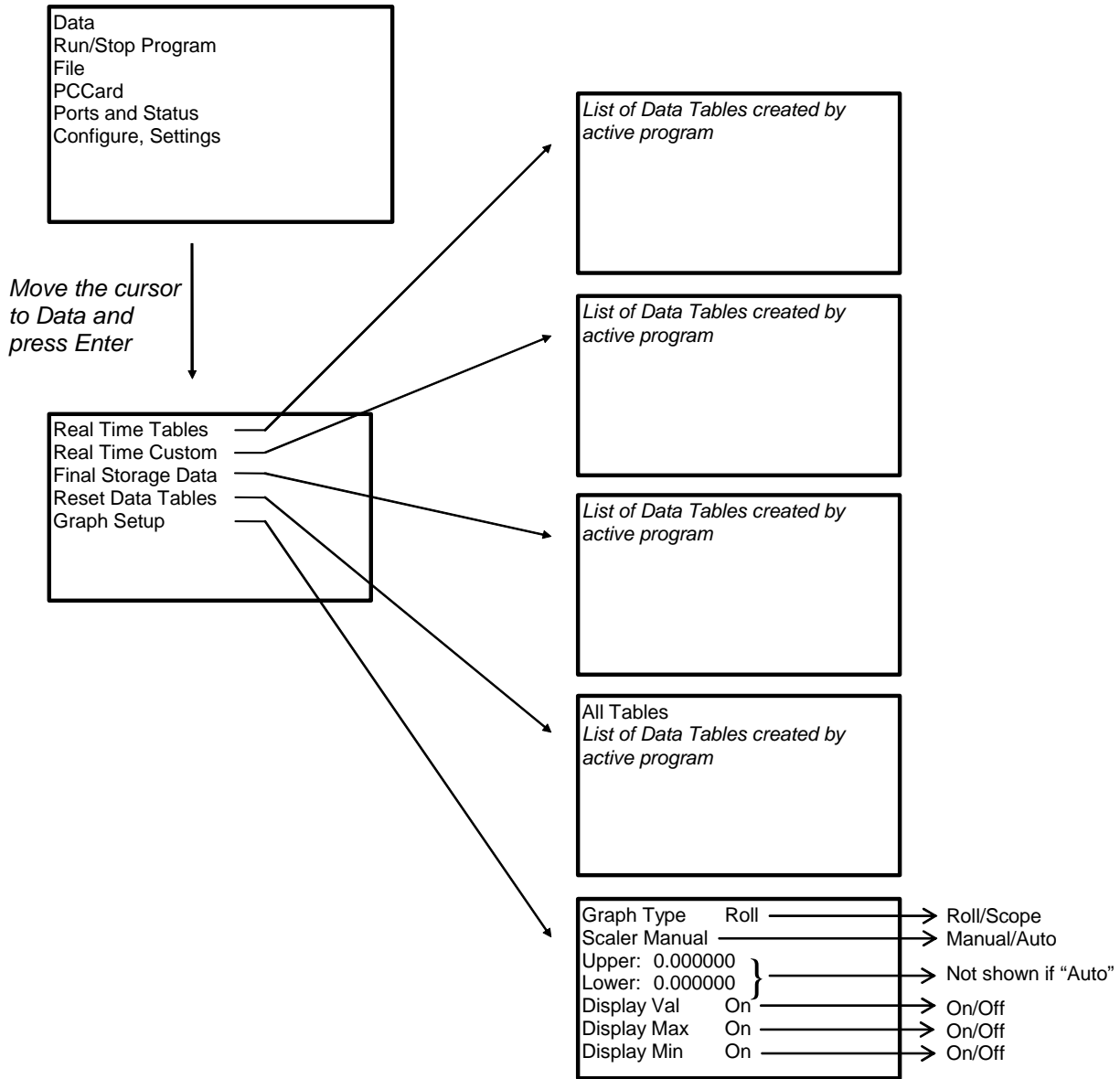


*Toggle backlight with ^
Adjust contrast with < >
< lighter darker >*

*Press any key
for Main Menu
(except <, >, ^,
or [Esc])*



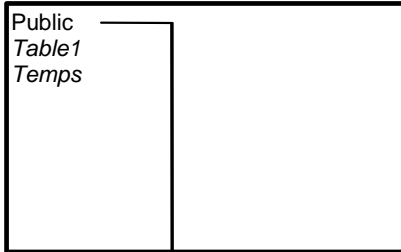
OV5.1 Data Display



Scope requires manual scalar

OV5.1.1 Real Time Tables

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



Move the cursor to desired table and press Enter

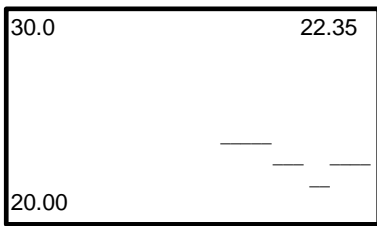
Tref	: 23.0234
TCTemp(1)	: 19.6243
TCTemp(2)	: 19.3429
	21.2003
Flag(1)	: -1.0000
Flag(2)	: 0.00000
Flag(3)	: 0.00000
Flag(4)	: 0.00000

Public Table values can be changed. Move the cursor to value and press Enter to edit value.

Edit field: Num
 TCTemp(3)
 Current Value:
 21.2003
 New Value:

Move the cursor to setting and press Enter to change

Press Graph for graph of selected field



Press Ins for Graph Setup

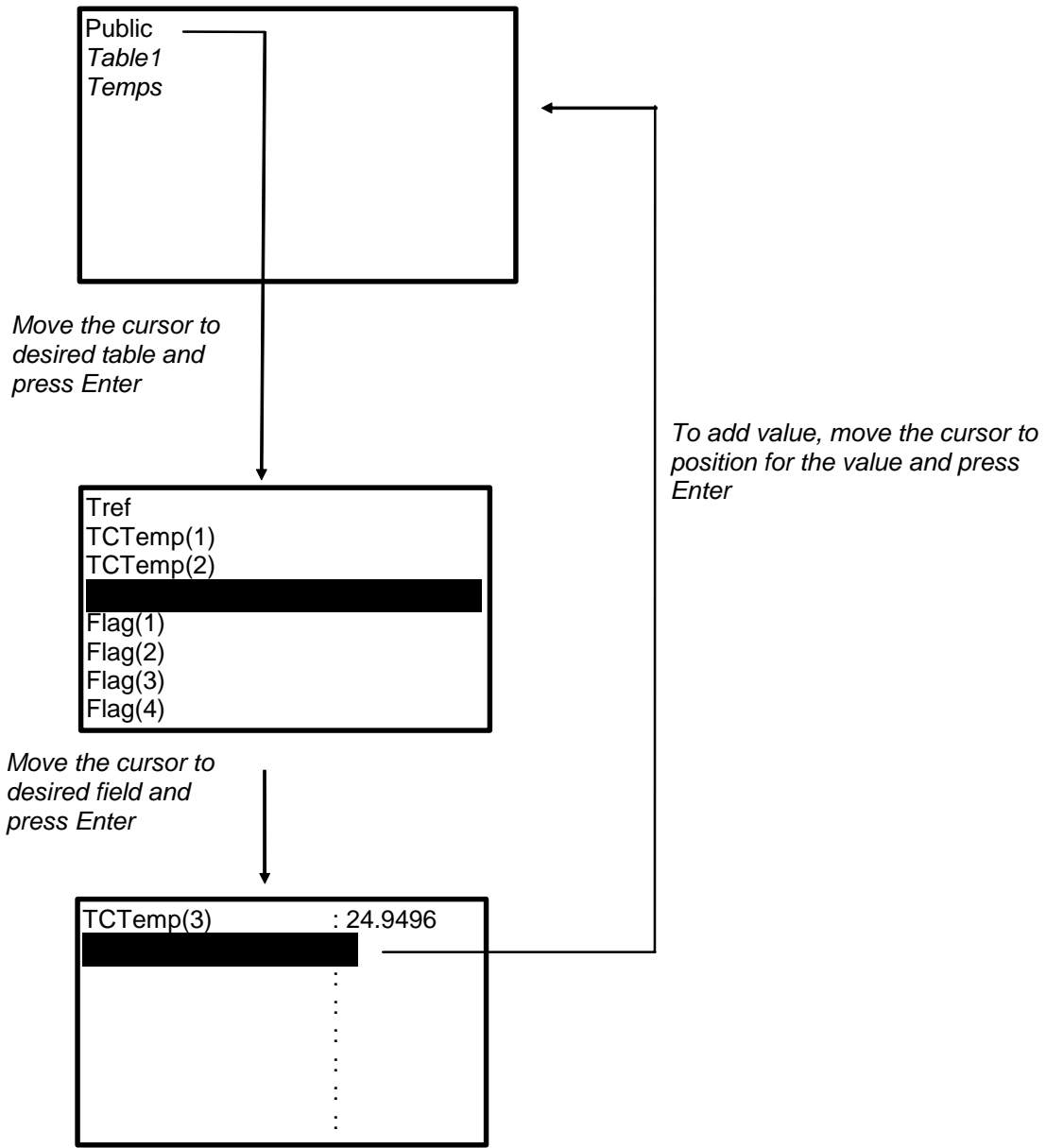
Scaler Manual	→	Auto/Manual
Upper: 30.000000		
Lower: 20.000000		
Display Val	On	→ On/Off
Display Max	On	→ On/Off
Display Min	On	→ On/Off
Graph Type	Roll	→ Roll/Scope

New values are displayed as they are stored.

OV5.1.2 Real Time Custom

The first time you navigate to Real Time Custom you will need to set up the display. The CR3000 will keep the setup as long as the same program is running.

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

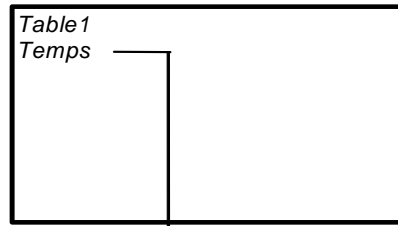


New values are displayed as they are stored.

To delete a field, move the cursor to that field and press Del

OV5.1.3 Final Storage Tables

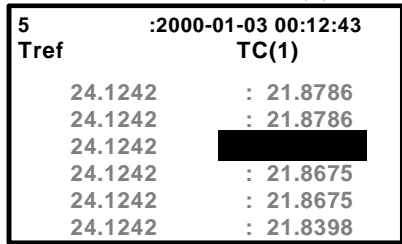
List of Data Tables created by active program. For Example:



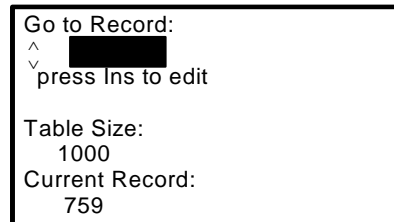
Move the cursor to desired Table and press Enter

Use Home (oldest), End (newest), PgUp (older), PgDn (newer), ←, →, ↑, and ↓ to move around in data table.

TimeStamp	Record	Tref	TC(1)	TC(2)	TC(3)
"2000-01-03 00:12:38"	0			21.934	22.8419
"2000-01-03 00:12:39"	1			21.9173	22.8364
"2000-01-03 00:12:40"	2			21.9229	22.8364
"2000-01-03 00:12:41"	3			21.9173	22.8419
"2000-01-03 00:12:42"	4			21.9173	22.8253
"2000-01-03 00:12:43"	5			21.9118	22.8364
"2000-01-03 00:12:44"	6			21.9173	22.8087
"2000-01-03 00:12:45"	7			21.9173	22.8142
"2000-01-03 00:12:46"	8			21.9395	22.8253
"2000-01-03 00:12:47"	9			21.9118	22.8308
"2000-01-03 00:12:48"	10			21.945	22.8364
"2000-01-03 00:12:49"	11			21.9506	22.8364



Press Ins for Jump To screen.

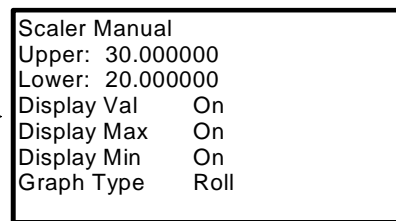


Use arrow up or down to scroll to the record number wanted, or press Ins and manually type in the record number.

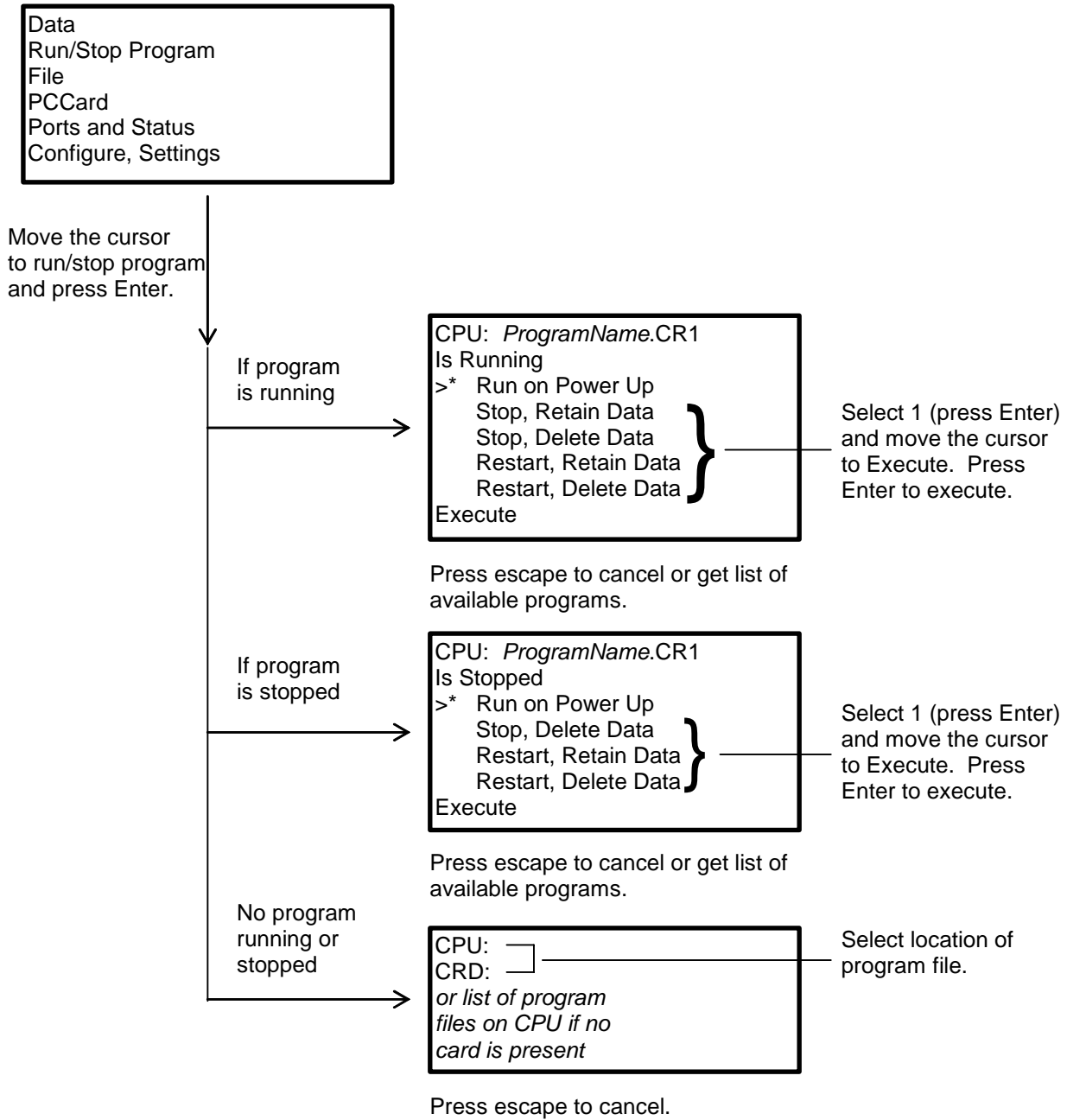
Press Graph for graph of selected field or for full screen display of string data. Use ←, →, PgUp, PgDn to move cursor and window of data graphed.



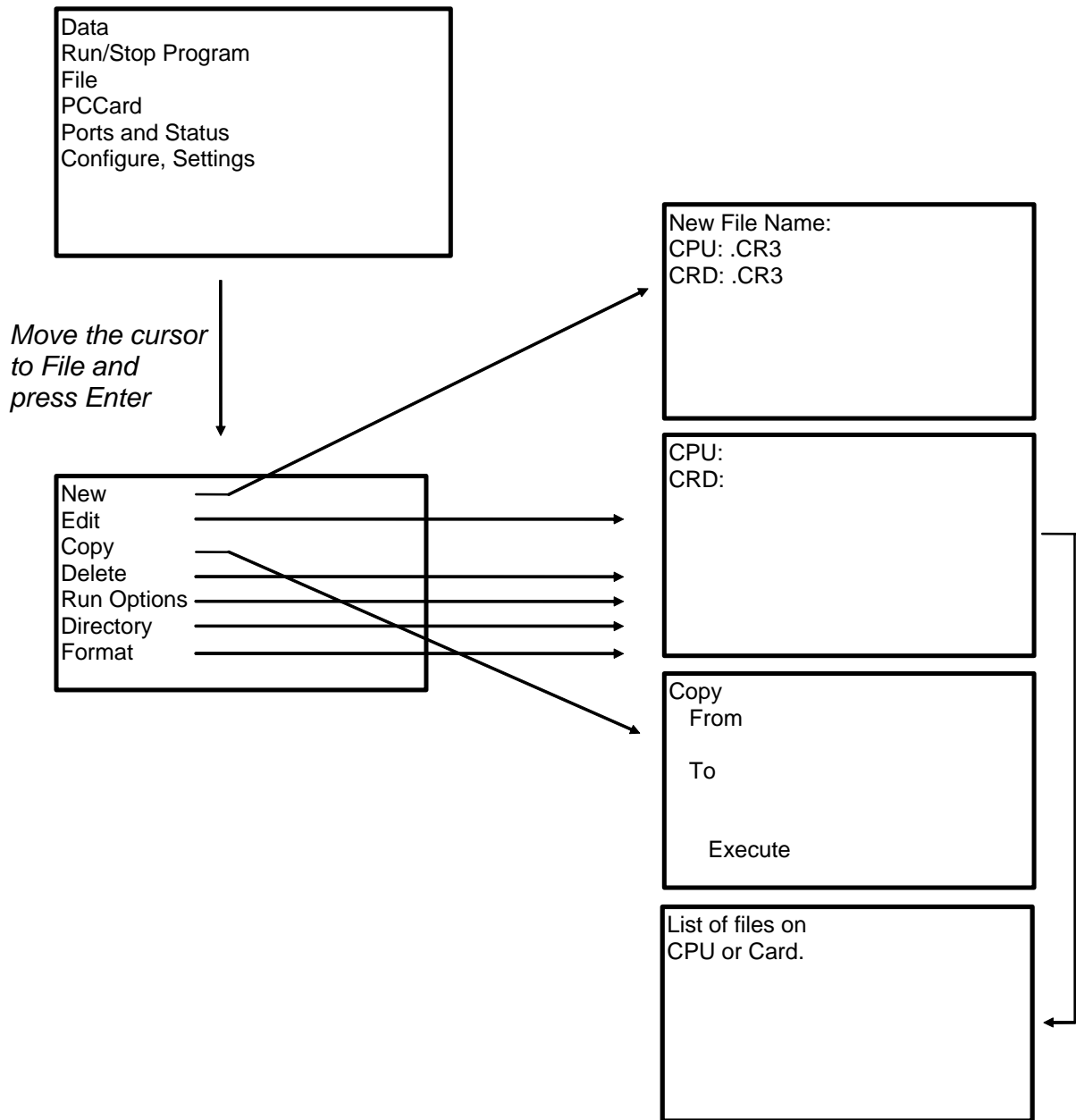
Press Ins for Graph Setup



OV5.2 Run/Stop Program



OV5.3 File Display



OV5.3.1 File: Edit

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

List of Program files on CPU: or
CRD: For Example:

```
CPU:
TCTEMP.CR3      0
RACE.CR3       0
```

```
Save Changes?
Yes
No
```

Move the cursor to desired Program and press Enter

```
CR3000
' TCTemp.CR3

Public TREF,TC(3),FLAG(8)

DataTable (Temps,1,1000)
Sample (1,TREF,IEEEE4)
Sample (3,TC(),IEEEE4)
```

```
INSERT
Instruction
Function
Blank Line
Block
Insert Off
```

Edit Directly or move cursor to first character of line and press Enter

Edit Instruction parameters with parameter names and some pick lists:

```
ENTER
Edit Instruction
Blank Line
Create Block
```

```
DataTable
TableName
> Temps
TrigVar
1
Size
1000
```

Insert blank line

```
Sample (1,TREF,IEEEE4)
Sample (3,TC(),IEEEE4)
EndTable

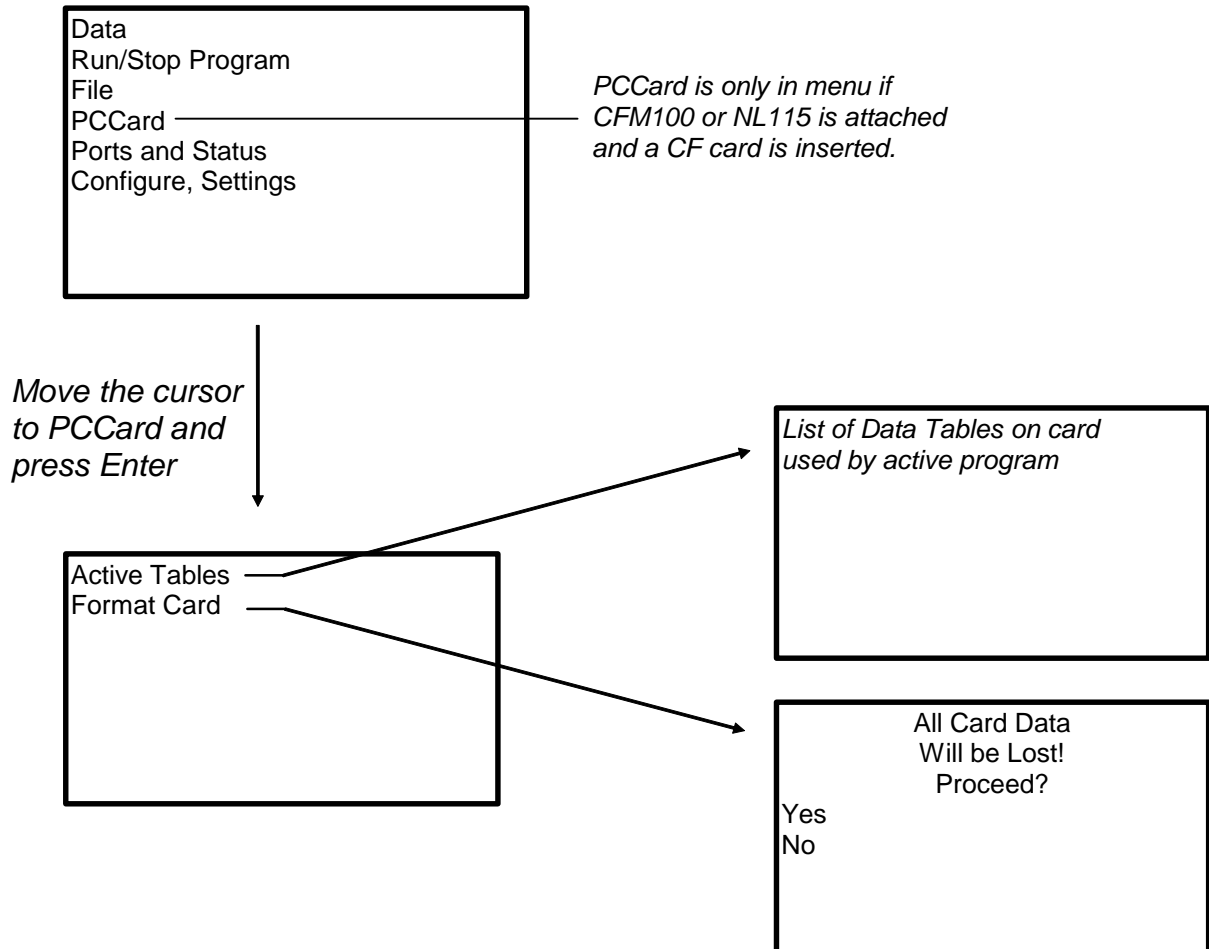
BeginProg
Scan(1,sec,3,0)
```

Move the cursor to highlight desired block and press Enter

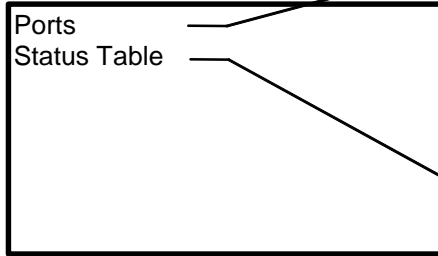
```
Block Commands
Copy
Cut
Delete
```

To insert a block created by this operation, move the cursor to desired place in program and press Ins.

OV5.4 PCCard Display



OV5.5 Ports and Status

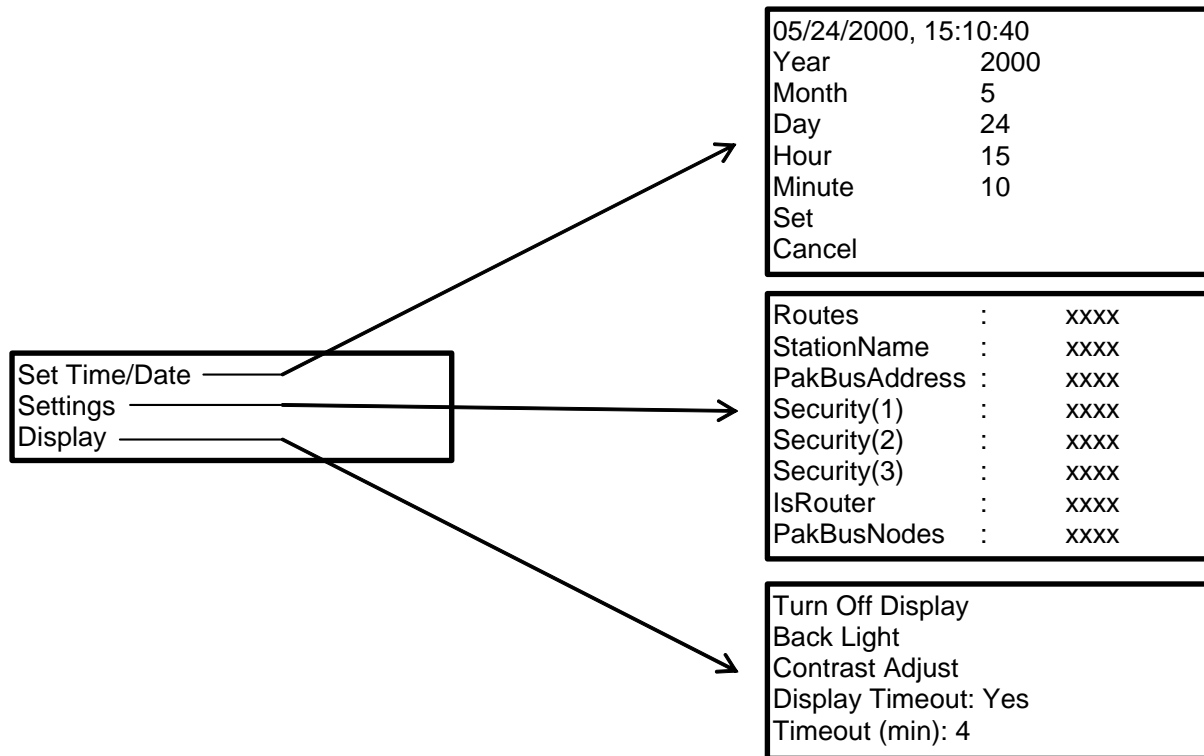


PortStatus (1): OFF
PortStatus (2): OFF
PortStatus (3): OFF
PortStatus (4): OFF
PortStatus (5): OFF
PortStatus (6): OFF
PortStatus (7): OFF
PortStatus (8): OFF

Move the cursor to the desired port and press Enter to toggle OFF/ON. The port must be configured as an output to be toggled.

*List of Status Variables
(see Appendix A)*

OV5.6 Settings



Move the cursor to time element and press Enter to change

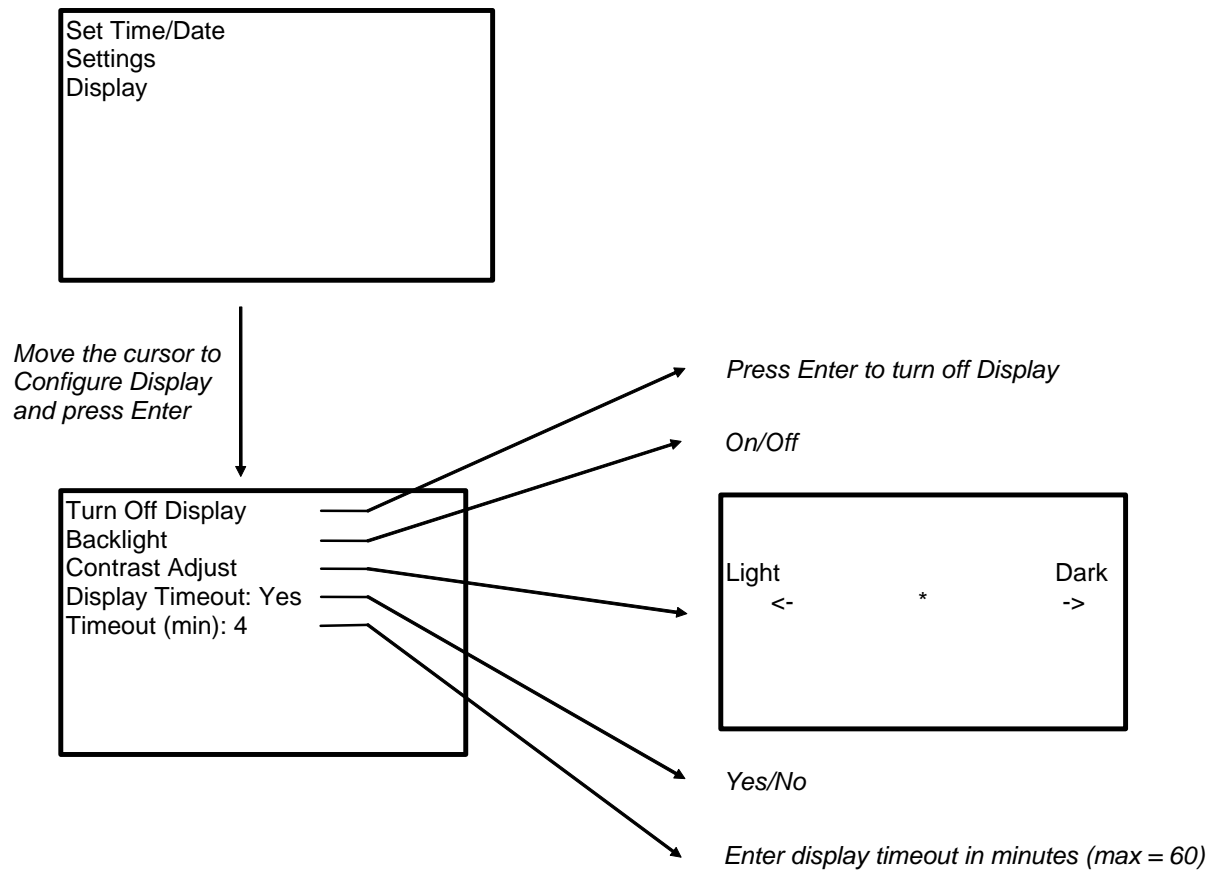
OV5.6.1 Set Time/Date

Move the cursor to time element and press Enter to change it. Then move the cursor to Set and press Enter to apply the change.

OV5.6.2 PakBus Settings

In the Settings menu, move the cursor to the PakBus element and press Enter to change it. After modifying, press Enter to apply the change.

OV5.6.3 Configure Display



OV6. Specifications

Electrical specifications are valid over a -25° to +50°C range unless otherwise specified; non-condensing environment required. To maintain electrical specifications, Campbell Scientific recommends recalibrating dataloggers every two years.

ANALOG INPUTS (SE1-SE28 or DIF1-DIF14)

14 differential (DF) or 28 single-ended (SE) voltage measurements individually configured. Ratiometric resistive bridge, thermocouple, and period average (frequency) measurements also supported on all analog input channels. Channel expansion provided by AM16/32 and AM25T multiplexers.

RANGES, RESOLUTION: 16-bit basic resolution (Basic Res). Resolution of DF measurements with input reversal is half the Basic Res (17-bits).

Input Range (mV) ¹	DF Res (µV) ²	Basic Res (µV)
±5000	83.33	167
±1000	33.3	16.67
±200	3.33	6.67
±50	0.83	1.67
±20	0.33	0.67

¹Range overhead of ~9% exists on all ranges to guarantee that the full-scale range values will not cause overrange.

²Resolution of DF measurements with input reversal.

ACCURACY³:

±(0.04% of reading + offset), 0° to 40°C
 ±(0.07% of reading + offset), -25° to 50°C
 ±(0.09% of reading + offset), -40° to 85°C (-XT only)

³Accuracy does not include sensor and measurement noise. Offsets are defined as:

Offset for DF w/input reversal = 1.5-Basic Res + 1.0 µV
 Offset for DF w/o input reversal = 3-Basic Res + 2.0 µV
 Offset for SE = 3-Basic Res + 5.0 µV

MEASUREMENT SPEED: Time includes 250 µs for conversion to engineering units. For voltage measurements, the CR3000 integrates the input signal.

Integration Type	Integration Time	Settling Time	Measurement Standard	Total Time Input Rev.
250	250 µs	200 µs	~0.7 ms	~1.4 ms
60 Hz filter	16.67 ms	3 ms	~20 ms	~40 ms
50 Hz filter	20.00 ms	3 ms	~23 ms	~46 ms

INPUT NOISE VOLTAGE: For DF measurements with input reversal on ±20 mV input range; digital resolution dominates for higher ranges.

250 µs Integration: 0.4 µV RMS
 50/60 Hz Integration: 0.19 µV RMS

COMMON MODE RANGE: ±5 V

DC COMMON MODE REJECTION: >100 dB

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

SUSTAINED INPUT VOLTAGE W/O DAMAGE: ±16 VDC max.

INPUT CURRENT: ±1 nA typical, ±6 nA max. @ 50°C; ±120 nA @ 85°C

INPUT RESISTANCE: 20 Gohms typical

ACCURACY OF BUILT-IN REFERENCE JUNCTION

THERMISTOR (for thermocouple measurements):

±0.3°C, -25° to 50°C;
 ±0.8°C, -40° to 85°C (-XT only)

PERIOD AVERAGE MEASUREMENTS: Any of the 28 SE analog inputs can be used for period averaging. Accuracy is ±(0.01% of reading + resolution) where resolution is 96 ns divided by the specified number of cycles to be measured.

Input Amplitude & Frequency:

Volt. Gain	Range Code	Signal (peak to peak) Min (mV) Max (V) ⁴	Pulse W. Min. (µs) Max. (µs)	Max. Freq. (kHz) ⁵
1	mV1000	200 10	2.5 200	200
5	mV200	20 2	5.0 100	100
20	mV50	5 2	10.0 50	50
50	mV20	2 2	25.0 20	20

⁴Maximum signal must be centered at datalogger ground.
⁵Assuming 50% duty cycle.

ANALOG OUTPUTS (Vx1-Vx4, Ix1-Ix3, CAO1, CAO2)
 4 switched voltage and 3 switched current outputs for ratiometric sensor/bridge excitation and 2 continuous voltage outputs. Switched outputs active only during measurement, one at a time.

	Range	Res	Current Source/Sink	Compliance Voltage
Vx:	±5 V	0.17 mV	±50 mA	N/A
CAO:	±5 V	0.17 mV	±15 mA	N/A
Ix:	±2.5 mA	0.08 µA	N/A	±5 V

Vx & CAO ACCURACY:

±(0.04% of setting + 0.5 mV), 0° to 40°C
 ±(0.07% of setting + 0.5 mV), -25° to 50°C
 ±(0.09% of setting + 0.5 mV), -40° to 85°C (-XT only)

Ix ACCURACY:

±(0.1% of setting + 0.5 µA), 0° to 40°C
 ±(0.13% of setting + 0.5 µA), -25° to 50°C
 ±(0.15% of setting + 0.5 µA), -40° to 85°C (-XT only)

Vx FREQUENCY SWEEP FUNCTION: The switched outputs provide a programmable swept frequency, 0 to 5 V square wave for exciting vibrating wire transducers.

RESISTANCE MEASUREMENTS

MEASUREMENT TYPES: The CR3000 provides ratiometric measurements of 4- and 6-wire full bridges, and 2-, 3-, and 4-wire half bridges. Precise, dual polarity excitation for voltage or current excitations eliminates DC errors. Offset values are reduced by a factor of 2 when excitation reversal is used.

VOLTAGE RATIO ACCURACY¹: Assuming excitation voltage of at least 500 mV, and not including bridge resistor errors

±(0.02% of voltage reading + offset)/V_x, 0° to 40°C
 ±(0.025% of voltage reading + offset)/V_x, -25° to 50°C
 ±(0.03% of voltage reading + offset)/V_x, -40° to 85°C

¹Accuracy does not include sensor and measurement noise. Offsets are defined as:

Offset for DF w/input reversal = 1.5-Basic Res + 1.0 µV
 Offset for DF w/o input reversal = 3-Basic Res + 2.0 µV
 Offset for SE = 3-Basic Res + 5.0 µV

ACCURACY WITH CURRENT EXCITATION¹:

Assuming excitation current of at least 500 µA.
 ±(0.02% of voltage reading + offset)/I_x, 0° to 40°C
 ±(0.025% of voltage reading + offset)/I_x, -25° to 50°C
 ±(0.03% of voltage reading + offset)/I_x, -40° to 85°C

¹Accuracy does not include sensor and measurement noise. Offsets are defined as:

Offset for DF w/input reversal = 1.5-Basic Res + 1.0 µV
 Offset for DF w/o input reversal = 3-Basic Res + 2.0 µV
 Offset for SE = 3-Basic Res + 5.0 µV

DEDICATED PULSE COUNTERS (P1-P4)

Four inputs individually selectable for switch closure, high frequency pulse, or low-level AC. Independent 24-bit counters (16.8 x 10⁶ counts) for each input.

SWITCH CLOSURE MODE:

Minimum Switch Closed Time: 5 ms
 Minimum Switch Open Time: 6 ms
 Max. Bounce Time: 1 ms open w/o being counted

HIGH FREQUENCY PULSE MODE:

Voltage Thresholds: Count upon transition from below 0.9 V to above 2.2 V after input filter with 1.2 µs time constant.

Maximum Input Voltage: ±20 V
 Maximum Input Frequency: 250 kHz

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

Input Hysteresis: 16 mV @ 1 Hz
 Maximum AC Input Voltage: ±20 V
 Minimum ac Input Voltage:

Sine wave (mV RMS)	Range (Hz)
20	1.0 to 20
200	0.5 to 200
2000	0.3 to 10,000
5000	0.3 to 20,000

DIGITAL CONTROL PORTS (C1-C8, SDM)

8 digital control ports (C1-C8) having multiple function capability including digital control outputs, digital control interrupts, pulse counting, switch closure, frequency/period measurements, edge timing, and SDI-12 communication. Independent asynchronous communication ports (UARTs) paired on control port pairs C1-C2, C3-C4, C5-C6, and C7-C8 resulting in four independent Tx/Rx pair for serial sensors.

INPUT STATE: high 3.8 to 5.3 V; low -0.3 to 1.2 V

INPUT HYSTERESIS: 1.4 V

INPUT RESISTANCE: 100 kohms

HIGH FREQUENCY MAX: 400 kHz

SWITCH CLOSURE FREQUENCY MAX: 150 Hz

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

OUTPUT RESISTANCE: 330 ohms

ADDITIONAL DIGITAL PORTS: SDM-C1, SDM-C2, SDM-C3 are dedicated for measuring SDM devices.

SWITCHED 12 V (SW12V)

Two independent 12 V unregulated sources switched on and off under program control. Thermal fuse hold current = 900 mA @ 20°C, 650 mA @ 50°C, 360 mA @ 85°C.

CE COMPLIANCE

STANDARD(S) TO WHICH CONFORMITY IS DECLARED: IEC61326:2002

COMMUNICATION

RS-232 PORTS:

9-pin: DCE (electrically isolated) for computer or non-CSI modem connection

COM1 to COM4: Four independent Tx/Rx pairs on control ports (non-isolated)

Baud Rate: Selectable from 300 to 115.2 kbps.

Format: 7, 8 data bits; 1, 2 stop bits; odd, even, or no parity

CS I/O PORT: Interface with CSI peripherals.

SDI-12: Digital Control ports 1, 3, 5, and 7 are individually configurable and meet Standard version 1.3 for datalogger mode. Up to ten SDI-12 sensors are supported per port.

SDM PORT: Interface with CSI Synchronous Devices for Measurement

PERIPHERAL PORT: Interface with CFM100 CompactFlash Module

SYSTEM

PROGRAM EXECUTION INTERVALS: 10 ms to 30 min. @ 10 ms increments

PROCESSOR: Renesas H8S 2674 (16-bit CPU with 32-bit internal core)

MEMORY: 2 Mbytes of Flash for operating system; 4 Mbytes of battery-backed SRAM for CPU usage, program storage and data storage

CLOCK ACCURACY: ±3 min. per year

SYSTEM POWER REQUIREMENTS

VOLTAGE: 10 to 16 VDC

TYPICAL CURRENT DRAIN: Sleep Mode: 2 mA
 1 Hz Sample Rate (one fast SE meas.): 3 mA
 100 Hz Sample Rate (one fast SE meas.): 10 mA
 100 Hz Sample Rate (one fast SE meas. w/RS-232 communications): 38 mA

Display on: add 1 mA to current drain

Backlight on: add 42 mA to current drain

INTERNAL BATTERIES: 10 Ahr alkaline or 7 Ahr rechargeable base. 1200 mAh lithium battery for clock and SRAM backup typically provides 3 years of back-up.

EXTERNAL BATTERIES: 12 VDC nominal; reverse polarity protected.

PHYSICAL SPECIFICATIONS

SIZE: 9.5" x 7.0" x 3.8" (24.1 x 17.8 x 9.6 cm). Terminal strips extend 0.875" (2.2 cm) and terminal strip cover extends 1.575" (4.0 cm) above the panel.

WEIGHT: 3.6 lbs (1.6 kg) with low profile base; 8.3 lbs (3.8 kg) with alkaline base; 10.7 lbs (4.8 kg) with rechargeable base.

WARRANTY

Three years against defects in materials and workmanship.

Section 1. Installation and Maintenance

1.1 Protection from the Environment

The normal environmental variables of concern are temperature and moisture. The standard CR3000 is designed to operate reliably from -25°C 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 PC board may result. Effective humidity control is the responsibility of the user.

The CR3000 is not hermetically sealed. Two half unit packets of DESI PAK desiccant are located by the batteries. This desiccant should, in most cases, keep the interior components dry between battery changes. 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 CR3000).

Extra desiccant should also be placed in the enclosure to prevent corrosion on the Wiring Panel terminals and CR3000/Wiring Panel connections.

Campbell Scientific offers environmental enclosures for housing a CR3000 and associated peripherals. The enclosure back plates have a grid of square holes punched on one inch centers. The black CR3000 datalogger base has four holes at the corners used to mount the datalogger. Small rubber mounting feet are installed in the four holes when shipped from the factory. The four rubber feet are removed to access the four mounting holes. The holes are on a 6 by 8 inch spacing. An enclosure supply kit is shipped with the enclosures. In the supply kit, are plastic grommets and compatible screws. Place the grommets in appropriate square holes then attach the datalogger base to the enclosure back plate with the supplied screws.

The fiberglass enclosures are classified as NEMA 4X (watertight, 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. The enclosures are white for reflecting solar radiation, thus reducing the internal enclosure temperature.

The Model ENC 12/14 fiberglass enclosure houses the CR3000 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 CR3000 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 CR3000 operates at a nominal 12 VDC. Below 10 V or above 16 volts the CR3000 does not operate properly.

The CR3000 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 CR3000 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 CR3000 typically draws 2 mA in the sleep state (with display off), 3 mA with a 1 Hz sample rate, 10 mA with a 100 Hz sample rate, and 38 mA with a 100 Hz sample rate and RS232 communications.

CAUTION

The 12 V and switched 12 V terminals on the wiring panel are not regulated by the CR3000; they obtain power directly from the internal battery or Power In terminal. If you use the CR3000 wiring panel to source power to other 12 V devices, be sure the power supply regulates the voltage within the range acceptable to the connected device. The maximum voltage output by a Campbell Scientific power supply is approximately 16 V (the charging voltage at -40°C).

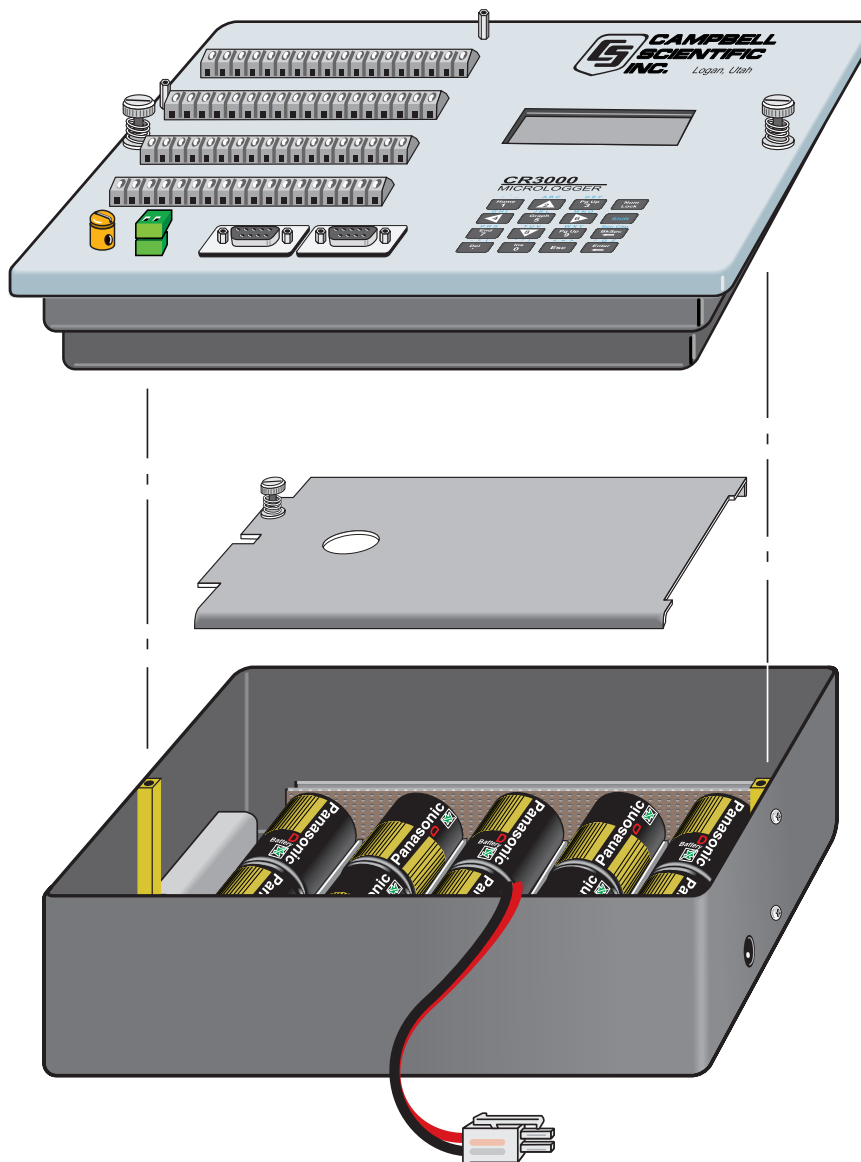
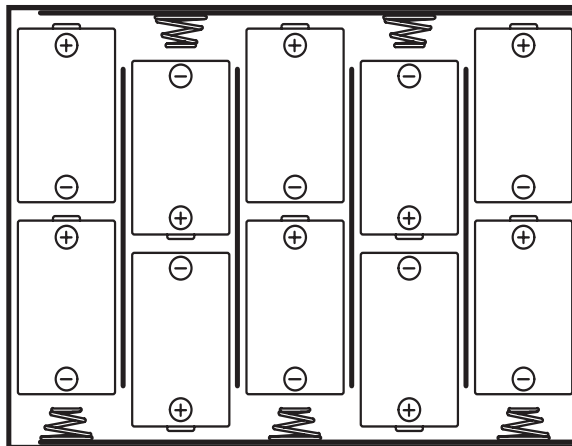


FIGURE 1.2-1. CR3000 Battery Pack and Panel

1.3 Campbell Scientific Power Supplies

The CR3000 is available with either alkaline or lead acid battery options. It may also be purchased without a battery option for connection to an external 12V power supply.

Each of the power supplies has a thermal fuse in the power circuit that limits source current. If excessive current is drawn, the fuse gets hot, increases in resistance, and limits current. When the problem is fixed, the fuse cools and the resistance decreases, eventually allowing current to pass. When excessive current is drawn due to shorting the power leads to the Wiring Panel, allow 10 to 15 seconds for the fuse to cool before connecting power.



10 D-TYPE ALKALINE BATTERIES

FIGURE 1.3-1. Alkaline Battery Orientation

While the CR3000 has a wide operating temperature range (-40 TO +80°C), the battery bases have more limited operating ranges. **Exceeding this range will degrade battery capacity and lifetime and could also cause permanent damage.**

1.3.1 CR3000 Alkaline Battery Base (Part Number 10519)

Install alkaline batteries as shown in Figure 1.3-1.

Suggested temperature range: -25° to +50°C (Table 1.3-1).

The CR3000 utilizes 10 alkaline D cells. After battery installation, the power switch on the side of the CR3000 will turn the datalogger on and off.

A fresh set of ten high-quality alkaline D cells can be expected to have about 10 amp hours at 20°C.

Voltage with ten new D cells is about 15.5 V. If external power is connected to the CR3000 with the alkaline power supply, power will be drawn from the source of highest voltage. In a typical configuration using an external lead-acid battery, the D cell voltage will discharge from 15 V to about 12 V before the lead-acid battery starts to supply current. This means the D cells will lose approximately 35% (15 Ah - 9.8 Ahr = 5.2 Ah) of their capacity before the external power takes over.

The amp-hour rating decreases with temperature as shown in Table 1.3-1. CRBasic keyword Battery can be used to monitor battery voltage (See section 7.6). Replace the alkaline cells before the CR3000 battery voltage drops below 11.0 V.

A thermal fuse in the alkaline base will allow a current drain of 1.85 A at 20°C (1.3 A at 50°C). In event of a short, the thermal fuse will cut power after about 40 seconds.

TABLE 1.3-1. Typical Alkaline Battery Service and Temperature	
Temperature (°C)	% of 20°C Service
20 - 50	100
15	98
10	94
5	90
0	86
-10	70
-20	50
-30	30

NOTE

This data is based on one "D" cell under conditions of 50 mA current drain with a 30 ohm load. As the current drain decreases, the percent service improves for a given temperature.

1.3.2 CR3000 Lead Acid Battery Base (Part Number 10518)

Temperature range: -40° to +60°C (Table 1.3-2).

Charging voltage: 17 to 24 VDC or 18 V RMS AC.

The CR3000 includes two 6V, 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 CR3000 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 1.3-2.

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 CR3000 on or off. Battery charging still occurs when the switch is off.

Should the lead acid batteries require replacement, consult Figure 1.3-2 for wiring.

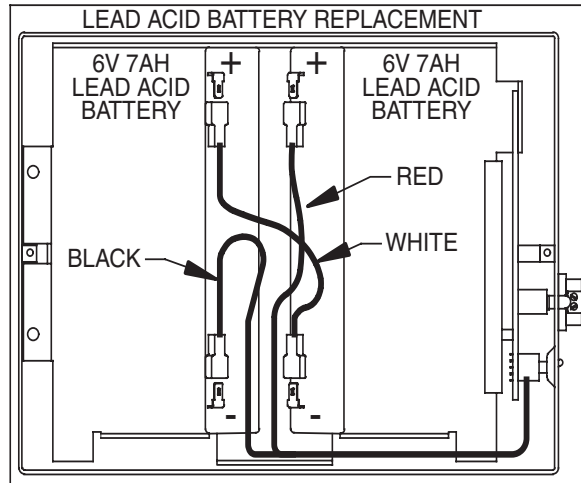


FIGURE 1.3-2. Lead Acid Battery Wiring

Monitor the power supply using CRBasic keyword Battery (Section 7.6). Users are strongly advised to incorporate this instruction into their 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. The keyword Battery measures the voltage internal to the CR3000, 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 CR3000 to measure the actual lead acid battery voltage.

TABLE 1.3-2. CR3000 Rechargeable Battery and AC Transformer Specifications	
Lead Acid Battery	
Battery Type	Yuasa NP7-6
Float Life @ 25°C	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	-15°C to 50°C charge -20°C to 60°C discharge
AC Transformer	
Input:	120 VAC, 60 Hz
Isolated Output:	18 VAC 1.11 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 makes the following recommendations:

1. A CR3000 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.

1.3.3 CR3000 Low Profile Base (Part Number 10695)

The low profile CR3000 option is not supplied with a battery base. In this configuration, the user supplies power to the CR3000 by attaching an external 12V power source to the Power In plug on the lower left of the CR3000 wiring panel. Campbell Scientific supplies three external power supplies that would work with the low profile configuration, the BPALK, PS100, and CH100.

1.3.3.1 BPALK Alkaline Power Supply

The BPALK uses 8 alkaline D cells and comes with an AA cell battery pack to supply power while replacing the D cells. The 4 AA batteries are not included.

To replace the batteries without stopping the datalogger program, 1) connect the external battery to the port labeled temporary, 2) remove the old batteries, 3) replace with new alkaline D cell batteries, and 4) remove the external battery.

A fresh set of eight alkaline D cells has 12.4 volts and a nominal rating of 7.5 amp-hours at 20°C. The amp-hour rating decreases with temperature as shown in Table 1.3-1. The CRBasic Battery instruction can be used to monitor battery voltage (see section 7.6). Replace the alkaline cells before the CR3000 battery voltage drops below 11 V.



FIGURE 1.3-3. BPALK Power Supply

1.3.3.2 PS100 Lead Acid Power Supply

The PS100 power supply includes a 12 V, 7.0 amp-hour lead acid battery, an AC transformer (18 V), and a temperature compensated charging circuit with a charge indicating diode. An AC transformer or solar panel should be connected to the PS100 at all times. The charging source powers the CR3000 while float charging the lead acid batteries. The internal lead acid battery powers the datalogger if the charging source is interrupted. The PS100 specifications are given in Table 1.3-3.

The two leads from the charging source can be inserted into either of the CHG ports, polarity doesn't matter. A transzorb provides transient protection to the charging circuit. A sustained input voltage in excess of 40 V will cause the transzorb to limit voltage.

The red light (LED) on the PS100 is on when a charging source is connected to the PS100 CHG ports. The switch turns power on and off to the 12 V ports, battery charging still occurs when the switch is off.

CAUTION

Switch the power to "off" before disconnecting or connecting the power leads to the Wiring Panel. The Wiring Panel and PS100 are at power ground. If 12 V is shorted to either of these, excessive current will be drawn until the thermal fuse opens.

Monitor the power supply voltage using the datalogger Battery instruction. Incorporate this instruction into your 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. The Battery instruction measures the voltage at the Power In terminals, not the voltage of the lead acid battery. External power sources must be disconnected from the CR3000 and charging circuit in order to measure the actual lead acid battery voltage.

TABLE 1.3-3. PS100, Battery, and AC Transformer Specifications	
Input Voltage (CHG terminals)	15 to 28 VDC or 18 VAC RMS
Battery Connections	
Charging Output Voltage:	Temperature compensated float charge for 12 V Battery
Temperature Compensation Range:	-40 to +60°C
Charging Current Limit:	1.2 Amps typical
Power Out (+12 terminals)	
Voltage:	Unregulated 12 V from Battery
Current Limited w / 3 A Thermal Fuse:	> 3 A @ < 20°C 3 A @ 20°C 2.1A @ 50°C 1.8 A @ 60°C
Batteries	
Operating Temperature Range:	-40 to +60°C
Capacity:	
PS100	7 Amp hours
BP12	12 Amp hours
BP24	24 Amp hours
AC Transformer: CSI Model No. 9591	
Input Voltage:	120 VAC
Output Voltage:	18 VAC RMS
Output Current (max):	1.2 Amps RMS
Protection (automatic reset):	85°C thermal reset breaker
UL Approval:	UL-1950
AC Transformer: CSI Model No. 14014	
Input Voltage:	90 - 264 VAC; 47 - 63 Hz
Output Voltage:	18 VDC
Output Current (max):	1.3 Amps
UL Approved, File No.:	E137895

There are inherent hazards associated with the use of sealed lead acid batteries. See section 1.3.2 for details.

1.3.3.3 CH100

The CH100 contains the same circuitry as the PS100. It is used to float charge an external 12 VDC Yuasa battery using AC or solar power. No internal batteries are contained in the CH100. Other power supply options are connecting a 12-volt battery directly to the CR3000, Section 1.5, or supplying power from a vehicle, Section 1.6.

1.3.3.4 A100 Null Modem Adapter

The A100 Null Modem Adapter is used when 5 volts is needed to power external modems with the PS100 or CH100. The A100 supplies 5 volts to pin 1 of the 9 pin null modem ports. A common use for the A100 is in radiotelemetry networks.

The maximum current drain on the 5-volt supply of the A100 is 150 milliamps.

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/m²). 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.

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.

Connect solar panel leads to the green connector at the side of the CR3000 next to the power switch (rechargeable base only). Polarity does not matter.

1.5 Direct Battery Connection to the CR3000 Wiring Panel

Any clean, battery backed 11 to 16 VDC supply may be connected to the Power In terminals on the front panel. When connecting external power to the CR3000, first, remove the green power connector from the CR3000 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 CR3000 will draw current from the source with the largest voltage. When power is connected through the front panel, switch control on the standard CR3000 power supplies is by-passed.

1.6 Vehicle Power Supply Connections

If a CR3000 is 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 CR3000 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 CR3000. The diodes allow the vehicle to power the CR3000 without the second supply attempting to power the vehicle.

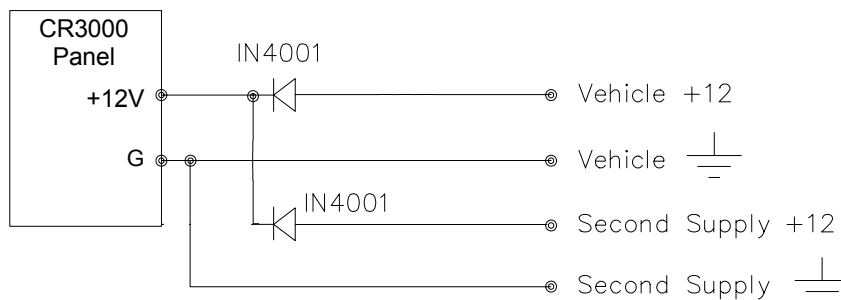


FIGURE 1.6-1. Connecting CR3000 to Vehicle Power Supply

1.7 CR3000 Grounding

Grounding the CR3000 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 CR3000 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 CR3000.

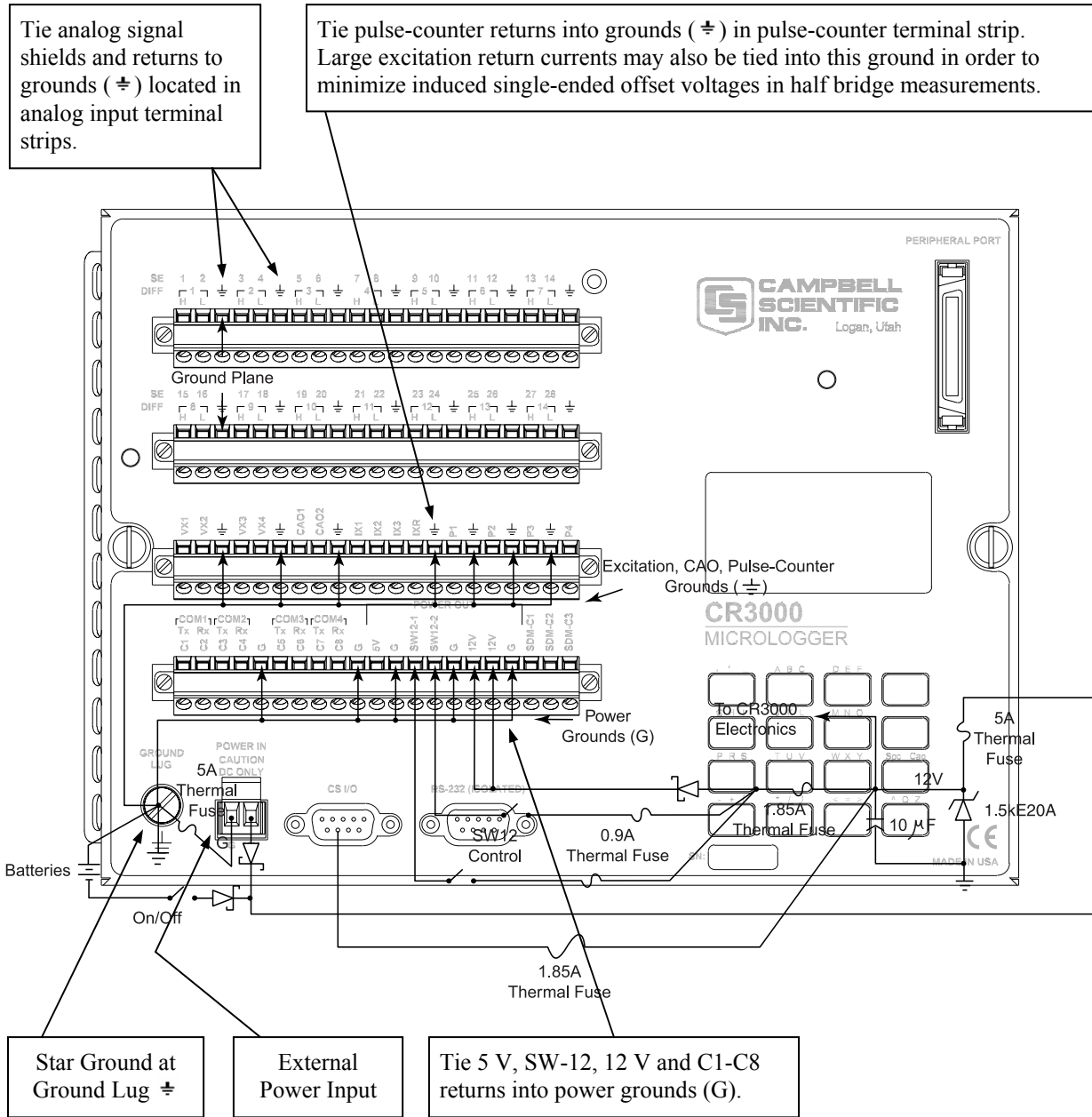


FIGURE 1.7-1. Schematic of CR3000 Grounds

The 9-pin serial I/O ports on the CR3000 are another path for transients to enter and damage the CR3000. Communications devices such as 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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 (\oplus) 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. Several ground wires can be connected to the same ground terminal.

Examples:

1. Connect 5 Volt, 12 Volt, and control grounds to G terminals.
2. Connect excitation grounds (VX and CAO) to the closest \oplus terminal on the excitation terminal block.
3. Connect the low side of single-ended sensors to the nearest \oplus terminal on the analog input terminal blocks.
4. Connect shield wires to the nearest \oplus 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 \oplus terminals next to the excitation 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 CR3000 is a convenient source of power for sensors and peripherals requiring a continuous or semi-continuous 5 VDC or 12 VDC source. The CR3000 has two continuous 12-Volt (12V) supply terminals, two switched 12 Volt (SW-12) supply terminals, and one continuous 5 Volt (5V) supply terminals. Voltage on the 12V and SW-12 terminals will change with the CR3000 supply voltage. The 5V terminal is regulated and will always remain near 5 Volts ($\pm 4\%$) so long as the CR3000 supply voltage remains above 10 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 CR3000.

TABLE 1.8-1. Current Sourcing Limits

<u>Terminals</u>	<u>Current Source Limit</u>
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 CR3000 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 CR3000 Peripherals

Peripheral	Typical Current Drain (mA)	
	Quiescent	Active
AM25T	0.0005	1
COM210 Phone Modem	0.12	140
SDM-INT8	0.4	13-20

1.9 Controlling Power to Sensors and Peripherals

Controlling power to an external device is a common function of the CR3000.

Many devices can conveniently be controlled with the SW-12 (Switched 12 Volt) terminals on the CR3000. Table 1.8-1 shows the current available from SW-12 port.

Campbell Scientific's SW12V Power Switch Assembly can also be used control power to one or two external devices simultaneously. It is comprised of an 18 inch (45 cm) length of cable that contains an internal switch. A control port controls the status of the switch, thereby controlling whether 12 Vdc power is passed to an external peripheral.

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 (C1 - C8) 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.

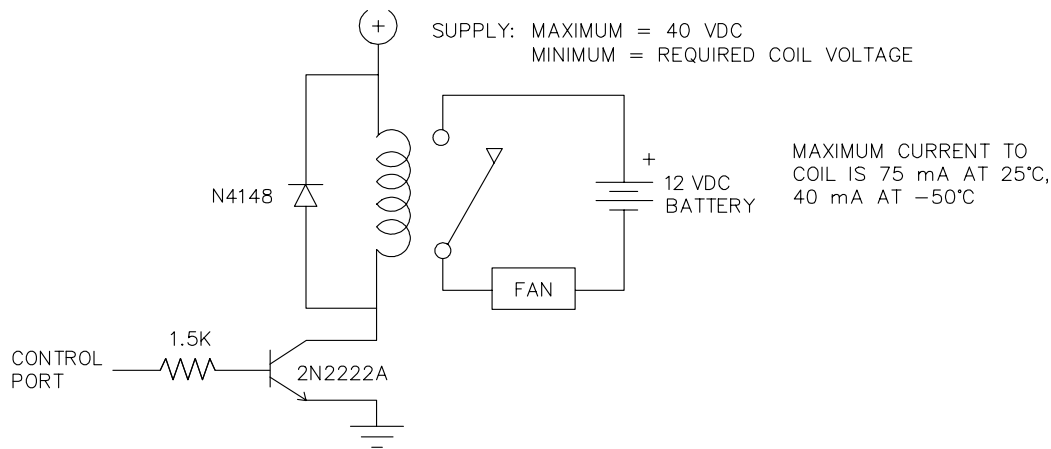


FIGURE 1.9-1. Relay Driver Circuit with Relay

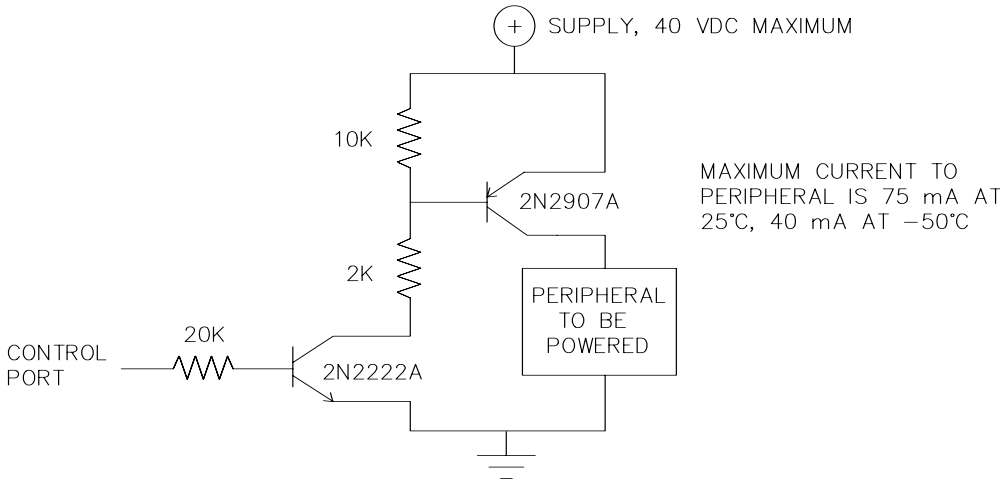


FIGURE 1.9-2. Power Switching without Relay

1.10 Maintenance

The CR3000 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.

The alkaline supply should not drop below 11.0 V before replacement. When not in use, remove the ten cells to eliminate potential corrosion of contact points and store in a cool dry place.

1.10.1 Desiccant

The CR3000 is shipped with desiccant to reduce humidity. Desiccant should be changed periodically. To prevent corrosion in uncontrolled or condensing atmospheres, the CR3000 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 CR3000 contains a lithium battery that operates the clock and SRAM when the CR3000 is not powered. The CR3000 does not draw any power from the lithium battery while it is powered by a 12 VDC supply. In a CR3000 stored at room temperature, the lithium battery should last approximately 3

years (less at temperature extremes). Where the CR3000 is powered most or all of the time the lithium cell should last much longer.

While powered from an external source, the CR3000 measures the voltage of the lithium battery daily. This voltage is displayed in the status table (Appendix A) A new battery will have approximately 3.6 volts. The CR3000 Status Table has a “Lithium Battery” field indicating the current value of the lithium battery. This value is updated in the background calibration. If the lithium cell is removed or allowed to discharge below the safe level, the CR3000 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 13519). Table 1.10-1 lists the specifications of the battery.

TABLE 1.10-1. CR3000 Lithium Battery Specifications		
Model	Tadiran	TL-59025 (3.6 V)
Capacity		1.2 Ah
self discharge rate		1%/year @ 20°C
Operating temperature range		-55°C to 85°C

The CR3000 must be partially disassembled to replace the lithium cell.

Figures 1.10-1 to 1.10-5 illustrate how to disassemble the CR3000. Reverse these steps to reassemble the CR3000. Disconnect any external power source wired into the Power In plug and/or set the power switch to the off position before proceeding with the lithium battery replacement.

The battery is held in place by a plastic cover. It can be removed by gently prying the cover from the battery holder.



FIGURE 1.10-1. CR3000 with wiring panel.



FIGURE 1.10-2. Loosen thumbscrews to remove CR3000 wiring panel from base.



FIGURE 1.10-3. Disconnect internal battery from wiring panel (BPALK and Sealed Rechargeable base only).

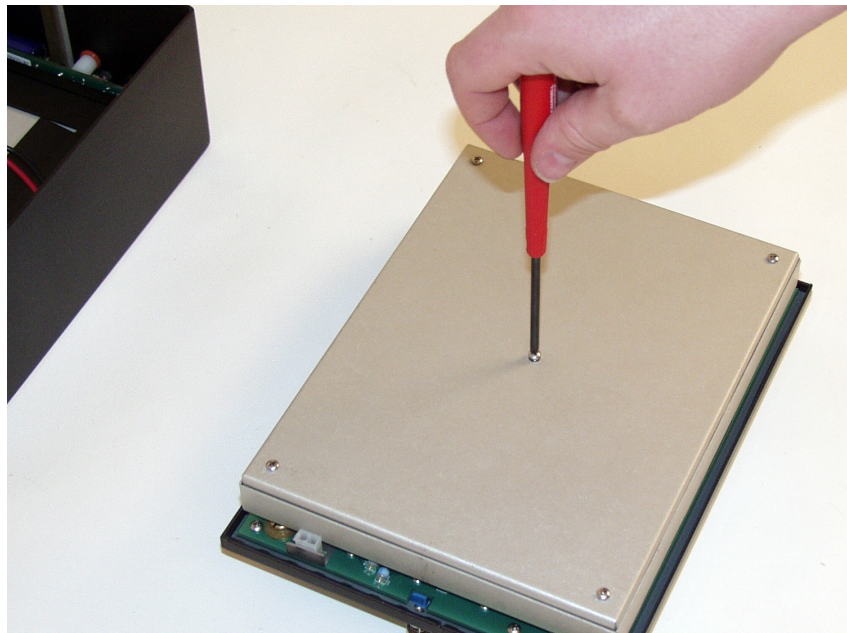


FIGURE 1.10-4. Remove plate to expose lithium battery.

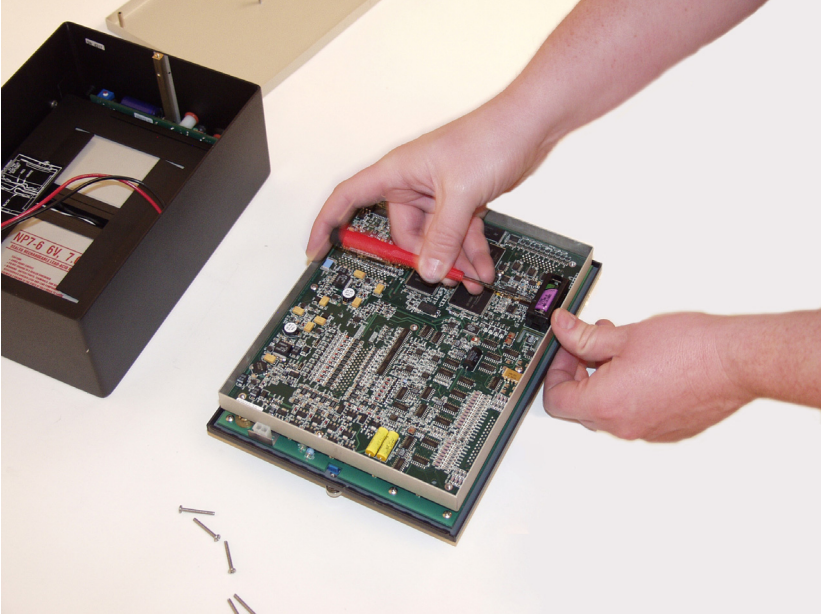


FIGURE 1.10-5. Remove battery cover and replace battery.

Section 2. Data Storage and Retrieval

The CR3000 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. The number of tables and the values to output in each table are selected when running Short Cut (Overview) or when writing a datalogger program directly (Sections 4 – 9).

2.1 Data Storage in CR3000

There are two possible areas for data storage on the CR3000:

- 1) Internal SRAM
- 2) The optional CFM100 Compact Flash Module and NL115 Ethernet Interface and Compact Flash Module allow storing data in a compact flash card.

Internal SRAM is used as either the sole storage area for a data table or as a buffer area when data are sent to compact flash (CF) card.

When the CR3000 gets a request for data that is stored on a CF card, the CR3000 only looks for the data in the CF 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 CF card by including the **CardOut** instruction within the data table declaration.

2.1.1 Internal SRAM

Internal SRAM is used as either the sole storage area for a data table or as a buffer area when data are sent to a CF card. The maximum number of data tables that can be created is 30. Internal SRAM is battery backed. Data remain in memory when the CR3000 is powered down. Data in SRAM are erased when a different program is loaded and run.

There are 4 Mbyte of SRAM. Some of this is used by the operating system and for CPU usage and program storage. The rest is available for data storage. When a new program is compiled, the CR3000 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 CFM100 or NL115

The CFM100 and NL115 are optional modules that plug onto the CR3000 peripheral port and have a slot for a Type I or Type II CF card. CF cards expand the CR3000's storage capacity. A program can send a maximum of 30 data tables to a CF card.

Campbell Scientific offers and recommends CF cards manufactured by Silicon Systems. Silicon Systems' CF cards are industrial-grade and have passed our ESD testing. Only industrial-grade CF cards with a storage capacity of 2 Gbytes or less should be used with our products. Refer to the PC/CF Card Information Application Note for more information.

NOTE

To install or remove the CFM100 or NL115 module, turn the CR3000 power off.

When a data table is sent to a CF card, the data table in SRAM is used as a buffer for transferring data to the card. When the card is present, the status table will show the size of the table on the card for the table size. If the card is removed, the size of the table in SRAM will be shown.

When a new program is compiled that sends data to the CF card, the CR3000 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 CR3000 will start storing data to them. If there is no card or if there is not enough space, the CR3000 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 CR3000 will create the data tables on the card and store the data that is accumulated in SRAM.

Data stored on cards can be retrieved through a communication link to the CR3000 or by removing the card and carrying it to a computer. Because of the popularity of compact flash cards for digital photography, there are many varieties of compact flash adapters for computers. There are also passive adapters that allow a compact flash card to be plugged into the PCMCIA card slot available on many laptops. The PCMCIA interface is much faster than the communication link. With large files transferring the CF card to the computer may be faster than collecting the data over the link.

The CR3000 uses either the FAT or the FAT 32 format for the CF cards. Cards can be formatted in a PC or in the CR3000. See the CFM100 or NL115 manual for additional information.

CAUTION

To prevent losing data, collect data from the CF card before sending the datalogger a new or modified program. When a program is sent to the datalogger using the Send button in the Connect screen of LoggerNet or PC400, an attribute is sent along with the program that commands the datalogger to erase all data on the CF card from the currently running program.

2.2 Internal Data Format

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)
Boolean	4 bytes	True/False	—
String	1 byte per character + 1 byte, 16 byte minimum	—	—

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 six data types used to store data. The data format is selected in the instruction that outputs the data. Within the CR3000, 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 (FP2) 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).

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).

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 via a communications link using one of Campbell Scientific's datalogger support software packages (e.g., PC200, PC400, LoggerNet) or by transferring a CF card from the CR3000 to the computer.

When the CR3000 is used without a computer in the field, or large data files are collected on a CF card, the CF card can be transported to the computer with the data on it.

The format of the data files on the CF card is different than the data file formats created when data are collected via a communications link. Data files read directly from the CF card generally need to be converted into another format to be used.

2.3.1 Via a Communications Link

See the manual and help for the software package you are using.

2.3.2 Via CF Card

When the CR3000 is used without a communication link in the field, or large data files are collected on a CF card, the card can be transported to the computer with the data on it. See the CFM100 or NL115 manual for additional information.

2.3.2.1 Inserting a CF Card

A card inserted in the CF slot of the CFM100 or NL115 when no program is running or when a program is running that does not use the CF card does not cause a response from the CR3000. When a new program is compiled that sends data to the CF card, the CR3000 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 CR3000 will start storing data to them.

When a card is inserted into a CR3000 that is programmed to send data to a card, the CR3000 will detect the card.

If the card has no data tables files with the same names as those created by the program, the CR3000 checks if the card has adequate space for the data tables. If the card has adequate space, the tables will be allocated and the CR3000 will start storing data to them.

If the card has existing data table files whose names match those created by the program, and the program has already run with a card, the CR3000 checks if the data table headers match the program exactly and if the time that the CR3000 last stored data to a card matches a time stored on the card, then the CR3000 starts appending to the data table on the card. This option allows removing the card to read it and then returning it to the CR3000 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 CR3000 overwrites its buffer while the card is out, there will be a gap in the data on the card.

If there is not enough space on the card or the data tables do not match, the CR3000 will turn the LED yellow indicating that the card is not being used.

2.3.2.2 Removing Card from CR3000

To remove a card, press the button on the CFM100 or NL115 . The CR3000 will store any buffered data to the card and then power if off. The Status LED will turn green when it is OK remove the card. Remove the card. The card will be reactivated after 20 seconds if it is not removed.

The CF card Status LED just above the CF card door is red when the card is being written to.

CAUTION

Removing a card or removing power from the card while it is active can cause garbled data and can actually damage the card. Before switching off the CR3000 power, press the CFM100 or NL115 button and wait for the green LED.

When the CF card is inserted in a computer, the data files can be copied to another drive or used directly from the CF 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.

2.3.2.3 Converting File Format

The CR3000 stores data on the CF card in TOB3 Format. TOB3 is a binary format that incorporates features to improve reliability of the CF cards. TOB3 allows the accurate determination of each record's time without the space required for individual time stamps.

When TOB3 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.

CardConvert software included in PC200, PC400, and LoggerNet will convert the data on the card to another format.

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 computer software used to collect the data (PC200, PC400, LoggerNet).

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","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

File Format

The format of the file on disk. TOA5 is an ASCII format. TOB1 is a Binary format.

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 CR3000 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 program that was running when the data were created.

DLD Signature

The signature of the program 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 CR3000 by appending underscore (_) and a three character mnemonic for the output processing (e.g., _AVG, _TOT, etc.).

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 TOA5.

```
"TOA5","Fritz","CR3000","1079","CR3000.Std.1.0","CPU:TCTemp.CR1","51399","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"
"2004-10-27 16:20:00",0,24.1,24.03,24.04,24.05,24.04
"2004-10-27 16:30:00",1,24.01,24.01,24.01,24,23.99
```

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

TOA5	Fritz	CR3000	1079	CR3000.Std.1.0	CPU:TCTemp.CR1	51399	Temp
TIMESTAMP	RECORD	RefT_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	
10/27/2004 16:20	0	24.1	24.03	24.04	24.05	24.04	
10/27/2004 16:30	1	24.01	24.01	24.01	24	23.99	

2.4.3 TOB1 Binary File Format

This is a sample of the TOB1 Binary file header

```
"TOB1","Fritz","CR3000","1079","CR3000.Std.1.0","CPU:TCTemp.CR1","51399","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"
"ULONG","ULONG","ULONG","FP2","FP2","FP2","FP2","FP2"
(Data are binary and not directly readable)
```

2.4.4 TOB3 Binary File Format

The TOB3 binary format has a header similar to the other formats. TOB3 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 TOB3 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. CR3000 Measurement Details

3.1 Analog Voltage Measurement Sequence

The CR3000 measures analog voltages with either an integrate and hold or 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. Using the fastest possible measurements at the minimum scan rate of 10 ms (100 Hz) the CR3000 can make and store measurements from all 14 differential channels. The maximum conversion rate is 1500 per second for measurements made on a single channel.

The timing of CR3000 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**), to 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**), to set the length of time to integrate a measurement (**Integ**), and to reverse the polarity of excitation voltage (**RevEx**).

3.1.1 Voltage Range

Fixed Voltage Ranges

The CR3000 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 ± 5000 mV range the full scale range is 10000 mV [5000 - (-5000)] and the resolution is one sixth of a millivolt; $10000/0.167 = 60000$. 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 CR3000 indicates the overrange by returning Not-A-Number (NAN) for the measurement.

AutoRange

For signals that do not fluctuate too rapidly, AutoRange allows the CR3000 to automatically choose the voltage range to use. The CR3000 AutoRange makes two measurements. The first measurement determines the range to use. It is

made with the 250 μ s 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 476 $^{\circ}$ C (± 25 mV range) but will occasionally see temperatures as high as 500 $^{\circ}$ C (± 250 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.

Open Circuit Detect / Pull into Common Mode

Another option selected with the voltage range code is to check for an open circuit and at the same time pull the signal into common mode range. The range codes for this option end in C. For example, the range code for the ± 20 mV range is “mV20”, the code for this range with open circuit detect is “mV20C”

The open circuit detect works by briefly (50 microseconds) connecting the voltage input to 300 mV within the CR3000. A differential voltage input has the high side connected to 300 mV and the low side connected to ground. After disconnecting, the input is allowed to settle, and the voltage measurement is made. If the sensor is open (inputs not connected and “floating”) the inputs will remain floating near the voltage they were connected to; a measurement on the ± 20 mV, ± 50 mV, or the ± 200 mV voltage range will overrange and return Not-A-Number (NAN). If the sensor is good, the signal from the sensor will drive the inputs to the correct value.

The autorange open circuit detect range (AutorangeC) will only autorange up to the ± 200 mV range and cannot be used for higher voltages. If AutorangeC was allowed to switch to the ± 1000 mV or ± 5000 mV range it would not be able to detect open circuits.

Briefly connecting the inputs to the internal CR3000 voltages also serves to pull a floating differential voltage into the CR3000 common mode (Section 3.2). The open circuit detect voltage range option should be used for making differential voltage measurements of thermocouples (TCDiff) and for other sensors with floating differential outputs (e.g., Solar radiation sensors).

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 CR3000 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 CR3000 is undergoing rapid temperature swings.

3.1.4 SettlingTime

When the CR3000 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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 CR3000 may use a combination of analog and digital integration.

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 μ s. 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 is specified as part of the measurement instruction. An integration time of 250 selects the 250 μ s integration, “_60 Hz” selects the 60 Hz rejection (16.667 ms), and “_50 Hz” selects 50 Hz rejection (20 ms).

The ± 5000 mV range does not use a full integration when the “_50 Hz” integration is selected. Instead, 50 Hz noise is rejected by averaging two of the 250 μ s integration measurements that are spaced exactly one half of a 50 or 60 Hz cycle apart. The average of these measurements is stored as the result of the measurement.

The integration 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 (1-14) 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 1-28.

Because a single ended measurement is referenced to CR3000 ground, any difference in ground potential between the sensor and the CR3000 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 CR3000 is grounded, the measured voltage would be 1 mV greater than the thermocouple output, or approximately 25 °C 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 CR3000. 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 CR3000 common mode range of ± 5 V. The common mode range is the voltage range, relative to CR3000 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 CR3000 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 CR3000 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 use the voltage range option to pull into common mode range (Section 3.1.1) or 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 CR3000 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 CR3000. 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 CR3000 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 CR3000 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 CR3000 circuitry.

3.3 Signal Settling Time

Whenever an analog input is switched into the CR3000 measurement circuitry prior to making a measurement, a finite amount of time is required for the signal to stabilize at its 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 CR3000 delays after switching to a channel to allow the input to settle before initiating the measurement. The default delays used by the CR3000 depend on the integration used and the voltage range. The default delay is 450 μ s for 250 μ s 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 CR3000.

3.3.2 Measuring the Necessary Settling Time

The CR3000 can measure the time required for a particular sensor/cable configuration to settle. This is done by making a number of measurements with different settling times. Looking at the series of measurements it is possible to see the settling of the sensor signal.

The following example demonstrates measuring the settling time for a differential voltage measurement. If you are not yet familiar with CR3000 programming, you may want to read Section 4 before trying to follow the example.

The series of measurements on the sensor is made with separate measurements for each settling time.

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.

```
'CR3000 Series Datalogger
'Program to measure the settling time of a sensor
'measured with a differential voltage measurement

Public PT(20)                'Variable to hold the measurements

DataTable (Settle,True,100)
  Sample (20,PT(),IEEE4)
EndTable

BeginProg
  Scan (1,Sec,3,0)
    BrFull (PT(1),1,mV20,1,Vx1,1,2500,True ,True ,100,250,1.0,0)
    BrFull (PT(2),1,mV20,1,Vx1,1,2500,True ,True ,200,250,1.0,0)
    BrFull (PT(3),1,mV20,1,Vx1,1,2500,True ,True ,300,250,1.0,0)
    BrFull (PT(4),1,mV20,1,Vx1,1,2500,True ,True ,400,250,1.0,0)
    BrFull (PT(5),1,mV20,1,Vx1,1,2500,True ,True ,500,250,1.0,0)
    BrFull (PT(6),1,mV20,1,Vx1,1,2500,True ,True ,600,250,1.0,0)
    BrFull (PT(7),1,mV20,1,Vx1,1,2500,True ,True ,700,250,1.0,0)
    BrFull (PT(8),1,mV20,1,Vx1,1,2500,True ,True ,800,250,1.0,0)
    BrFull (PT(9),1,mV20,1,Vx1,1,2500,True ,True ,900,250,1.0,0)
    BrFull (PT(10),1,mV20,1,Vx1,1,2500,True ,True ,1000,250,1.0,0)
    BrFull (PT(11),1,mV20,1,Vx1,1,2500,True ,True ,1100,250,1.0,0)
    BrFull (PT(12),1,mV20,1,Vx1,1,2500,True ,True ,1200,250,1.0,0)
    BrFull (PT(13),1,mV20,1,Vx1,1,2500,True ,True ,1300,250,1.0,0)
    BrFull (PT(14),1,mV20,1,Vx1,1,2500,True ,True ,1400,250,1.0,0)
    BrFull (PT(15),1,mV20,1,Vx1,1,2500,True ,True ,1500,250,1.0,0)
    BrFull (PT(16),1,mV20,1,Vx1,1,2500,True ,True ,1600,250,1.0,0)
    BrFull (PT(17),1,mV20,1,Vx1,1,2500,True ,True ,1700,250,1.0,0)
    BrFull (PT(18),1,mV20,1,Vx1,1,2500,True ,True ,1800,250,1.0,0)
    BrFull (PT(19),1,mV20,1,Vx1,1,2500,True ,True ,1900,250,1.0,0)
    BrFull (PT(20),1,mV20,1,Vx1,1,2500,True ,True ,2000,250,1.0,0)
  CallTable Settle
  NextScan
EndProg
```

The program was run on a Druck water level pressure transducer with 200 feet of cable. The first six measurements are shown in Table 3.3-1. All 20 values are plotted in Figure 3.3-1. The reading has settled by the fourteenth measurement, PT(14), thus a settling time of 1400 μ s is adequate.

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

TOA5 TIMESTAMP TS	Pepe' RECORD RN	CR3000 PT(1)	1079 PT(2)	CR3000.Std.01.00 PT(3)	CPU:Settlebridge.CR3 PT(4)	1455 PT(5)	Settle PT(6)
		Smp	Smp	Smp	Smp	Smp	Smp
1/3/2000 23:34	0	0.03638599	0.03901386	0.04022673	0.04042887	0.04103531	0.04123745
1/3/2000 23:34	1	0.03658813	0.03921601	0.04002459	0.04042887	0.04103531	0.0414396
1/3/2000 23:34	2	0.03638599	0.03941815	0.04002459	0.04063102	0.04042887	0.04123745
1/3/2000 23:34	3	0.03658813	0.03941815	0.03982244	0.04042887	0.04103531	0.04103531
1/3/2000 23:34	4	0.03679027	0.03921601	0.04022673	0.04063102	0.04063102	0.04083316

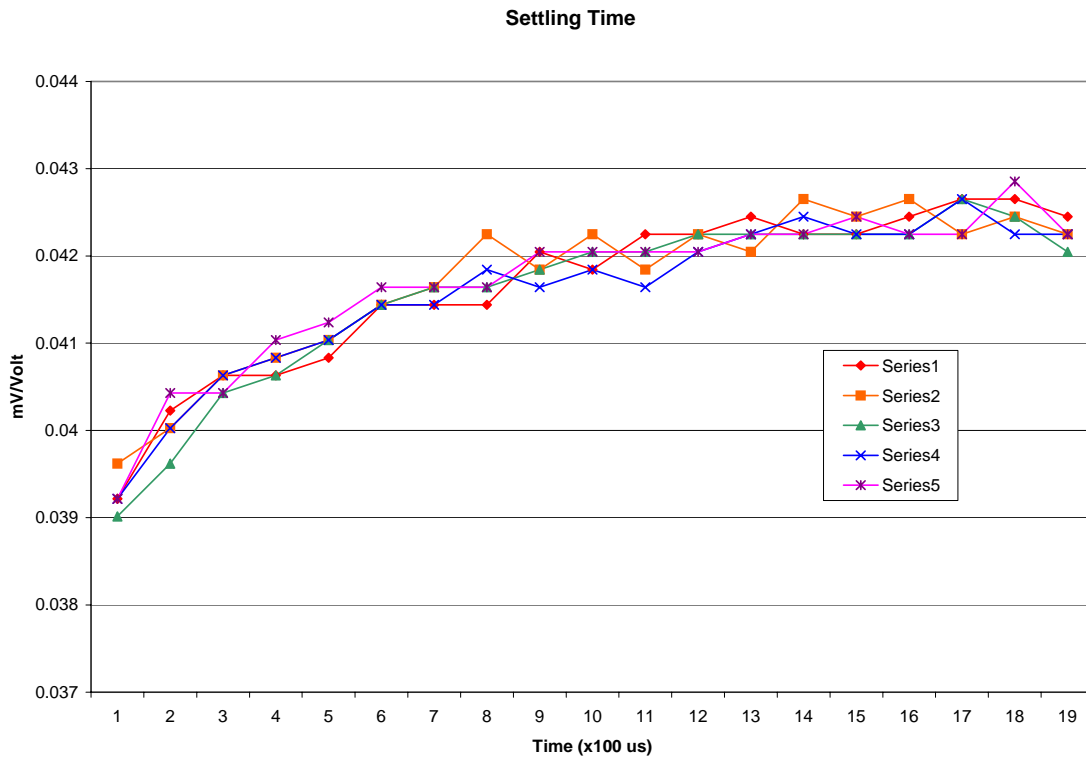


FIGURE 3.3-1. Settling Time for Pressure Transducer

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 CR3000 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 CR3000 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 °C, 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 CR3000 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

The panel temperature thermistor is just under the panel in the center of the two rows of analog input terminals.

The thermistor (Betatherm 10K3A1A) has an interchangeability specification of 0.1 °C for temperatures between 0 and 70 °C. Below freezing and at higher temperatures this specification is degraded. Combined with possible errors in the completion resistor measurement, and the Steinhart and Hart equation used to calculate the temperature from resistance, the accuracy of panel temperature is $\pm 0.3^{\circ}\text{C}$ -25 to 50°C and $\pm 0.8^{\circ}\text{C}$ -55 to 85°C.

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 CR3000 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.

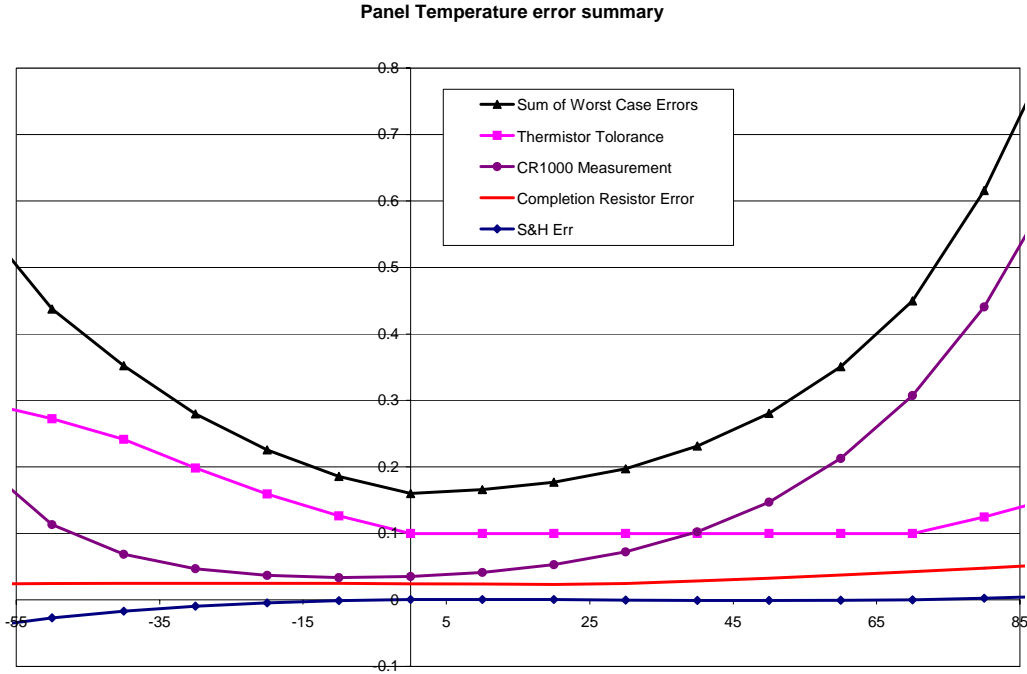


FIGURE 3.4-1. Panel Temperature Errors

With an external driving gradient, the temperature gradients on the input panel can be much worse. For example, the CR3000 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 -65 to 70°C in approximately 90 minutes. Figure 3.4-3 shows the results when going from 75 to 25°C. 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 CR3000. Under these conditions, terminals closest to the panel thermistor had the least error while terminals farthest from the thermistor showed the most error in temperature measurements. A second set of tests run with thermocouples connected to channels 8 through 14 showed a similar pattern of temperature measurement errors.

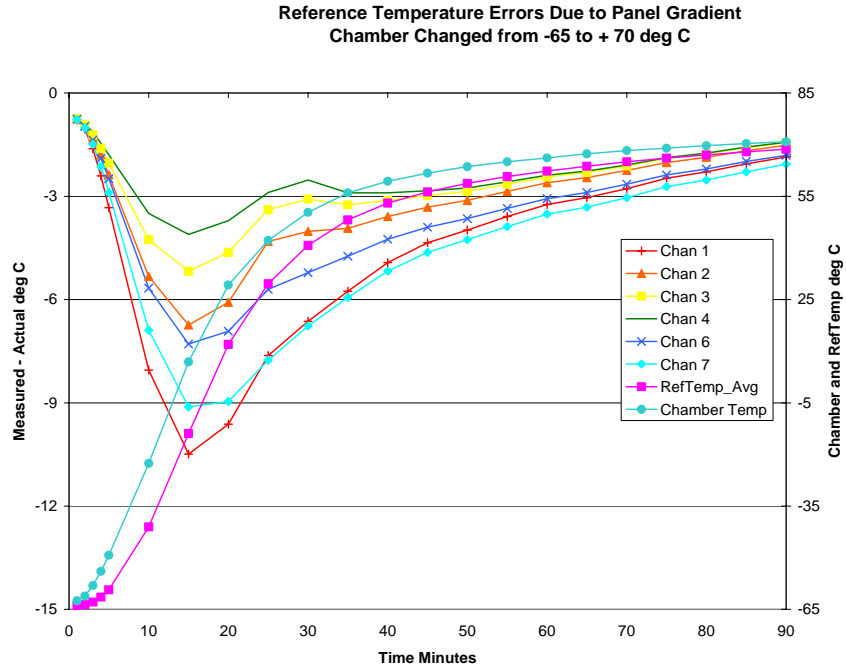


FIGURE 3.4-2. Panel Temperature Gradients during -65 to 70 °C Change

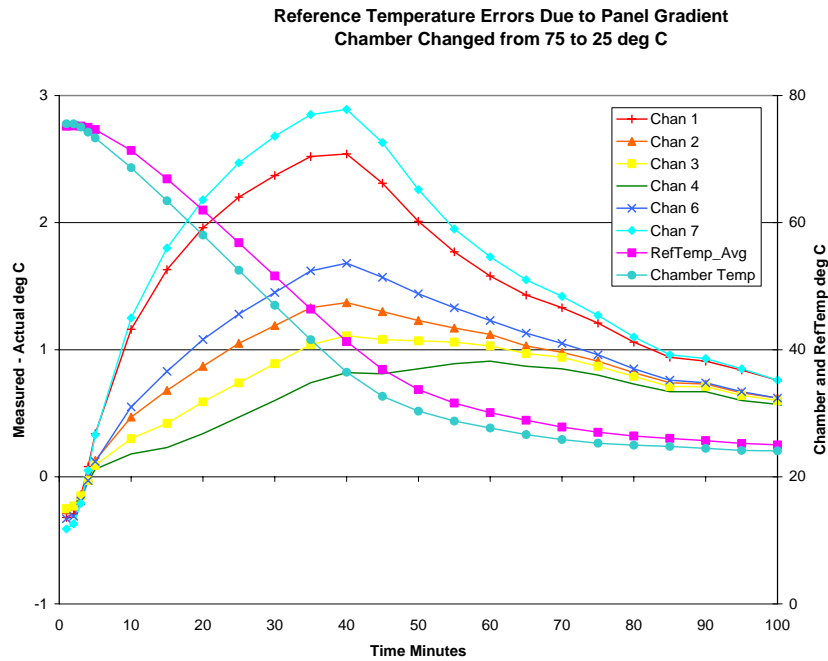


FIGURE 3.4-3. Panel Temperature Gradients during 75 to 25 °C Change

Thermocouple Limits of Error

The standard reference which lists thermocouple output voltage as a function of temperature (reference junction at 0°C) 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 CR3000.

Thermocouple Type	Temperature Range °C	Limits of Error (Whichever is greater)	
		Standard	Special
T	-200 to 0	± 1.0°C or 1.5%	
	0 to 350	± 1.0°C or 0.75%	± 0.5°C or 0.4%
J	0 to 750	± 2.2°C or 0.75%	± 1.1°C or 0.4%
E	-200 to 0	± 1.7°C or 1.0%	
	0 to 900	± 1.7°C or 0.5%	± 1.0°C or 0.4%
K	-200 to 0	± 2.2°C or 2.0%	
	0 to 1250	± 2.2°C or 0.75%	± 1.1°C or 0.4%
R or S	0 to 1450	± 1.5°C or 0.25%	± 0.6°C 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 °C, 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 CR3000 differential voltage measurement is specified as $\pm (0.07\%$ of the measured voltage plus the input offset error of 3 times the basic resolution of the range being used to make the measurement plus 2 μV). The input offset error reduces to 1.5* the basic resolution +1 μV 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 0.07% 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 °C. The possible slope error in the voltage measurement is $0.0007 \times 830.7 \mu\text{V} = 0.58 \mu\text{V}$ or 0.014 °C (0.58/42.4). The basic resolution on the ± 20 mV range is 0.67 μV or 0.016 °C. The 2 μV offset is an error of 0.047 °C. Thus, the possible error due to the voltage measurement is 0.10 °C on a non-reversing differential, or 0.06 °C 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 °C. The TC output is on the order of 52 mV, requiring the ± 200 mV input range. At 1300 °C, a K thermocouple outputs 34.9 μV per °C. The possible slope error in the voltage measurement is $0.0007 \times 52 \text{ mV} = 36 \mu\text{V}$ or 1.03 °C (36/34.9). The basic resolution on the 200 mV range is 6.67 μV or 0.19 °C. Thus, the possible error due to the voltage measurement is 1.65 °C on a non-reversing differential, or 1.34 °C with a reversing differential measurement.

TABLE 3.4-2. Voltage Range for Maximum Thermocouple Resolution

TC Type and temp. range °C	Temp. range for ± 20 mV range	Temp. range for ± 50 mV range	Temp. range for ± 200 mV range
T -270 to 400	-270 to 395	not used	not used
E -270 to 1000	-270 to 300	-270 to 675	>675
K --270 to 1372	-270 to 500	-270 to 1250	>1250
J -210 to 1200	-210 to 385	-210 to 885	>885
B 0 to 1820	0 to 1820	not used	not used
R -50 to 1768	-50 to 1690	-50 to 1768	not used
S -50 to 1768	-50 to 1768	not used	not used
N -270 to 1300	-270 to 595	-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 250 μ s integration and input reversal is 0.4 μ V RMS. On a type T thermocouple (approximately 40 μ V/ $^{\circ}$ C) this is 0.01 $^{\circ}$ C. Note that this is an RMS value, some individual readings will vary by greater than this. By using the “60 Hz” integration for 16.67 μ s the noise level is reduced to .19 μ V 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 CR3000'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.

TC Type	Range $^{\circ}$C		Limits of Error $^{\circ}$C
T	-270	to 400	
	-270	to -200	+ 18 @ -270
	-200	to -100	\pm 0.08
	-100	to 100	\pm 0.001
	100	to 400	\pm 0.015
J	-150	to 760	\pm 0.008
	-100	to 300	\pm 0.002
E	-240	to 1000	
	-240	to -130	\pm 0.4
	-130	to 200	\pm 0.005
	200	to 1000	\pm 0.02
K	-50	to 1372	
	-50	to 950	\pm 0.01
	950	to 1372	\pm 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 CR3000 environmental operating range, so there is no problem when the CR3000 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 CR3000 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 μ V; using the reference temperature of 272.6 °C, the CR3000 calculates a temperature difference of -10.2 °C, a -0.2 °C error. The temperature calculated by the CR3000 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 CR3000 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 μ V (0.01% of 10 mV) which at 45 °C changes the temperature by 0.012 °C. The reference thermistor is 20 °C but is indicating 20.15 °C, and the terminal that the thermocouple is connected to is 0.05 °C cooler than the reference thermistor.

TABLE 3.4-5. Example of Errors in Thermocouple Temperature				
Source	Error: °C : % of Total Error			
	Single Differential 250 µs Integration		Reversing Differential 50/60 Hz Rejection Integration	
	ANSI TC Error (1°C)	TC Error 1% Slope	ANSI TC Error (1°C)	TC Error 1% Slope
Reference Temp.	0.15°:11.7%	0.15°:31.1%	0.15°:12.4%	0.15°:36.4%
TC Output	1.0°:78%	0.2°:41.5%	1.0°:82.5%	0.2°:48.6%
Voltage Measurement	0.10°:7.8%	0.10°:20.8%	0.05°:4.1%	0.05°:12.1%
Noise	0.03°:2.3%	0.03°:6.2%	0.01°:0.8%	0.01°:2.4%
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	1.282°:100%	0.482°:100%	1.212°:100%	0.412°: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 CR3000. 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 CR3000. Alternatively, the junction box can be used to couple extension grade thermocouple wire to the thermocouples, and the CR3000 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 CR3000. This is only a factor when using type K thermocouples, where the upper limit of the reference compensation linearization is 100 °C and the upper limit of the extension grade wire is 200 °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 3.4-4 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 CR3000. 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.

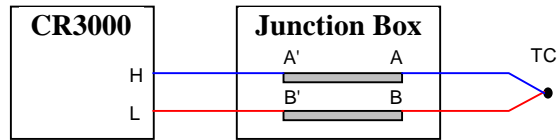


FIGURE 3.4-4. Diagram of Junction Box

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 3.4-3) 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 CR3000 can be used to measure the temperature gradients within the junction box.

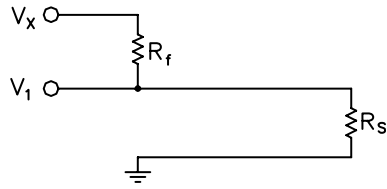
3.5 Bridge Resistance Measurements

There are six bridge measurement instructions included in the standard CR3000 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



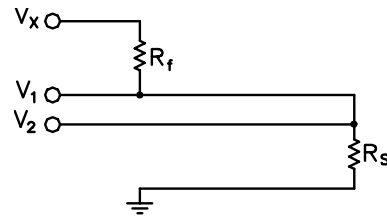
$X = \text{result w/mult} = 1, \text{offset} = 0$

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

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

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

BrHalf3W



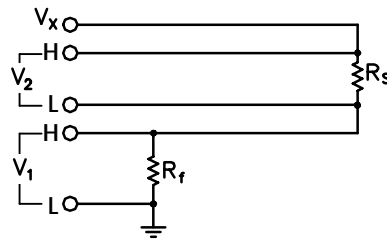
$X = \text{result w/mult} = 1, \text{offset} = 0$

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

$$R_s = R_f X$$

$$R_f = R_s / X$$

BrHalf4W



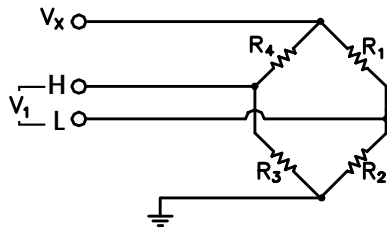
$X = \text{result w/mult} = 1, \text{offset} = 0$

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

$$R_s = R_f X$$

$$R_f = R_s / X$$

BrFull



$X = \text{result w/mult} = 1, \text{offset} = 0$

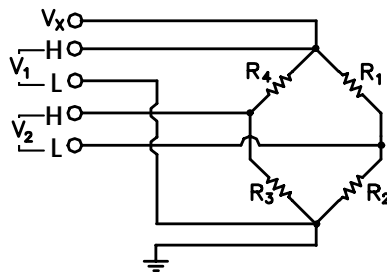
$$X = 1000 \frac{V_1}{V_x} = 1000 \left(\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2} \right)$$

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

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

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

BrFull6W



$X = \text{result w/mult} = 1, \text{offset} = 0$

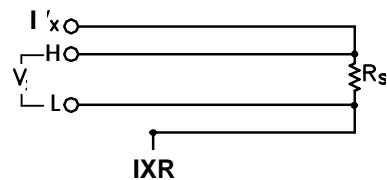
$$X = 1000 \frac{V_2}{V_1} = 1000 \left(\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2} \right)$$

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

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

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

Resistance

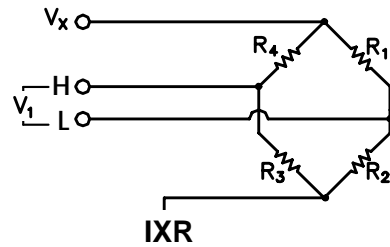


$X = \text{result w/mult} = 1, \text{offset} = 0$

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

Resistance

used to measure full bridge

 $X = \text{result w/mult} = 1, \text{ offset} = 0$

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

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

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

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

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

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

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 CR3000 ground, and can be represented by the model diagrammed in Figure 3.6-1.

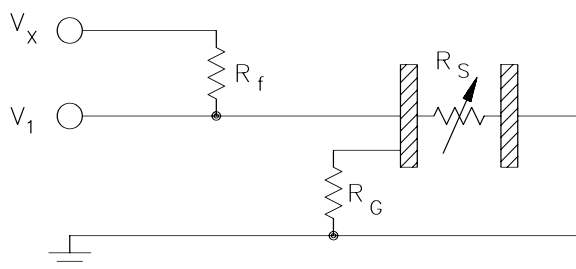


FIGURE 3.6-1. Model of Resistive Sensor with Ground Loop

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 CR3000 earth ground. With R_G in the network, the measured signal is:

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

$R_s R_f / R_G$ 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 CR3000 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 CR3000 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 = $1/\text{scan interval}$ (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 10 ms scan gives a resolution of 100 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.

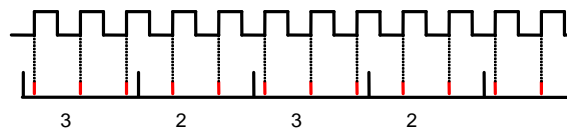


FIGURE 3.7-1. Varying counts within Pulse interval.

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 1000 Hz ($2000 \text{ RPM} \times (1 \text{ min}/60 \text{ s}) \times 30 = 1000$). The multiplier to convert from frequency to RPM is 2 RPM/Hz ($1 \text{ RPM}/(30 \text{ pulses}/60\text{s}) = 2$). At a 1 second scan interval, the resolution is 2 RPM. However, if the scan interval were 10 ms, the resolution would be 200 RPM. At the 10 ms scan, if every thing was perfect, each interval there would be 10 counts. However, a slight variation in the frequency might cause 9 counts within one interval and 11 in the next, causing the result to vary between 1800 and 2200 RPM!

3.8 Self Calibration

The CR3000 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 CR3000 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 CR3000, the calibration only measures the range/integration type combinations that occur in the running CR3000 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 usec integration multiple measurements are made and averaged to come up with gain values to use in the measurement. 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 CR3000 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 CR3000 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 CR3000 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 CR3000 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.

3.9 Measurement Accuracy

Accuracy, which really means inaccuracy, defines how far from the truth a measurement may be. The term “uncertainty” is preferred over accuracy in metrology (the science of measurements), although accuracy is more widely used in instrumentation. A typical datalogger accuracy specification includes only the dataloggers contribution to measurement error, and not the error of attached sensors. Since there is no universal method of specifying voltage measurement accuracy several different methods exist. In the past Campbell Scientific, Inc. has

used $\pm (0.X\% \text{ of Full-Scale Range})$ to specify measurement accuracy for a given range, which results in a fixed number for each input range.

While the $\pm (0.X\% \text{ of Full-Scale Range})$ method is simple to test and utilize in accuracy estimates, it overestimates errors for inputs much less than full-scale. This is because typical measurement instrumentation has measurement errors composed of two major components; a gain error and an offset error as illustrated in Figure 3.9-1. Gain error is minimized by factory calibration, but tends to increase with temperature deviations and/or component aging. An accuracy specification of $\pm (0.X\% \text{ of reading} + \text{Offset})$ better represents the instrument performance illustrated in Figure 3.9-1 than does $\pm (0.X\% \text{ of Full-Scale Range})$. Campbell Scientific, Inc. utilizes $\pm (0.X\% \text{ of reading} + \text{Offset})$ accuracy specifications for the CR9000, CR5000, CR3000 and CR1000 dataloggers.

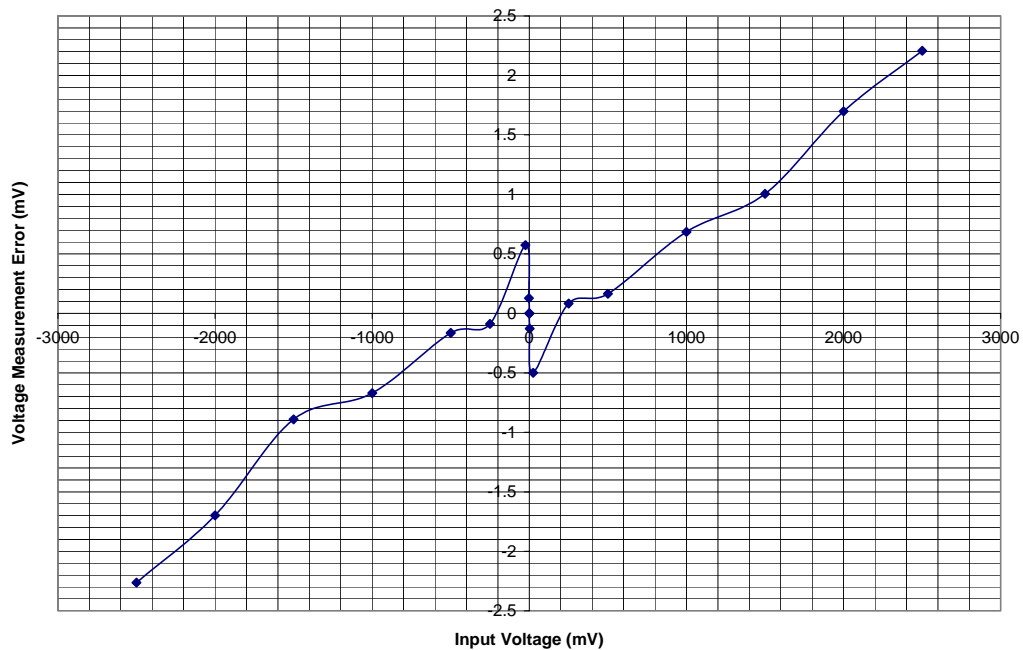


FIGURE 3.9-1. Instrument voltage measurement error performance illustrating dependence upon input voltage.

The following discussion relates to the CR10X and CR1000 dataloggers, but the principles are applicable to the CR3000 as well.

A graphical representation of the CR10X $\pm (0.X\% \text{ of Full-Scale Range})$ and CR1000 $\pm (0.X\% \text{ of reading} + \text{Offset})$ voltage measurement accuracy specifications is illustrated in Figure 3.9-2.

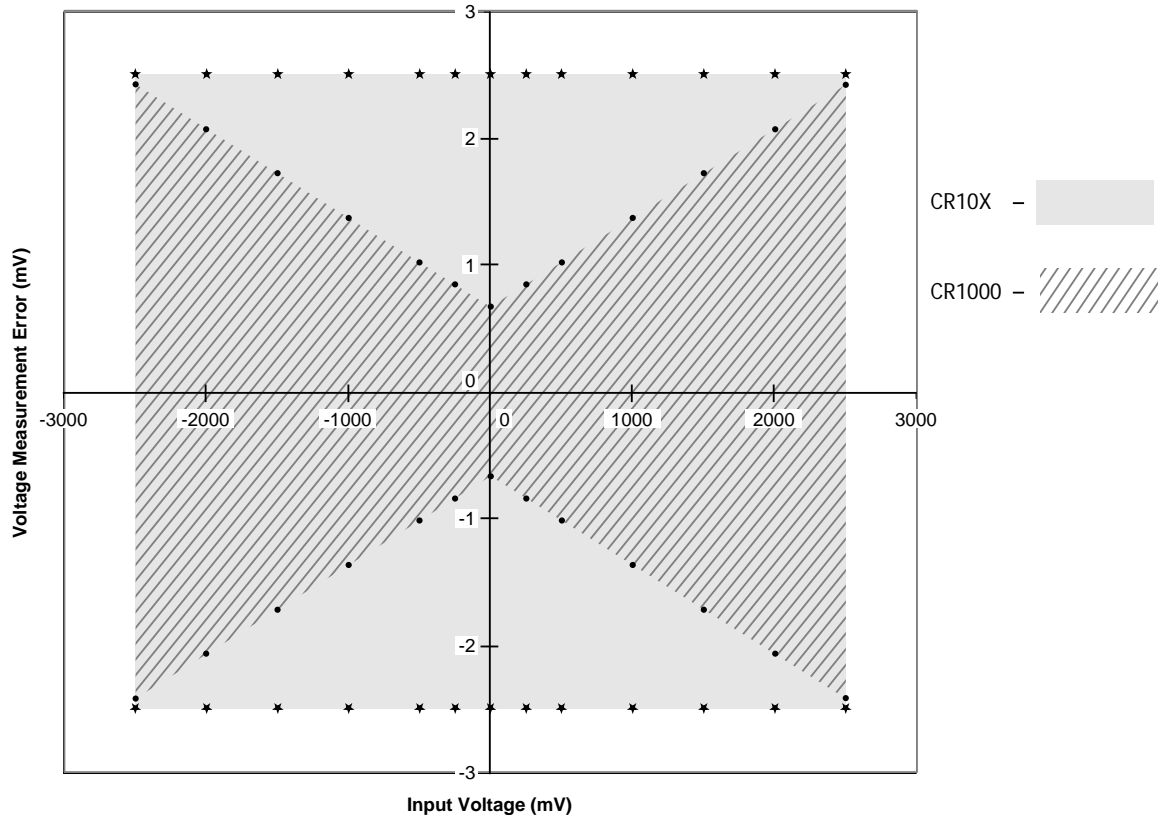


FIGURE 3.9-2. Comparison of the CR10X \pm (0.X% of Full-Scale Range) and CR1000 \pm (0.X% of reading + Offset) voltage measurement accuracy specifications for the (0 to 40)°C temperature range.

The CR10X accuracy specification for the ± 2500 mV input range over the (0 to 40)°C temperature range is \pm (0.05% of Full-Scale Range) = ± 2.5 mV, where Full-Scale Range has been defined as 5000 mV for the ± 2500 mV input range because that is the range the A/D conversion must span. The resulting CR10X accuracy specification equals ± 2.5 mV for the entire ± 2500 mV input range over the (0 to 40)°C temperature range, illustrated by the shaded rectangular area in Figure 3.9-2.

The CR1000 accuracy specification for the ± 2500 mV input range over the (0 to 40)°C temperature range is \pm (0.07% of reading + Offset) where Offset = Basic Resolution + 1 μ V for a differential measurement with input reversal. For a 2500 mV input signal over the (0 to 40)°C temperature range the Basic Resolution is 667 μ V, resulting in Offset = 668 μ V. The resulting voltage measurement accuracy over a ± 2500 mV input range over the (0 to 40)°C temperature range is a function of the input voltage as illustrated in Figure 3.9-2.

For a 2500 mV input signal over the (0 to 40)°C temperature range, the CR1000 accuracy specification results in ± 2.42 mV error which is comparable with the ± 2.5 mV for the CR10X. For a 500 mV input signal the specified CR10X accuracy is again ± 2.5 mV, whereas the CR1000 accuracy improves to ± 1.02 mV, which better represents actual instrument performance.

Section 4. CRBasic - Native Language Programming

The CR3000 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:

$$\text{TempF} = \text{TempC} * 1.8 + 32$$

With the CR3000 there may be many temperature (or other) measurements. For example, sixteen thermocouples could be attached to the CR3000 to measure temperature at fourteen locations. Rather than have 14 different names, a *variable array* with one name and 14 elements may be used. A thermocouple temperature might be called TCTemp. With an array of 14 elements the names of the individual temperatures are TCTemp(1), TCTemp(2), TCTemp(3), ... TCTemp(14). 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. Each procedure has a keyword name and a series of parameters that contain the information needed by the CR3000 to complete the procedure. For example, the instruction for measuring the temperature of the CR3000 input panel is:

```
PanelTemp (Dest, Integ)
```

PanelTemp is the keyword name of the instruction. The two parameters associated with PanelTemp are: *Destination*, the name of the variable in which to put the temperature; and *Integration*, 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 CR3000 code. When the CR3000 compiler sees the ' it ignores the rest of the line.

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

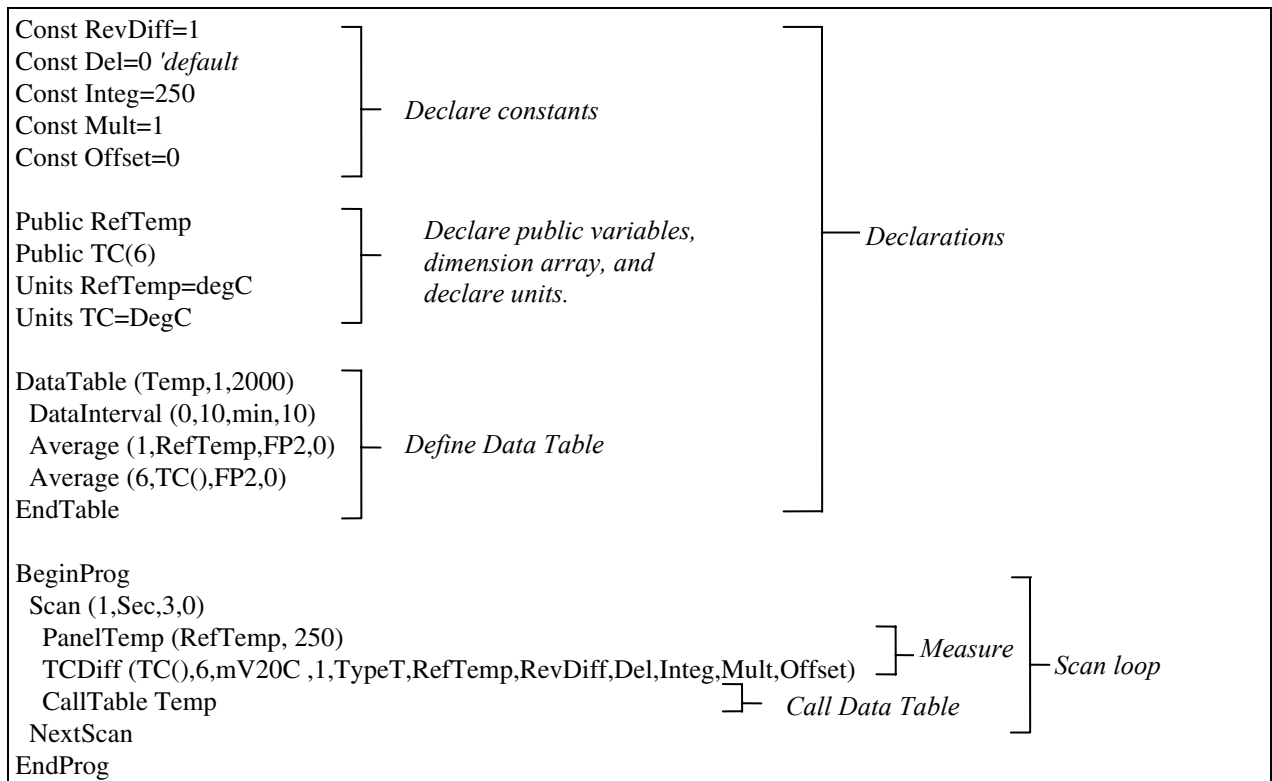
4.2 Programming Sequence

The following table describes the structure of a typical CR3000 program:

Declarations	<i>Make a list of what to measure and calculate.</i>
Declare constants	<i>Within this list, include the fixed constants used,</i>
Declare Public variables	<i>indicate the values that the user is able to view while the program is running,</i>
Dimension variables	<i>the number of each measurement that will be made,</i>
Define Aliases (optional)	<i>and specific names for any of the measurements.</i>
Declare Units (optional)	<i>Specify units for variables. This is for header information in the data tables and for the user's documentation. The CR3000 makes no checks on accuracy of units.</i>
Define data tables.	<i>Describe, in detail, tables of data that will be saved from the experiment.</i>
Process/store trigger	<i>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?</i>
Table size	<i>Set the size of the table in CR3000 RAM</i>
Other on-line storage devices	<i>Should the data also be sent to the PC card?</i>

Processing of Data	<i>What data are to be output (current value, average, maximum, minimum, etc.)</i>
Define Subroutines	<i>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.</i>
Program	<i>The program section defines the action of datalogging</i>
Set scan interval	<i>The scan sets the interval for a series of measurements</i>
Measurements	<i>Enter the measurements to make</i>
Processing	<i>Enter any additional processing with the measurements</i>
Call Data Table(s)	<i>The Data Table must be called to process output data</i>
Initiate controls	<i>Check measurements and Initiate controls if necessary</i>
NextScan	<i>Loop back (and wait if necessary) for the next scan</i>
End Program	

4.3 Example Program



4.3.1 Data Tables

Data storage follows a fixed structure in the CR3000 in order to optimize the time and space required. Data are stored in tables such as:

TOA5	Fritz	CR3000	1079	CR3000.Std.1.0	CPU:TCTemp.CR3	51399	Temp	
TIMESTAMP	RECORD	RefT_Avg	TC_Avg(1)	TC_Avg(2)	TC_Avg(3)	TC_Avg(4)	TC_Avg(5)	TC_Avg(6)
TS	RN	degC	DegC	DegC	DegC	DegC	DegC	DegC
		Avg	Avg	Avg	Avg	Avg	Avg	Avg
10/28/2004 12:10	119	23.52	23.49	23.49	23.5	23.49	23.5	23.5
10/28/2004 12:20	120	23.55	23.51	23.51	23.51	23.51	23.51	23.52
10/28/2004 12:30	121	23.58	23.52	23.53	23.53	23.53	23.53	23.53
10/28/2004 12:40	122	23.58	23.53	23.54	23.54	23.54	23.54	23.54

The user's program determines the values that are output and their sequence. The CR3000 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 CR3000 to fill this row out (e.g., Units RefTemp = degC). The units are strictly for the user's documentation; the CR3000 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,min,10)
    Average(1,RefTemp,fp2,0)
    Average(6,TC(),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 CR3000 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,min,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 minutes, 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: WorstCase 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 data table 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.

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)
BOOLEAN	4 byte Signed Integer	4 bytes	0, -1	True or False (-1 or 0)
STRING	ASCII String	Set by programmer		

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,Sec,3,0)
    PanelTemp(RefTemp, 250)
    TCDiff(TC),6,mV20C,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,Sec,3,0)
    
```

The Scan instruction has four parameters. The *Interval* is the interval between scans. *Units* are the time units for the interval. The minimum interval is 10 milliseconds. The maximum scan interval is 30 minutes. 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 second, three scans are buffered, and the measurements and output continue indefinitely.

4.4 Variable Data Types

The declaration of variables (via the DIM or the PUBLIC statement) allow an optional type descriptor AS that specifies the data type. The default data type, without a descriptor, is IEEE4 floating point (FLOAT). The data types are FLOAT, LONG, BOOLEAN, and STRING.

4.4.1 FLOAT

“AS FLOAT” specifies the default IEEE4 data type. If no data type is explicitly specified with the AS statement, then FLOAT is assumed.

```
Public Z, RefTemp, TCTemp(3)
Public X AS FLOAT
```

4.4.2 LONG

“AS LONG” specifies the variable as a 32 bit long integer, ranging from –2,147,483,648 to +2,147,483,647 (31 bits plus the sign bit). There are two possible reasons a user would do this: (1) speed, since the OS can do math on integers faster than with floats, and (2) resolution, since the LONG has 31 bits compared to the 24 bits in the IEEE4.

Examples:

```
Dim I AS LONG
Public LongCounter AS LONG
```

4.4.3 BOOLEAN

“AS BOOLEAN” specifies the variable as a 4 byte Boolean. Boolean variables are typically used for flags and to represent conditions or hardware that have only 2 states (e.g., On/Off, Ports). A Boolean variable uses the same 32 bit long integer format as a LONG but can set to only one of two values: True, which is represented as –1, and false, which is represented with 0. The Boolean data type allows application software to display it as an ON/OFF, TRUE/FALSE, RED/BLUE, etc.

```
Public Switches(8) AS Boolean, FLAGS(16) AS Boolean
```

4.4.4 STRING

“AS STRING * size” specifies the variable as a string of ASCII characters, NULL terminated, with size specifying the maximum number of characters in the string. The size argument is optional. The minimum string size, and the default if size is not specified, is 16. A string is convenient in handling serial sensors, dial strings, text messages, etc.

Strings can be dimensioned only up to 2 dimensions instead of the 3 allowed for other data types. (This is because the least significant dimension is actually used as the size of the string.)

```
Public FirstName AS STRING      'Creates string of 16 characters
Public LastName AS STRING * 20 'Creates string of 20 characters
```

4.4.5 Numerical Expressions with Floats, Longs and Booleans

Floats, Longs and Booleans are converted automatically into each other

Boolean from Float or Long

When a float or long integer is converted to a boolean, zero is False (0), Any non-zero value will set the Boolean to True (-1)

```
Public X, Y
Public I AS Long, B AS Boolean
BeginProg
    X = 0
    Y = 0.125
    I = 126
    B = X      'This will set B = False (0)
    B = Y      'This will Set B = True (-1)
    B = I      'This will Set B = True (-1)
EndProg
```

Float from Long or Boolean

When a Long or Boolean is converted to a float, the integer value is loaded into the Float. Booleans will be converted as -1 or 0 depending on whether the value is true or false. Note that integers greater than 24 bits (16,777,215; the size of the mantissa for a Float) will lose resolution when converted to a float.

Long from Float or Boolean

Booleans will be converted as -1 or 0 depending on if the value is true or false. When a Float is converted to a Long integer it is truncated. This conversion is the same as the INT function (Section 8). Note that the integer conversion is to the integer equal to or less than the value of the float. This may not be intuitive for negative numbers, for example:

```
Dim I as Float
BeginProg
    I = 4.6      'This will set I to 4.
    I = -4.6     'This will set I to -5.
EndProg
```

If the Float is greater than maximum long integer, the integer will be set to the maximum (+2,147,483,647). If the float is less than the minimum long integer, the integer is set to the minimum (-2,147,483,648).

Expressions are evaluated as integers for as long as possible

```
Public X, I AS Long
BeginProg
  I = 126
  X = (I+3) * 3.4
      'I+3 is evaluated as an integer,
      'then converted to FLOAT before
      'it is multiplied by 3.4
EndProg
```

Constants will be converted to Longs and/or Floats at compilation

If a constant (either entered as a number or declared with CONST) can be expressed correctly as an integer, the compiler will use the type that is most efficient in each expression. The integer version will be used if possible, i.e., if the expression has not yet confronted a float.

```
Public I AS Long, 'I is an integer
Public F1, F2 'F1 and F2 are Floats
CONST ID = 10
BeginProg
  I = ID * 5      'ID (10) and 5 are loaded at
                  'compile time as Floats
  F1 = F2 + ID   'ID (10) is loaded at compile
                  'time as a float to avoid a
                  'run time conversion from an
                  'integer before each addition
EndProg
```

4.5 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 4.5-1).

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.

4.6 Logical Expression Evaluation

4.6.1 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 4.6-1. The CR3000 evaluates the test or parameter as a number; 0 is false, not equal to 0 is true.

TABLE 4.6-1. 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	$\neq 0$	0
Digital port	5 Volts	0 Volts

4.6.2 Expression Evaluation

Conditional tests require the CR3000 to evaluate an expression and take one path if the expression is true and another if the expression is false. For example:

If $X \geq 5$ then $Y=0$

will set the variable Y to 0 if X is greater than or equal to 5.

The CR3000 will also evaluate multiple expressions linked with **and** or **or**. For example:

If $X \geq 5$ and $Z=2$ then $Y=0$

will only set $Y=0$ if both $X \geq 5$ and $Z=2$ are true.

If $X \geq 5$ or $Z=2$ then $Y=0$

will set $Y=0$ if either $X \geq 5$ or $Z=2$ is true (see And and Or in Section 8). A condition can include multiple **and** and **or** links.

4.6.3 Numeric Results of Expression Evaluation

The CR3000 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 CR3000 expression evaluator evaluates the expression, $X \geq 5$, 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 $X \leq Y$.

The CR3000 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

4.7 Flags

While any variable can be used as a flag as far as logical tests in CRBasic are concerned, it is best to use Boolean variables. 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 4.6). Boolean variables can only have one of two values, true (-1) or false (0).

4.8 Parameter Types

Instruction 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 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 4.8-1 lists the maximum length and allowed characters for the names for Variables, Arrays, Constants, etc.

TABLE 4.8-1. 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	

4.8.1 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 4.6.3). 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.

4.8.2 Arrays of Multipliers and 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.

```

Public Pressure(3), Mult(3), Offset(3)

DataTable (AvgPress,1,-1)
  DataInterval (0,60,Min,10)
  Average (3,Pressure(),IEEE4,0)
EndTable

BeginProg
  'Calibration Factors:
  Mult(1)=0.123 : Offset(1)=0.23
  Mult(1)=0.115 : Offset(1)=0.234
  Mult(1)=0.114 : Offset(1)=0.224

  Scan (1,Sec,10,0)
  'VoltSe instruction using array of multipliers and offsets:
  VoltSe (Pressure(),3,mV5000,1,True,0,_60Hz,Mult(),Offset())
  CallTable AvgPress
  NextScan
EndProg

```

4.9 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(1,2)`

could be used in the example program (Section 4.3) to calculate the change in the 10 minute average temperature of the first thermocouple between the most recent average and the one that occurred ten minutes earlier.

In addition to accessing the data actually output in a table, there is additional information about the data table that can be retrieved using the same `Tablename.fieldname` syntax.

`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.tablesize(1,1) = the size of the table in number of records.

Tablename.tablefull(1,1) = -1 (True) or 0 (False) to indicate if a fill and stop table is full or if a ring memory table has begun overwriting its oldest data. (0 indicates the table is not full or that oldest data has not been overwritten.)

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.

Tablename.eventcount(1,1) = the number of data storage events that have occurred in a data table using the DataEvent instruction.

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* = *AliasName*

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
```

AngleDegrees

The AngleDegrees declaration is used to set math functions in the program to return, or to expect as the source, degrees instead of radians.

Syntax

AngleDegrees

Remarks

The AngleDegrees instruction is placed in the declarations section of the program, before the code enclosed in the BeginProg/EndProg instructions.

AngleDegrees affects the following instructions that return an angle in radians: ATN, ATN2, ACOS, ASIN, RectPolar.

Angle Degrees affects the following instructions that expect an angle in radians as the source: COS, COSH, TAN, TANH, SIN, SINH.

Negative radians will convert to negative degrees.

AS type

The declaration of variables (via the DIM or the PUBLIC statement) allow an optional type descriptor AS that specifies the data type. The default data type, without a descriptor, is IEEE4 floating point (FLOAT). The data types are FLOAT, LONG, BOOLEAN, and STRING.

AS FLOAT specifies the default IEEE4 data type. If no data type is explicitly specified with the AS statement, then FLOAT is assumed.

```
Public Z, RefTemp, TCTemp(3)
Public X AS FLOAT
```

AS LONG specifies the variable as a 32 bit long integer, ranging from –2,147,483,648 to +2,147,483,647 (31 bits plus the sign bit). There are two possible reasons a user would do this: (1) speed, since the OS can do math on integers faster than with floats, and (2) resolution, since the LONG has 31 bits compared to the 24 bits in the IEEE4.

Examples:

```
Dim I AS LONG
Public LongCounter AS LONG
```

AS BOOLEAN specifies the variable as a 4 byte Boolean. Boolean variables are typically used for flags and to represent conditions or hardware that have only 2 states (e.g., On/Off, Ports). A Boolean variable uses the same 32 bit long integer format as a LONG but can set to only one of two values: True, which is represented as –1, and false, which is represented with 0. The Boolean data type allows application software to display it as an ON/OFF, TRUE/FALSE, RED/BLUE, etc.

```
Public Switches(8) AS BOOLEAN, FLAGS(16) AS BOOLEAN
```

AS STRING * *size* specifies the variable as a string of ASCII characters, NULL terminated, with *size* specifying the maximum number of characters in the string. A string is convenient for handling serial sensors, dial strings, text messages, etc.

String arrays can only have up to 2 dimensions instead of the 3 allowed for other data types. (This is because the least significant dimension is actually used as the size of the string.)

```
Public FirstName AS STRING * 20
Public LastName AS STRING * 20
```

Const

Declares symbolic constants for use in place of numeric entries.

Syntax

Const *constantname* = *expression*

Remarks

The **Const** statement has these parts:

Part	Description
<i>constantname</i>	Name of the constant.
<i>expression</i>	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 easier to modify. Unlike variables, constants can't be changed while your program is running.

Caution Constants must be defined before referring to them.

Const Declaration Example

The example uses Const to define 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 memory for the variables. In CRBasic, **ALL** variables **MUST** be declared.

Syntax

Dim *varname*[[*subscripts*]] [, *varname*[[*subscripts*]]]

Remarks

The **Dim** statement has these parts:

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

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

In CRBasic the lowest number in a dimension is 1 not 0.

' Create the variable array A with 8 elements
Dim A(8)

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 set to 0.

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

FormatFloat

Converts a floating point value into a string.

Syntax

String =**FormatFloat** (Float, FormatString)

Remarks

The string conversion of the floating point value is formatted based on the FormatString.

Part

Description

Float The variable or constant that holds the floating point value to be converted.

FormatString how the floating point value will be represented in the converted string. Note that the format string must be enclosed in quotes. The options are (m = mantissa; d = decimal; x = exponent):

Code

Description

%f	Decimal notation in the form of +mmm.ddddd; precision is 6 places to the right of the decimal
%e (or %E)	Decimal notation in the form of +m.ddddd e+xx; precision is 6 places to the right of the decimal
%g (or %G)	Mantissa and decimal are variable; trailing 0s and decimals are omitted
%Y.Zf	Decimal notation in the form of +m.d; precision is defined by Y places to the left of the decimal and Z places to the right of the decimal
%Ye (or %YE)	Decimal notation in the form of +m.d e+xx; precision is defined by Y characters to the right of the decimal
%Yg (or %YG)	Mantissa and decimal are variable; precision is defined by Y.

FormatFloat Example

The following program shows the use of the FormatFloat instruction. Using the software's numeric monitor, enter a value into FloatVal that should be converted into a string value.

```
Public FloatVal as Float
Public StringVal as String * 20

BeginProg
  Scan (1,Sec,3,0)
  StringVal = FormatFloat (FloatVal,"%f")
  NextScan
EndProg
```

INTDV

Performs an integer division of two numbers.

Syntax

X INTDV Y

Remarks

The INTDV function divides one number by another and returns the integer portion of the result. The function can be used in an expression or set equal to a variable (e.g., Result = X INTDV Y).

PipelineMode

The PipelineMode instruction is used to configure the datalogger to separate measurement, processing and SDM communication tasks and execute them concurrently.

Syntax

PipelineMode

Remarks

The datalogger has two processing modes: sequential mode and pipeline mode. In sequential mode, instructions are executed by the datalogger sequentially as they occur in the program. In pipeline mode, measurement tasks and processing tasks are handled separately and executed concurrently.

The default mode of operation is pipeline mode. However, when the datalogger program is compiled, the datalogger analyzes the program instructions and automatically switches to sequential mode if the code requires it. The datalogger can be forced to run in the pipeline or sequential mode by placing the appropriate instruction at the beginning of the program before the BeginProg instruction.

See OV2.3 for more description of the Pipeline and Sequential mode.

Public

Dimensions a variable as public and available in the Public table of the CR3000.

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
```

PWR

Performs an exponentiation on a variable.

Syntax

PWR (X, Y)

Remarks

The PWR function applies the exponent Y to the base X. The function can be used in an expression or set equal to a variable (e.g., Result = PWR(X,Y).

SequentialMode

The SequentialMode instruction is used to configure the datalogger to perform all instructions sequentially as they occur in the program.

Syntax

SequentialMode

Remarks

The datalogger has two processing modes: sequential mode and pipeline mode. In sequential mode, instructions are executed by the datalogger sequentially as they occur in the program. In pipeline mode, measurement tasks and processing tasks are handled separately and executed concurrently.

The default mode of operation is pipeline mode. However, when the datalogger program is compiled, the datalogger analyzes the program instructions and automatically switches to sequential mode if the code requires it. The datalogger can be forced to run in the pipeline or sequential mode by placing the appropriate instruction at the beginning of the program before the BeginProg instruction.

See OV2.3 for more description of the Pipeline and Sequential mode.

StationName

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 in the Status table and stored in the data table headers (Section 2.4).

Sub, ExitSub, EndSub

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.
<i>SubName</i>	Name of the Subroutine. <i>Subname</i> cannot be the same as any other globally recognized name in the program.
<i>VariableList</i>	List of variables that are passed to the Subroutine when it is called. 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).

If the Subroutine variable list is used, the variable names used in this list should not be the same names as variables, aliases, or constants declared elsewhere. 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.

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

```
'CR3000
'Declare Variables used in Program:
Public RefT, TC_C(4), TC_F(4), I

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



```

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

BeginProg
  Scan (1,Sec,3,0)
  'Measure Temperatures (panel and 4 thermocouples) in deg C
  PanelTemp (RefT,250)
  TCDiff (TC_C),4,mV2_5C,1,TypeT,RefT,True ,0,250,1.0,0)
  'Call Output Table for C
  CallTable TempsC

  'Convert Temperatures to F using Subroutine:

  'Subroutine call using Call statement,
  'RefT is used for both source and destination.
  Call ConvertCtoF(RefT, RefT)
  For I = 1 to 4
    'Subroutine call without Call statement:
    ConvertCtoF(TC_C(I),TC_F(I))
  Next I

  CallTable TempsF
NextScan
EndProg

```

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. The unit name appears in the header of the output files. 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 the CRBasic editor or the CR3000.

Example

```

Dim TCTemp( 1 )
Units TCTemp( 1 ) = Deg_C

```


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.

Parameter & Data Type	Enter	
Name <i>Name</i>	The name for the data table. The table name is limited to eight characters.	
TrigVar <i>Constant Variable, or Expression</i>	The name of the variable to test for the trigger. Trigger modifiers add additional conditions.	
	Value	Result
	0	Do not trigger
	≠ 0	Trigger
Size <i>Constant</i>	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.	

DataTable Example - see Section 4.3.

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 CR3000 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 CR3000 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 CR3000 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, which requires an additional 12 bytes per record.

Entering a negative number tells the CR3000 not to keep track of lapses. Only the periodic time stamps (approximately once per K of data) are inserted.

Parameter & Data Type	Enter								
TintoInt <i>Constant</i>	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.								
Interval <i>Constant</i>	Enter the time interval on which the data in the table is to be recorded. The interval may be in μ s, ms, s, or minutes, whichever is selected with the Units parameter. Enter 0 to make the data interval the same as the scan interval.								
Units <i>Constant</i>	The units for the time parameters, PowerOff is the only instruction that uses hours or days. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Code</th> <th>Units</th> </tr> </thead> <tbody> <tr> <td>MSEC</td> <td>milliseconds</td> </tr> <tr> <td>SEC</td> <td>seconds</td> </tr> <tr> <td>MIN</td> <td>minutes</td> </tr> </tbody> </table>	Code	Units	MSEC	milliseconds	SEC	seconds	MIN	minutes
Code	Units								
MSEC	milliseconds								
SEC	seconds								
MIN	minutes								
Lapses <i>Constant</i>	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.								

OpenInterval

When the DataInterval instruction is included in a data table, the CR3000 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 CR3000 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 OpenInterval 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 = 250         'Use 250 s 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,mV20C,9,TypeT,RefTemp,True,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 9) is automatically stored in the table between each event.

Parameter & Data Type	Enter						
RecsBefore Constant	The number of records to store before the Start Trigger.						
StartTrig Variable, or Expression	The variable or expression test to Trigger copying the pre trigger records into the data table and start storing each new record.. <table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Do not trigger</td> </tr> <tr> <td>≠ 0</td> <td>Trigger</td> </tr> </tbody> </table>	Value	Result	0	Do not trigger	≠ 0	Trigger
Value	Result						
0	Do not trigger						
≠ 0	Trigger						
StopTrig Variable, Expression or Constant	The variable, expression or constant to test to stop storing to the data table. The CR3000 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. <table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Do not trigger</td> </tr> <tr> <td>≠ 0</td> <td>Trigger</td> </tr> </tbody> </table>	Value	Result	0	Do not trigger	≠ 0	Trigger
Value	Result						
0	Do not trigger						
≠ 0	Trigger						
RecsAfter Constant	The number of records to store after the Stop Trigger occurs.						

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,mV20C,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 (all data erased) from within the program by executing the ResefTable instruction.

Example:

```

DataTable (Temp,1,2000)
  DataInterval(0,100,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.

To use WorstCase, a DataTable sized to hold one data storage event must be created to act as the event buffer. Data can be stored to the table using the DataEvent instruction or some other trigger condition.

The user must create an algorithm in the program by which to test the WorstCase event. The algorithm should calculate a numerical ranking value of the event, which is stored as a variable. When WorstCase is executed, it checks the ranking variable. If the value of the variable is a new worst case, then the data in the event table will replace the data in the table which holds the least significant event currently stored.

Multiple WorstCase events can be saved. The number of WorstCase events is specified with the NumCases variable. A separate table is created for each of the WorstCase events. These tables use the name of the DataTable with a two-digit number appended to the end (i.e., a table called Temp's WorstCase tables will be named Temp01, Temp02, Temp03...). An additional table, nameWC (e.g., TempWC), stores the rank variable values for each of the worst case tables as well as the last time to which each table was written.

If the data storage event table is stored on a compact flash card all the worst case tables will be stored on the same card. Because the WorstCase instruction requires the capability of erasing and writing over data, CPU Flash memory cannot be used with WorstCase.

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 <i>name</i>	The TableName parameter is the name of the DataTable for which to create WorstCase Events. The length of this name should be 6 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.	
MaxMin <i>Constant</i>	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 <i>Constant</i>	The minimum change that must occur in the RankVariable before a new worst case is stored.	
RankVar <i>Variable</i>	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.

```
'CR3000 Series Datalogger

Const NumCases = 5           'Number of Worst Cases to save
Const Max = 1                'A constant to indicate ranking maximum values in worst case

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 (Evt,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,10,0)
  PanelTemp (RefTemp,250)
  TCDiff(TC(),5,mV2_5C,1,TypeT,RefTemp,True,0,250,1,0)
  CallTable Evt
  IF Evt.EventEnd(1,1) then   '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 Evt.TC(1,I)>=30 'Quit looping when at end or TC(1)<30
    WorstCase(Evt,NumCases,Max,0,NumAbove30) 'Check for worst case
  EndIf
  NextScan
EndProg
```

6.3 Export Data Instructions

CardOut (StopRing, Size)

Used to send output data to a Compact Flash card. This instruction creates a data table on the CF card. CardOut must be entered within each data table declaration that is to store data on a CF card. The data is saved to the card with a name consisting of the datalogger's StationName (if one is assigned), the DataTable name, and a .DAT extension (stationname.tablename.dat). The file will be dated based on the time the program was compiled in the datalogger.

If a file is found on the card with the same name, the data will be appended to that file if the file header is identical. To be identical the file must have been created on the same datalogger using the same program. If it is not identical, the datalogger will generate a compile error and the user must decide whether to delete the old files or rename the new ones.

Parameter & Data Type	Enter						
StopRing <i>Constant</i>	A code to specify if the Data Table on the CF card is fill and stop or ring (newest data overwrites oldest).						
	<table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Ring</td> </tr> <tr> <td>1</td> <td>Fill and Stop</td> </tr> </tbody> </table>	Value	Result	0	Ring	1	Fill and Stop
	Value	Result					
0	Ring						
1	Fill and Stop						
Size <i>Constant</i>	The size to make the data table. The number of data sets (records) to allocate memory for in the CF 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. Enter -1000 to set the size of the table on the card to the size of the table in the datalogger's memory.						

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 <i>Array</i>	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 is Flag ().
Rate <i>Constant</i>	How frequently to send new values to the DSP4 in milliseconds.

DSP4 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 <i>Variable or Array</i>	The variable that holds a result code for the transmission. The result codes are:	
	Result 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; GOES not attached
7	Random message transmit failure (could be no data in buffer)	
Table <i>Table Name</i>	The data table from which record(s) should be transmitted.	
TableOption <i>Constant</i>	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	

Parameter & Data Type	Enter
BufferControl <i>Constant</i>	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
DataFormat <i>Constant</i>	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 RAW57; 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	
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	
For DataFormat options 1, 4, 5, 6, 7, and 8, if the data being transmitted is formatted as a string, the datalogger will send the string. For DataFormat options 0, 2, and 3, the datalogger will search for numeric values, convert them to the appropriate format, and send them to the transmitter.	

GOESGPS (GoesArray1(6), GoesArray2(7))

The GOESGPS instruction is used to store GPS data from the satellite into two variable arrays.

Syntax

GOESGPS (GoesArray1(6), GoesArray2(7))

Remarks

The GOESGPS instruction returns two arrays. The first array, which must be dimensioned to 6, holds a result code indicating the success of the instruction, followed by global positioning information.

The result codes are as follows:

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; GOES not attached
7	ACK not returned following data append or overwrite command

The GPS data values are as follows:

Value	Description
Time	Seconds since January 1, 2000
Latitude	Fractional degrees; 100 nanodegree resolution
Longitude	Fractional degrees; 100 nanodegree resolution
Elevation	Signed 32-bit number, in centimeters
Magnetic Variation	Fractional degrees; 1 millidegree resolution

The second array, which must be dimensioned to 7, holds the following time values: year, month, day hour (GMT), minute seconds, microseconds.

GOESSetup (ResultCode, PlatformID, MsgWindow, STChannel, STBaud, RChannel, RBaud, STInterval, STOffset, RInterval)

The GOESSetup instruction is used to program the GOES transmitter for communication with the satellite.

Syntax

GOESSetup (ResultCode, PlatformID, MsgWindow, STChannel, STBaud, RChannel, RBaud, STInterval, STOffset, RInterval)

Remarks

Since the purpose of this instruction is to set up the transmitter for communication, it only has to be run once within the datalogger program (i.e. it can be placed between the BeginProg and Scan instructions). Information for all parameters in this instruction is supplied by NESDIS.

See the CRBasic Editor or the SATHDRGOES manual for more detailed information on the instruction.

GOESStatus (ResultCode, 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																							
ResultCode <i>Array</i>	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.																							
	<table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Command executed successfully</td> </tr> <tr> <td>1</td> <td>Checksum failure in response</td> </tr> <tr> <td>2</td> <td>Timeout waiting for STX character after SDC addressing</td> </tr> <tr> <td>3</td> <td>Wrong character (not STX) received after SDC addressing</td> </tr> <tr> <td>4</td> <td>Received a NAK</td> </tr> <tr> <td>5</td> <td>Timed out waiting for ACK</td> </tr> <tr> <td>6</td> <td>CS I/O port not available</td> </tr> <tr> <td>7</td> <td>Transmit random message failure, could be no data</td> </tr> <tr> <td>9</td> <td>Invalid command</td> </tr> </tbody> </table>	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			
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																							
StatusCommand <i>Constant</i>	The StatusCommand specifies the type of information requested from the transmitter.																							
	<table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> <th>Array Dim Required</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Read time</td> <td>4</td> </tr> <tr> <td>1</td> <td>Status</td> <td>13</td> </tr> <tr> <td>2</td> <td>Last message status</td> <td>14</td> </tr> <tr> <td>3</td> <td>Transmit random message</td> <td>1</td> </tr> <tr> <td>4</td> <td>Read error register</td> <td>10</td> </tr> <tr> <td>5</td> <td>Reset error register</td> <td>1</td> </tr> <tr> <td>6</td> <td>Return transmitter to online mode</td> <td>1</td> </tr> </tbody> </table>	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
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 <i>Constant</i>	The number of averages to calculate. When Reps is greater than one, the source must be an array.
Source <i>Variable</i>	The name of the Variable that is to be averaged.

Parameter & Data Type	Enter	
Data Type <i>Constant</i>	A code to select the data storage format.	
	Code	Data Format
	IEEE4	IEEE 4 byte floating point
	FP2	Campbell Scientific 2 byte floating point
DisableVar <i>Constant, Variable, or Expression</i>	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 (DimX, XVal, DataType, DisableVar, NumOfCov)

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

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

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	
DimX <i>Constant</i>	The number of elements in the array to include in the covariance calculations	
XVal <i>Variable Array</i>	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)).	
Data Type <i>Constant</i>	A code to select the data storage format.	
	Alpha Code	Data Format
	IEEE4	IEEE 4 byte floating point
	FP2	Campbell Scientific 2 byte floating point
DisableVar <i>Constant, Variable, or Expression</i>	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
NumOfCov <i>Constant</i>	The number of covariances to calculate. The maximum number of covariances is $Z/2*(Z+1)$. Where $Z = \mathbf{NumVals}$. 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 (SecArray, DataType, N, SampleInterval, 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		
SrcArray <i>Variable</i>	The name of the Variable array that contains the input data for the FFT.		
DataType <i>Constant</i>	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 <i>Constant</i>	Number of points in the original time series. The number of points must be a power of 2 (i.e., 512, 1024, 2048, etc.).		
SampleInterval <i>Constant</i>	The sampling interval of the time series.		
Units <i>Constant</i>	The units for Tau.		
	Alpha Code	Units	
	USEC	microseconds	
	MSEC	milliseconds	
	SEC	seconds	
	MIN	Minutes	
	HR	Hours	
Option <i>Constant</i>	A code to indicate what values to calculate and output.		
	Code	Result	
	0	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.	
	1	Amplitude spectrum. The output is N/2 magnitudes. With $Acos(wt)$; A is magnitude.	
	2	Amplitude and Phase Spectrum. The output is N/2 pairs of magnitude and phase; with $Acos(wt - \phi)$; A is amplitude, ϕ is phase $(-\pi, \pi)$.	
	3	Power Spectrum. The output is N/2 values normalized to give a power spectrum. With $Acos(wt - \phi)$, the power is $A^2 / 2$. The summation of the N/2 values yields the total power in the time series signal.	
	4	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.	
5	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.		

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

Normalization details:

Complex FFT result i , $i = 1 .. N/2$: $a_i * \cos(w_i * t) + b_i * \sin(w_i * t)$.
 $w_i = 2\pi(i-1)/T$.
 $\phi_i = \text{atan2}(b_i, a_i)$ (4 quadrant arctan)
 $\text{Power}(1) = (a_1^2 + b_1^2)/N^2$ (DC)
 $\text{Power}(i) = 2 * (a_i^2 + b_i^2)/N^2$ ($i = 2..N/2$, AC)
 $\text{PSD}(i) = \text{Power}(i) * T = \text{Power}(i) * N * \text{tau}$
 $A_1 = \text{sqrt}(a_1^2 + b_1^2)/N$ (DC)
 $A_i = 2 * \text{sqrt}(a_i^2 + b_i^2)/N$ (AC)

Notes:

- Power is independent of the sampling rate ($1/\text{tau}$) and of the number of samples (N).
- The PSD is proportional to the length of the sampling period ($T=N*\text{tau}$), 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

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,IIEEE4,SIZE_FFT,10, msec,0)
EndTable
DataTable(IFFT,1,1)           'inverse FFT
    fft(y,IIEEE4,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("Fieldname1:Description1,Fieldname2:Description2...")

The FieldNames instruction is used immediately following an output processing instruction to change the default field names that the datalogger generates for results sent to the DataTable. The names are entered in the form of "Fieldname:Description". The fieldname and description must be separated by a colon, and the entire string must be enclosed in quotation marks.

The Fieldname element is the name to be used for the field; field names are limited to 19 characters. The Description element (which is optional) provides a way for the user to include further information about the field. In a collected data file, the Description is included in the header line below the Fieldname, along with the processing description. The maximum number of characters in the Description depends upon the description that is automatically generated for the processing type. The total maximum characters for the two is 65,

including quotation marks, spaces, and other characters. As an example, the processing description and fieldname description for a Sample instruction might look like the following:

Smp,;This is a sample air temp"

Smp,; and the opening and closing quotation marks use 7 characters; therefore, there are 58 remaining for the fieldname description.

If an output instruction generates multiple fields, individual names may be entered for each or an array may be used. Individual names should be separated by commas. If an array is used, the array name and dimension must be specified (i.e., "Temp(4)" specifies an array of 4 field names; Temp(1) through Temp(4)). Note that an expression which evaluates as a constant can also be used to specify the array dimension. When the program is compiled, the datalogger 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 field names, the default names are used for the remaining fields.

Examples

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)

This instruction 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 <i>Variable or Array</i>	The variable that is tested to determine which bin is selected. The Histogram4D instruction requires an array dimensioned with at least as many elements as histogram dimensions.														
DataType <i>Constant</i>	A code to select the data storage format. <table border="1"> <thead> <tr> <th>Alpha Code</th> <th>Data Format</th> </tr> </thead> <tbody> <tr> <td>IEEE4</td> <td>IEEE 4 byte floating point</td> </tr> <tr> <td>FP2</td> <td>Campbell Scientific 2 byte floating point</td> </tr> </tbody> </table>	Alpha Code	Data Format	IEEE4	IEEE 4 byte floating point	FP2	Campbell Scientific 2 byte floating point								
Alpha Code	Data Format														
IEEE4	IEEE 4 byte floating point														
FP2	Campbell Scientific 2 byte floating point														
DisableVar <i>Constant, Variable, or Expression</i>	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is $\neq 0$ 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 border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Process current input</td> </tr> <tr> <td>$\neq 0$</td> <td>Do not process current input</td> </tr> </tbody> </table>	Value	Result	0	Process current input	$\neq 0$	Do not process current input								
Value	Result														
0	Process current input														
$\neq 0$	Do not process current input														
Bins <i>Constant</i>	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 <i>Constant</i>	The Form argument is 3 digits - ABC <table border="1"> <thead> <tr> <th>Code</th> <th>Form</th> </tr> </thead> <tbody> <tr> <td>A = 0</td> <td>Reset histogram after each output.</td> </tr> <tr> <td>A = 1</td> <td>Do not reset histogram.</td> </tr> <tr> <td>B = 0</td> <td>Divide bins by total count.</td> </tr> <tr> <td>B = 1</td> <td>Output total in each bin.</td> </tr> <tr> <td>C = 0</td> <td>Open form. Include outside range values in end bins.</td> </tr> <tr> <td>C = 1</td> <td>Closed form. Exclude values outside range.</td> </tr> </tbody> </table> 101 means: Do not reset. Divide bins by total count. Closed form.	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.
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.														
WtVal <i>Constant or Variable</i>	The variable name of the weighted value. Enter a constant for a frequency distribution of the BinSelect value.														
LoLim <i>Constant</i>	The lower limit of the range covered by the bin select value.														
UpLim <i>Constant</i>	The upper limit of the range of the bin select value.														

Histogram4D (BinSelect, 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 car 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, SecondDim, CrossingArray, 2ndArray, Hysteresis, Option)

Parameter & Data Type	Enter														
Source <i>Variable or Array</i>	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.														
DataType <i>Constant</i>	A code to select the data storage format. <table border="1"> <thead> <tr> <th>Alpha Code</th> <th>Data Format</th> </tr> </thead> <tbody> <tr> <td>IEEE4</td> <td>IEEE 4 byte floating point</td> </tr> <tr> <td>FP2</td> <td>Campbell Scientific 2 byte floating point</td> </tr> </tbody> </table>	Alpha Code	Data Format	IEEE4	IEEE 4 byte floating point	FP2	Campbell Scientific 2 byte floating point								
Alpha Code	Data Format														
IEEE4	IEEE 4 byte floating point														
FP2	Campbell Scientific 2 byte floating point														
DisableVar <i>Constant, Variable, or Expression</i>	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 border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Process current input</td> </tr> <tr> <td>≠ 0</td> <td>Do not process current input</td> </tr> </tbody> </table>	Value	Result	0	Process current input	≠ 0	Do not process current input								
Value	Result														
0	Process current input														
≠ 0	Do not process current input														
NumLevels <i>Constant</i>	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.														
SecondDim <i>Constant</i>	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.														
Crossing Array <i>Array</i>	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.														
SecondArray <i>Array</i>	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.														
Hysteresis <i>Constant</i>	The minimum change in the source that must occur for a crossing to be counted.														
Option <i>Constant</i>	The Option code is 3 digits - ABC <table border="1"> <thead> <tr> <th>Code</th> <th>Form</th> </tr> </thead> <tbody> <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> </tbody> </table> <p>101 means: Count on rising edge, reset count to 0 after each output, output counts.</p>	Code	Form	A = 0	Count on falling edge (source goes form > level to <level)	A = 1	Count on rising edge (source goes from < level to >level)	B = 0	Reset histogram counts to 0 after each output.	B = 1	Do not reset histogram; continue to accumulate counts.	C = 0	Divide count in each bin by total number of counts in all bins.	C = 1	Output total counts in each bin.
Code	Form														
A = 0	Count on falling edge (source goes form > level to <level)														
A = 1	Count on rising edge (source goes from < level to >level)														
B = 0	Reset histogram counts to 0 after each output.														
B = 1	Do not reset histogram; continue to accumulate counts.														
C = 0	Divide count in each bin by total number of counts in all bins.														
C = 1	Output total counts in each bin.														

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*SecondDim. For a one dimensional level crossing histogram, enter 1 for SecondDim.

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).

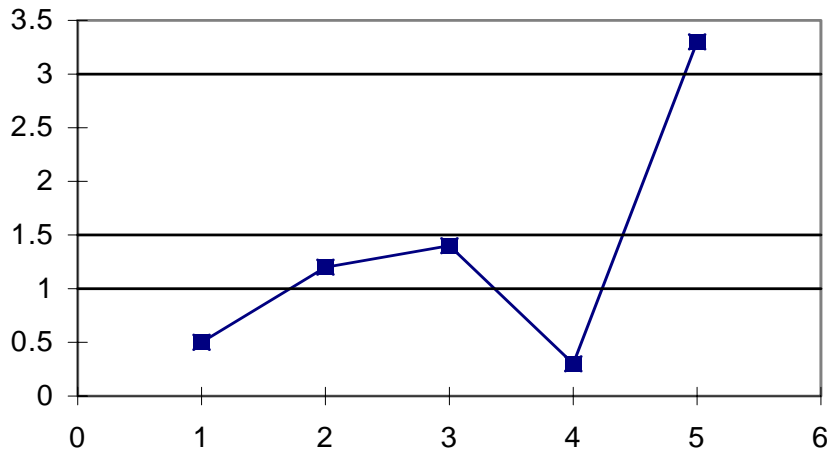


FIGURE 6.4-1. Example Crossing Data

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.

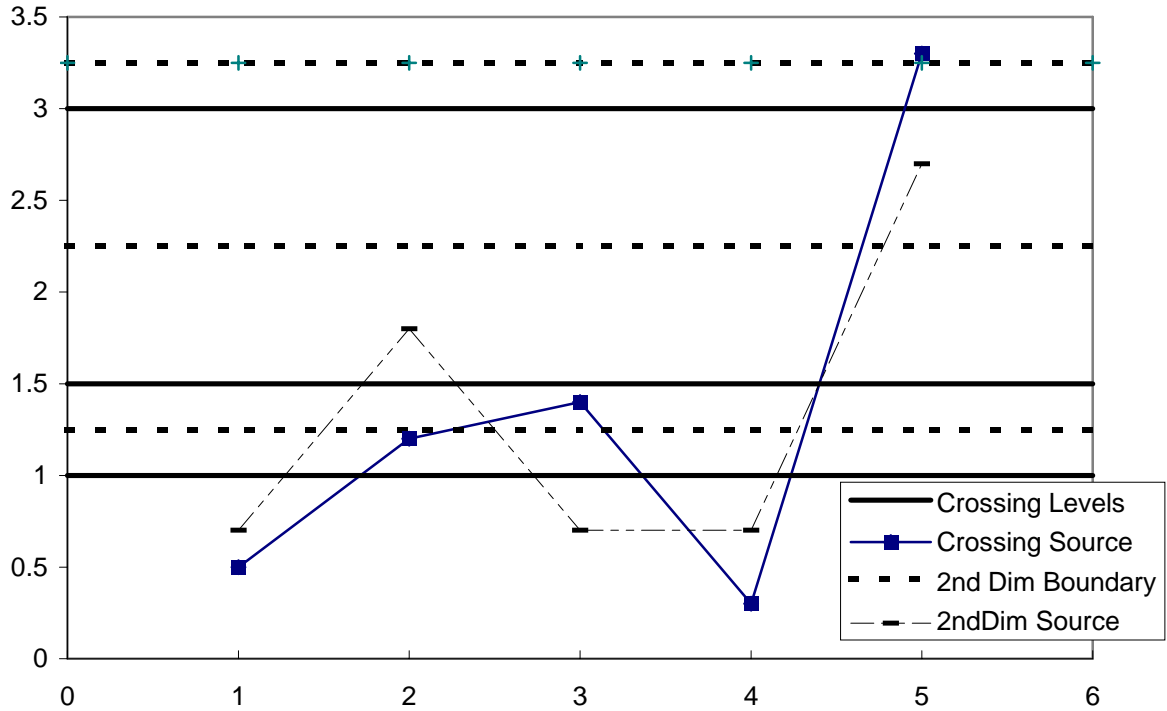


FIGURE 6.4-2. Crossing Data with Second Dimension Value

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 <i>Constant</i>	The number of maximum values to determine. When repetitions are greater than 1, the source must be an array.						
Source <i>Variable</i>	The name of the Variable that is the input for the instruction.						
Data Type <i>Constant</i>	A code to select the data storage format. <table border="1"> <thead> <tr> <th>Alpha Code</th> <th>Data Format</th> </tr> </thead> <tbody> <tr> <td>IEEE4</td> <td>IEEE 4 byte floating point</td> </tr> <tr> <td>FP2</td> <td>Campbell Scientific 2 byte floating point</td> </tr> </tbody> </table>	Alpha Code	Data Format	IEEE4	IEEE 4 byte floating point	FP2	Campbell Scientific 2 byte floating point
Alpha Code	Data Format						
IEEE4	IEEE 4 byte floating point						
FP2	Campbell Scientific 2 byte floating point						
DisableVar <i>Constant, Variable, or Expression</i>	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is $\neq 0$ 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. <table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Process current input</td> </tr> <tr> <td>$\neq 0$</td> <td>Do not process current input</td> </tr> </tbody> </table>	Value	Result	0	Process current input	$\neq 0$	Do not process current input
Value	Result						
0	Process current input						
$\neq 0$	Do not process current input						
Time <i>Constant</i>	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. <table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Do not store time</td> </tr> <tr> <td>1</td> <td>Store time</td> </tr> </tbody> </table>	Value	Result	0	Do not store time	1	Store time
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 in the time parameter.

Parameter & Data Type	Enter						
Reps <i>Constant</i>	The number of minimum values to determine. When repetitions are greater than 1, the source must be an array.						
Source <i>Variable</i>	The name of the Variable that is the input for the instruction.						
Data Type <i>Constant</i>	A code to select the data storage format. <table border="1"> <thead> <tr> <th>Alpha Code</th> <th>Data Format</th> </tr> </thead> <tbody> <tr> <td>IEEE4</td> <td>IEEE 4 byte floating point</td> </tr> <tr> <td>FP2</td> <td>Campbell Scientific 2 byte floating point</td> </tr> </tbody> </table>	Alpha Code	Data Format	IEEE4	IEEE 4 byte floating point	FP2	Campbell Scientific 2 byte floating point
Alpha Code	Data Format						
IEEE4	IEEE 4 byte floating point						
FP2	Campbell Scientific 2 byte floating point						
DisableVar <i>Constant, Variable, or Expression</i>	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is $\neq 0$ 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. <table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Process current input</td> </tr> <tr> <td>$\neq 0$</td> <td>Do not process current input</td> </tr> </tbody> </table>	Value	Result	0	Process current input	$\neq 0$	Do not process current input
Value	Result						
0	Process current input						
$\neq 0$	Do not process current input						
Time <i>Constant</i>	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. <table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Do not store time</td> </tr> <tr> <td>1</td> <td>Store time</td> </tr> </tbody> </table>	Value	Result	0	Do not store time	1	Store time
Value	Result						
0	Do not store time						
1	Store time						

RainFlow (Source, DataType, DisableVar, MeanBins, AmpBins, LoLim, PuLim, 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	Data Format
	IEEE4 FP2	IEEE 4 byte floating point 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 ≠ 0	Process current input 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 UpLim - LoLim 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 UpLim - LoLim divided by the number of bins.	
LoLim 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.	
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.		

This instruction processes data with the rainflow counting algorithm, essential to estimating cumulative damage fatigue to components undergoing stress/strain cycles.

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 $(C_{1,1}, C_{1,2}, \dots, C_{1,N}, C_{2,1}, C_{2,2}, \dots, C_{M,N})$. Multiple repetitions are sequential in memory. Shown in two dimensions, the output is:

$C_{1,1}$	$C_{1,2}$.	.	.	$C_{1,N}$
$C_{2,1}$	$C_{2,2}$.	.	.	$C_{2,N}$
.
.
.
$C_{M,1}$	$C_{M,2}$.	.	.	$C_{M,N}$

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 ($[\text{high limit} - \text{low limit}] / \text{no. amplitude bins}$) 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

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 <i>Constant</i>	The number of values to sample. When repetitions are greater than 1, the source must be an array.
Source <i>Variable</i>	The name of the Variable to sample.
DataType <i>Constant</i>	A code to select the data storage format.
	Alpha Code Data Format
	IEEE4 IEEE 4 byte floating point
	FP2 Campbell Scientific 2 byte floating point
	String ASCII string; size defined by program
	Boolean 0 = False; -1 = True

SampleMaxMin (Reps, Source, DataType, DisableVar)

The SampleMaxMin instruction is used to sample one or more variable(s) when another variable (or any variable in an array of variables) reaches its maximum or minimum for the defined output period.

The SampleMaxMin instruction is placed inside a DataTable declaration, following the Maximum or Minimum instruction that will be used trigger the sample. SampleMaxMin samples whenever a new maximum or minimum is detected in the preceding instruction. When a new sample is taken, the previous value(s) are discarded. The sample(s) recorded in the data table will be those taken when the last maximum or minimum occurred.

When the Repetitions parameter for the Maximum or Minimum instruction is greater than 1, SampleMaxMin samples whenever a new maximum or minimum occurs in any of the variables in the Maximum/Minimum source array. To ensure the sample is taken only when a new maximum or minimum occurs in a single specific variable, the preceding maximum or minimum instruction must have repetitions=1.

Parameter & Data Type	Enter
Reps <i>Constant</i>	The number of values to sample. When repetitions are greater than 1, the source must be an array.
Source <i>Variable</i>	The Source is the name of the variable or variable array that is sampled when a new maximum or minimum occurs for the preceding Maximum or Minimum instruction.
DataType <i>Constant</i>	Select the format in which to save the data
	Entry Description
	IEEE4 IEEE four-byte floating point
	FP2 Campbell Scientific two-byte floating point

Parameter & Data Type	Enter
DisableVar <i>Constant, Variable or Expression</i>	The DisableVar is a Constant, Variable, or Expression that is used to determine whether the current measurement is included in the values to evaluate for a maximum or minimum
	Value Result
	0 Process current input
	≠0 Do not process current input

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 <i>Constant</i>	The number of standard deviations to calculate. When repetitions are greater than 1, the source must be an array.
Source <i>Variable</i>	The name of the Variable that is the input for the instruction.
DataType <i>Constant</i>	A code to select the data storage format.
	Alpha Code Data Format
	IEEE4 IEEE 4 byte floating point FP2 Campbell Scientific 2 byte floating point
DisableVar <i>Constant, Variable, or Expression</i>	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. The data table in which the Totalize instruction appears must be called every scan for Totalize to work correctly. Do not put the CallTable instruction for that table in a conditional statement.

Parameter & Data Type	Enter
Reps <i>Constant</i>	The number of totals to calculate. When repetitions are greater than 1, the source must be an array.
Source <i>Variable</i>	The name of the Variable that is the input for the instruction.

Parameter & Data Type	Enter
Data Type <i>Constant</i>	A code to select the data storage format.
	Alpha Code Data Format
	IEEE4 IEEE 4 byte floating point FP2 Campbell Scientific 2 byte floating point
DisableVar <i>Constant, Variable, or Expression</i>	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.

Parameter & Data Type	Enter
Repetitions <i>Constant</i>	The number of wind sets (speed/direction or East/North) to calculate results for.
Speed/East Dir/North <i>Variables or Arrays</i>	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.
Data Type <i>Constant</i>	A code to select the data storage format.
	Alpha Code Data Format
	IEEE4 IEEE 4 byte floating point FP2 Campbell Scientific 2 byte floating point
DisableVar <i>Constant, Variable, or Expression</i>	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
Subinterval <i>Constant</i>	The number of samples per sub-interval calculation. Enter 0 for no sub-interval calculations.
SensorType <i>Constant</i>	The type of wind sensors
	Value Sensor Type
	0 Speed and Direction 1 East and North

Parameter & Data Type	Enter	
OutputOpt Constant	Value	Outputs (for each rep)
	0	<ol style="list-style-type: none"> 1. Mean horizontal wind speed, S. 2. Unit vector mean wind direction, $\Theta 1$. 3. Standard deviation of wind direction, $\sigma(\Theta 1)$. <p>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.</p>
	1	<ol style="list-style-type: none"> 1. Mean horizontal wind speed, S. 2. Unit vector mean wind direction, $\Theta 1$.
2	<ol style="list-style-type: none"> 1. Mean horizontal wind speed, S. 2. Resultant mean wind speed, \bar{U}. 3. Resultant mean wind direction, Θu. 4. Standard deviation of wind direction, $\sigma(\Theta u)$. <p>This standard deviation is calculated using Campbell Scientific's wind speed weighted algorithm.</p> <p>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.</p>	

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 transition¹.

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

$$\sigma(\Theta) = [((\sigma_{\Theta 1})^2 + (\sigma_{\Theta 2})^2 \dots + (\sigma_{\Theta M})^2) / M]^{1/2}$$

where $\sigma(\Theta)$ is the standard deviation over the output interval, and $\sigma_{\Theta 1} \dots \sigma_{\Theta M}$ are sub-interval standard deviations.

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

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

$$\text{Desired sub-interval (secs)} / \text{scan rate (secs)}$$

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
- U_{e_i} = east-west component of wind
- U_{n_i} = north-south component of wind
- N = number of samples

Calculations:

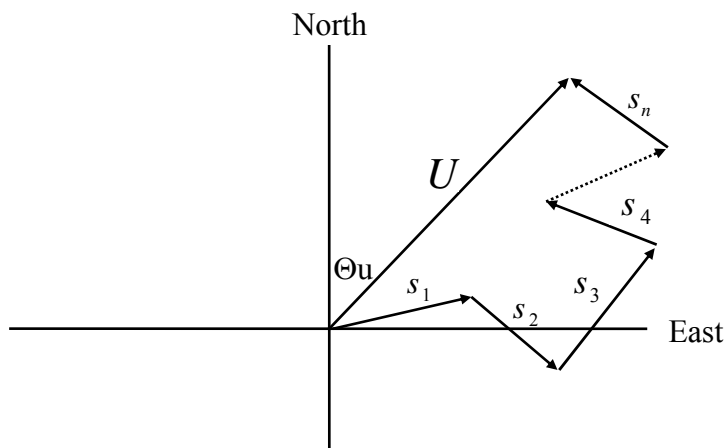


FIGURE 6.4-3. Input Sample Vectors

In Figure 6.4-3, 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 U_{e_i} and U_{n_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 Θ_u . If the input sample interval is t , the number of samples in output interval T is $N = T / t$. The mean vector magnitude is $\bar{U} = U / N$.

Scalar mean horizontal wind speed, S:

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

where in the case of orthogonal sensors:

$$S_i = (U_{e_i}^2 + U_{n_i}^2)^{1/2}$$

Unit vector mean wind direction, Θ_1 :

$$\Theta_1 = \text{Arctan}(U_x/U_y)$$

where

$$U_x = (\sum \sin \Theta_i)/N$$

$$U_y = (\sum \cos \Theta_i)/N$$

or, in the case of orthogonal sensors

$$U_x = (\sum (U_{e_i}/U_i))/N$$

$$U_y = (\sum (U_{n_i}/U_i))/N$$

$$\text{where } U_i = (U_{e_i}^2 + U_{n_i}^2)^{1/2}$$

Standard deviation of wind direction, $\sigma(\Theta_1)$, using Yamartino algorithm:

$$\sigma(\Theta_1) = \text{arc sin}(\varepsilon)[1 + 0.1547 \varepsilon^3]$$

where,

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

and U_x and U_y are as defined above.

Resultant mean horizontal wind speed, \bar{U} :

$$\bar{U} = (U_e^2 + U_n^2)^{1/2}$$

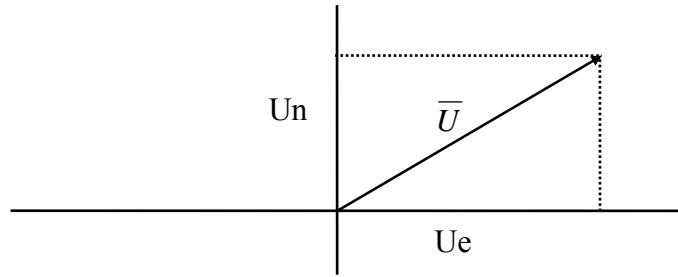


FIGURE 6.4-4. Mean Wind Vector

where for polar sensors:

$$U_e = (\sum S_i \sin \Theta_i)/N$$

$$U_n = (\sum S_i \cos \Theta_i)/N$$

or, in the case of orthogonal sensors:

$$U_e = (\sum U_{e_i})/N$$

$$U_n = (\sum U_{n_i})/N$$

Resultant mean wind direction, Θ_u :

$$\Theta_u = \text{Arctan}(U_e/U_n)$$

Standard deviation of wind direction, $\sigma(\Theta_u)$, using Campbell Scientific algorithm:

$$\sigma(\Theta_u) = 81(1 - \bar{U}/S)^{1/2}$$

The algorithm for $\sigma(\theta u)$ is developed by noting (Figure 6.4-4) that

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

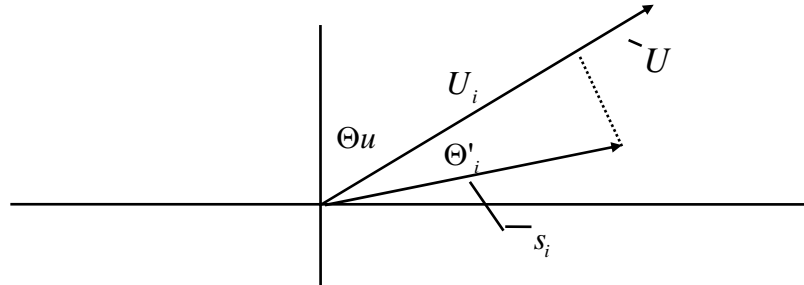


FIGURE 6.4-5. Standard Deviation of Direction

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 $\text{Cos}(\theta')$ 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.

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

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

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

have never been greater than a few degrees.

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

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

Section 7. Measurement Instructions

7.1 Voltage Measurements

VoltDiff – Differential Voltage Measurement.....	7-3
VoltSE – Single-ended Voltage Measurement.....	7-3

7.2 Thermocouple Measurements

Measure the Output of Thermocouples and Convert to Temperature.	
TCDiff – Differential Voltage Measurement of Thermocouple	7-3
TCSE – Single-ended Voltage Measurement of Thermocouple.....	7-4

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 gages, and pressure and force transducers.

7.3 Half Bridges

BrHalf – Half Bridge	7-5
BrHalf3W – Three Wire Half Bridge	7-6
BrHalf4W – Four Wire Half Bridge.....	7-6

7.4 Full Bridges

BrFull – Four Wire Full Bridge.....	7-8
BrFull6W – Six Wire Full Bridge	7-8

7.5 Excitation

ExciteCAO.....	7-10
ExciteI.....	7-11
ExciteV - Sets the specified switched voltage excitation channel to the voltage specified	7-11
SW12	7-12

7.6 Self Measurements

Battery – Measures battery voltage	7-12
Calibrate – Adjusts the calibration for analog measurements.....	7-12
InstructionTimes - Returns the execution time of each instruction	7-15
PanelTemp – (Used as a reference for thermocouple measurements)	7-16

7.7 Digital I/O

CheckPort - Returns the status of a control port.....	7-16
PeriodAvg - Measures the period or the frequency of a signal on a single-ended channel	7-16
PortsConfig - Used to configure one or more control ports as either input or output	7-19

PortGet - Reads the status of one of the eight control ports.....	7-19
PortSet – Sets Digital Ports.....	7-20
PulseCount – Pulse/Frequency Measurement.....	7-20
PulseCountReset - Resets Pulse Counters and Running Averages Used in Pulse Count Instruction	7-23
PulsePort.....	7-23
ReadIO – Reads State of Digital I/O Ports.....	7-23
SDI12Record - Used to retrieve the results from an SDI-12 sensor	7-24
Vibrating Wire	7-25
WriteIO – Sets Digital Outputs.....	7-26

7.8 Specific Sensors

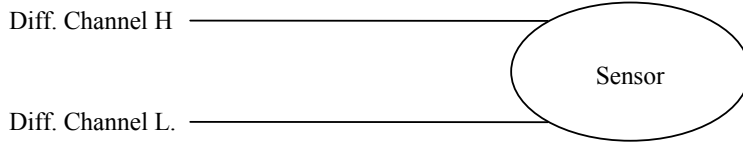
CS110.....	7-27
CS616.....	7-27
HydraProbe	7-28
SlowAntenna.....	7-29
Therm107, Therm108, Therm109.....	7-29

7.9 Peripheral Devices

AM25T - Controls the AM25T multiplexer.....	7-30
CS7500 - Communicates with the CS7500 open path CO ₂ sensor	7-31
CSAT3 - Communicates with the CSAT3 sonic anemometer	7-32
SDMAO4 – Sets the voltage to an SDM-AO4 output device.....	7-33
SDMCAN – Measures and controls the SDM-CAN interface	7-34
SDMCD16AC – Controls an SDM-CD16AC, SDMCD16, or SDM-CD16D 16 channel relay/control port device.....	7-40
SDMINT8 – Allows the use of the SDM-INT8, 8 channel interval timer, with the CR3000	7-40
SDMIO16.....	7-43
SDMSIO4 – Controls and transmits/retrieves data from a CSI SDM-SIO4 interface	7-47
SDMSpeed - Changes the rate that the CR3000 uses to clock the SDM data.....	7-48
SDMSW8A – Controls the SDM-SW8A 8-channel switch closure module	7-48
SDMTrigger - Allows the CR3000 to synchronize when measurements are made.....	7-49
SDMX50.....	7-50
TDR100	7-50

7.1 Voltage Measurements

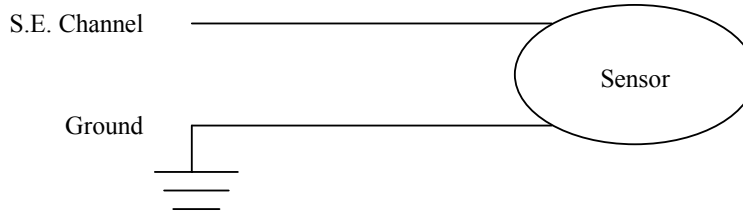
VoltDiff (Dest, Reps, Range, DiffChan, RevDiff, SettlingTime, Integ, Mult, Offset)



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

See section 7.2 for parameter descriptions.

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

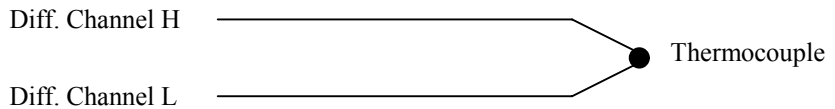


This instruction measures the voltage at a single ended input with respect to ground. With a multiplier of one and an offset of zero, the result is in millivolts or volts depending on the range selected.

See section 7.2 for parameter descriptions.

7.2 Thermocouple Measurements

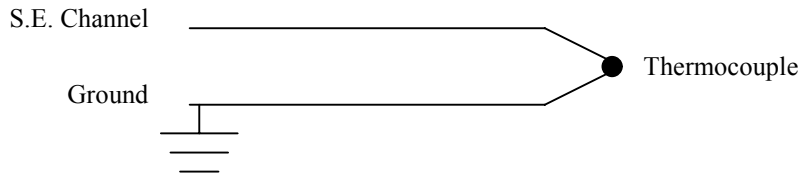
TCDiff (Dest, Reps, Range, DiffChan, TCTYPE, TRef, RevDiff, SettlingTime, Integ, Mult, Offset)



This instruction measures a thermocouple with a differential voltage measurement and calculates the thermocouple temperature ($^{\circ}C$) for the thermocouple type selected. The instruction adds the measured voltage to the voltage calculated for the reference temperature relative to $0^{\circ}C$, and converts the combined voltage to temperature in $^{\circ}C$. The ranges that are specified with a code ending in C (e.g., mV20C) briefly ($10 \mu s$) 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, MeasOff, 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.

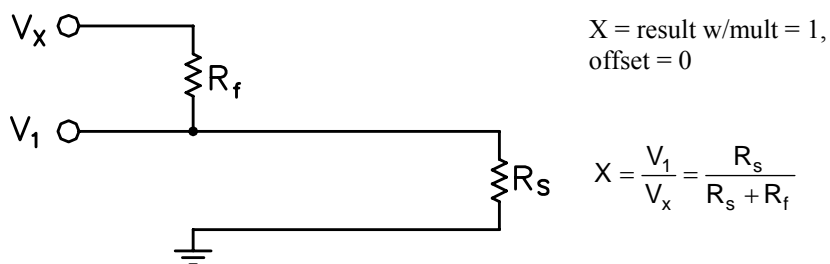
Voltage and Thermocouple Parameters

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	Voltage Range	
	mV5000	± 5000 mV	
	mV1000	± 1000 mV	
	mV200	± 200 mV	
	mV50	± 50 mV	
	mV20	± 20 mV	
	Autorange	mV20 – mV5000	Selects range (Sect. 3.1)
	mV200C	± 200 mV	The mV200C, mV50C, and mV20C
	mV50C	± 50 mV	ranges pull the channel into common
	mV20C	± 20 mV	mode range and check for open input
AutorangeC	mV20 – mV200	Selects C range	
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 channels. If the channel is entered as a negative number, all reps occur on that channel.		
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 channels. If the channel is entered as a negative number, all reps occur on that channel.		
TCType Constant	Alpha Code	Thermocouple Type	
	TypeT	Copper Constantan	
	TypeE	Chromel Constantan	
	TypeK	Chromel Alumel	
	TypeJ	Iron Constantan	
	TypeB	Platinum Rhodium	
	TypeR	Platinum Rhodium	
	TypeS	Platinum Rhodium	
TypeN	NiCrSi-NiSi		

Parameter & Data Type	Enter		
TRef <i>Variable</i>	The name of the variable that is the reference temperature for the thermocouple measurements.		
RevDiff <i>Constant</i>	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
MeasOfs <i>Constant</i>	Code	Value	Result the Ground offset voltage is subtracted from single ended measurements.
	False	0	Offset voltage is corrected from background calibration
	True	≠0	Offset voltage is measured each scan
SettlingTime <i>Constant</i>	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	All	250 μS 450 μS (default)
	0	All	_50Hz, _60 Hz 3 mS (default)
	>=100	All	All μS entered
Integ <i>Constant</i>	The time spent on integration in microseconds for each of the channels measured.		
	Entry	Integration	
	250	250 μS	
	_60Hz or 16667	16,667 μS (reject 60 Hz noise)	
	_50 Hz or 20000	20,000 μS (reject 50 Hz noise)	
Mult, Offset <i>Constant, Variable, Array, or Expression</i>	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 zero 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.		

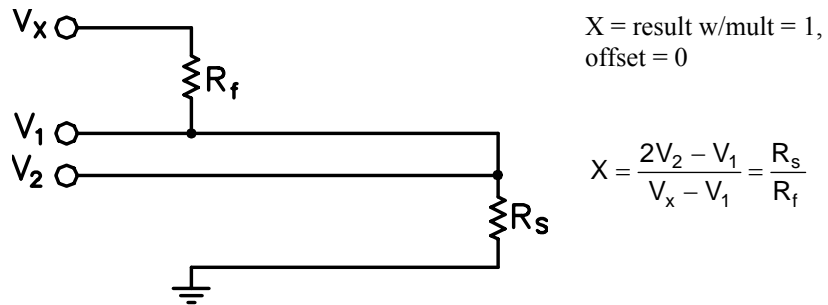
7.3 Half Bridges

BrHalf (Dest, Repts, Range, SEChan, ExChan, MeasPEX, ExmV, RevEx, SettlingTime, Integ, Mult, Offset)



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.

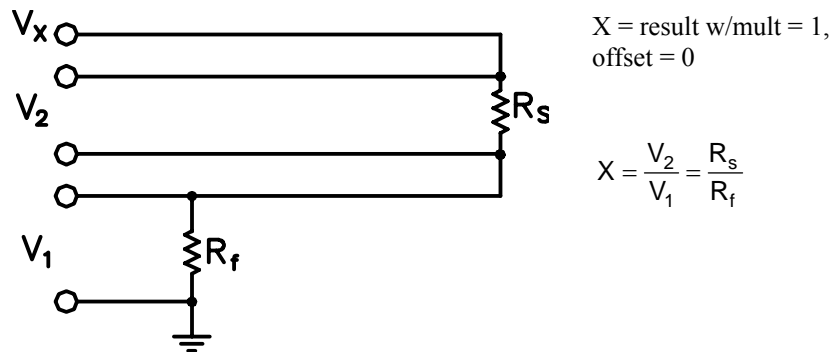
BrHalf3W (Dest, Reps, Range, SEChan, ExChan, MeasPEX, ExmV, RevEx, SettlingTime, Integ, Mult, Offset)



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)



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 .

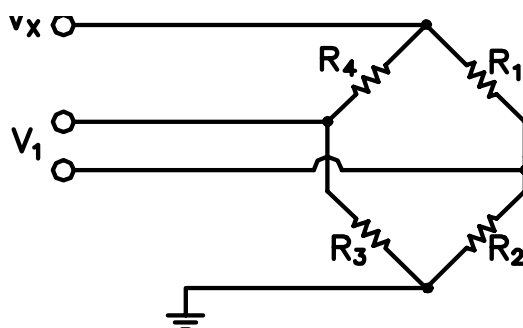
Half Bridge Parameters

Parameter & Data Type	Enter			
Dest <i>Variable or Array</i>	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 <i>Constant</i>	The number of repetitions for the measurement or instruction.			
Range <i>Constant</i>	Alpha Code	Voltage Range		
	mV5000	± 5000 mV		
	mV1000	± 1000 mV		
	mV200	± 200 mV		
	mV50	± 50 mV		
	mV20	± 20 mV		
	Autorange	mV20 – mV5000	Selects range (Sect. 3.1)	
	mV200C	± 200 mV	The mV200C, mV50C, and mV20C	
	mV50C	± 50 mV	ranges pull the channel into common	
	mV20C	± 20 mV	mode range and check for open input	
AutorangeC	mV20 – mV200	Selects C range		
SEChan <i>Constant</i>	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 <i>Constant</i>	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 for the measurement and switched off between Measurements.	
	VX2	2		
	VX3	3		
VX4	4			
MeasPEx <i>Constant</i>	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 <i>Constant</i>	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 <i>Constant</i>	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 <i>Constant</i>	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 <i>Constant</i>	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	All	250 μS	450 μS (default)
	0	All	_50Hz, _60 Hz	3 mS (default)
	≥100	All	All	μS entered

Parameter & Data Type	Enter	
Integ <i>Constant</i>	The time spent on integration in microseconds for each of the channels measured.	
	Entry	Integration
	250 _60Hz or 16667 _50 Hz or 20000	250 μS 16,667 μS (reject 60 Hz noise) 20,000 μS (reject 50 Hz noise)
Mult, Offset <i>Constant, Variable, Array, or Expression</i>	A multiplier and offset by which to scale the raw results of the measurement.	

7.4 Full Bridges

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

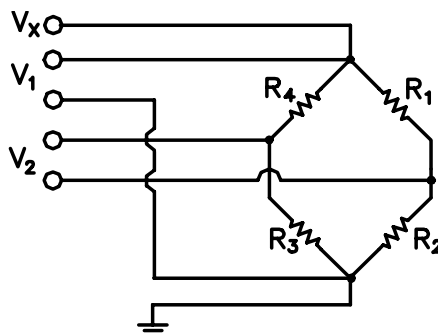


X = result w/mult = 1, offset = 0

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

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 <i>Variable or Array</i>	The Variable in which to store the results of the instruction. When Repts are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Repts.		
Reps <i>Constant</i>	The number of repetitions for the measurement or instruction.		
Range <i>Constant</i>	Alpha Code	Voltage Range	
	mV5000 mV1000 mV200 mV50 mV20 Autorange mV200C mV50C mV20C AutorangeC	± 5000 mV ± 1000 mV ± 200 mV ± 50 mV ± 20 mV mV20 – mV5000 ± 200 mV ± 50 mV ± 20 mV mV20 – mV200	Selects range (Sect. 3.1) The mV200C, mV50C, and mV20C ranges pull the channel into common mode range and check for open input Selects C range
DiffChan <i>Constant</i>	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 repts occur on that channel.		
ExChan <i>Constant</i>	Enter the excitation channel number to excite the first measurement.		
	Alpha Code	Code/ Channel	Result
	VX1 VX2 VX3 VX4	1 2 3 4	Switched excitation channels, are switched to the excitation voltage for the measurement and switched off between measurements
MeasPEx <i>Constant</i>	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 Repts.		
ExmV <i>Constant</i>	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 <i>Constant</i>	Code	Value	Result (Reversing requires twice as much time to complete)
	False True	0 ≠0	Excite only with the excitation voltage entered A second measurement is made with the voltage polarity reversed to cancel offsets
RevDiff <i>Constant</i>	Code	Value	Result (Reversing requires twice as much time to complete)
	False True	0 ≠0	Signal is measured with the high side referenced to the low A second measurement is made after reversing the inputs to cancel offsets

Parameter & Data Type	Enter			
SettlingTime <i>Constant</i>	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	All	250 μ S	450 μ S (default)
	0	All	_50Hz, _60 Hz	3 mS (default)
	>=100	All	All	μ S entered
Integ <i>Constant</i>	The time spent on integration in microseconds for each of the channels measured.			
	Entry	Integration		
	250	250 μ S		
	_60Hz or 16667	16,667 μ S (reject 60 Hz noise)		
	_50 Hz or 20000	20,000 μ S (reject 50 Hz noise)		
Mult, Offset <i>Constant, Variable, Array, or Expression</i>	A multiplier and offset by which to scale the raw results of the measurement.			

7.5 Excitation

ExciteCAO (CAOChannel, ExmV, FastUpdate)

The CR3000 has two continuous analog out (CAO) channels. These channels are used to provide a continuous 5 volt supply to external peripherals. The CAO channels can source 15 mA for continuous voltage and sink 15 mA. The analog output degrades approximately 25 μ V every second. This requires the instruction to be periodically repeated to maintain a given output.

Parameter & Data Type	Enter		
CAOChannel <i>Constant</i>	Enter the CAO channel that should be set high		
	Alpha Code	Code/ Channel	Result
	1	CAO1	Set CAO channel 1 high
	2	CAO2	Set CAO channel 2 high
ExmV <i>Constant</i>	The excitation voltage, in millivolts, to apply to the CAO Channel. The allowable range is \pm 5000 mV.		
FastUpdate <i>Constant Variable or Expression</i>	Used to set the amount of time measurement is delayed after the CAO port is set high. When this parameter is set to True, measurement will be delayed approximately 300 msec. If the parameter is set to False, the measurement will be delayed 2.6 msec.		

Excitel (IxChan, IxuA, Delay)

This instruction sets the specified switched current excitation channel to the current specified.

Parameter & Data Type	Enter												
IxChan <i>Constant</i>	Enter the current excitation channel to use for the first measurement. If the Repts parameter is greater than 1, the excitation channel used for subsequent measurements will be incremented with each measurement based on the MeasPEX parameter.												
	<table border="1"> <thead> <tr> <th>Alpha Code</th> <th>Code/ Channel</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>IX1</td> <td>7</td> <td>Current excitation channel 1.</td> </tr> <tr> <td>IX2</td> <td>8</td> <td>Current excitation channel 2.</td> </tr> <tr> <td>IX3</td> <td>9</td> <td>Current excitation channel 3.</td> </tr> </tbody> </table>	Alpha Code	Code/ Channel	Result	IX1	7	Current excitation channel 1.	IX2	8	Current excitation channel 2.	IX3	9	Current excitation channel 3.
Alpha Code	Code/ Channel	Result											
IX1	7	Current excitation channel 1.											
IX2	8	Current excitation channel 2.											
IX3	9	Current excitation channel 3.											
Ixu <i>Constant</i>	The current excitation, in microAmps, to apply to the excitation channel. The allowable range is ± 2500 microAmps.												
Delay <i>Constant Variable or Expression</i>	Specifies the length of time the current excitation channel is enabled, after which, the channel is set low and the datalogger moves on to the next instruction. If the Delay is set to 0, the current channel will be enabled and the current will be held until the end of the program scan or until another instruction sets a voltage, CAO, or current channel high.												

NOTE

This instruction must NOT be placed inside a conditional statement.

ExciteV (ExChan, ExmV, XDelay)

This instruction sets the specified switched voltage excitation channel to the voltage specified.

Parameter & Data Type	Enter												
ExChan <i>Constant</i>	Enter the excitation channel number to excite the first measurement.												
	<table border="1"> <thead> <tr> <th>Alpha Code</th> <th>Code/ Channel</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>VX1</td> <td>1</td> <td rowspan="4">Switched excitation channels, are switched to the excitation voltage for the measurement and switched off between measurements</td> </tr> <tr> <td>VX2</td> <td>2</td> </tr> <tr> <td>VX3</td> <td>3</td> </tr> <tr> <td>VX4</td> <td>4</td> </tr> </tbody> </table>	Alpha Code	Code/ Channel	Result	VX1	1	Switched excitation channels, are switched to the excitation voltage for the measurement and switched off between measurements	VX2	2	VX3	3	VX4	4
Alpha Code	Code/ Channel	Result											
VX1	1	Switched excitation channels, are switched to the excitation voltage for the measurement and switched off between measurements											
VX2	2												
VX3	3												
VX4	4												
ExmV <i>Constant</i>	The excitation voltage in millivolts. Allowable range ± 5000 mV.												
XDelay <i>Constant Variable or Expression</i>	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.												

SW12 (SW12Channel, State)

The SW12 instruction is used to set a Switched 12-volt supply high or low.

The datalogger has two switched 12-volt outputs, each 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

Each SW-12 supply is unregulated and can supply up to 900 mA at 20C, up to 630 mA at 50C, and up to 360 mA at 85C. A resettable polymeric fuse protects against over-current. Reset is accomplished by removing the load or turning off the SW-12 for several seconds.

Parameter & Data Type	Enter		
SW12Channel			
<i>Constant</i>	Alpha Code	Code/ Channel	Result
	SW12-1	1	Switched 12V channel 1 set according to State parameter
	SW12-2	2	Switched 12V channel 2 set according to State parameter
State	Value	Result	
<i>Constant</i>	0	The SW-12 ports are switched off (0V)	
<i>Variable or Expression</i>	≠0	The SW-12 ports are switched On (Connected to supply voltage)	

7.6 Self Measurements

Battery (Dest)

This instruction reads the battery voltage and stores it in the destination variable. The units for battery voltage are volts.

Calibrate (Dest, Range)

The Calibrate instruction places the CR3000 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). Calibration is typically performed to compensate for errors in voltage measurements due to temperature.

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.									
Range Constant	Option to calibrate ranges not being used. Dest must be entered before Range parameter									
	<table border="1"> <thead> <tr> <th>Alpha Code</th> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>False</td> <td>=0</td> <td>Calibrate only Voltage ranges used in program</td> </tr> <tr> <td>True</td> <td>≠0</td> <td>Calibrate all Voltage ranges</td> </tr> </tbody> </table>	Alpha Code	Value	Result	False	=0	Calibrate only Voltage ranges used in program	True	≠0	Calibrate all Voltage ranges
	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 CR3000 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 Range option is selected, the calibrate instruction only measures the range and integration combinations that occur in the measurements in the program. For the 250 μ s and zero integration calibrations multiple measurements are averaged for the calibration values. The 250 μ s 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 μ s 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 μ Sec integrate 5000 mV single ended offset
17	250 μ Sec integrate 5000 mV differential offset
18	250 μ Sec integrate 5000 mV gain
19	250 μ Sec integrate 1000 mV single ended offset
20	250 μ Sec integrate 1000 mV differential offset
21	250 μ Sec integrate 1000 mV gain
22	250 μ Sec integrate 200 mV single ended offset
23	250 μ Sec integrate 200 mV differential offset
24	250 μ Sec integrate 200 mV gain
25	250 μ Sec integrate 50 mV single ended offset
26	250 μ Sec integrate 50 mV differential offset

27	250 μ Sec integrate 50 mV gain
28	250 μ Sec integrate 20 mV single ended offset
29	250 μ Sec integrate 20 mV differential offset
30	250 μ Sec integrate 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

InstructionTimes (Dest)

The InstructionTimes instruction returns the execution time of each instruction in the program.

The InstructionTimes instruction loads the Dest array with execution times for each instruction in the program (in microseconds). InstructionTimes must appear before the BeginProg statement in the program.

Each element in the array corresponds to a line number in the program. To accommodate all of the instructions in the program, the array must be dimensioned to a number greater than or equal to the total number of lines in the program, including blank lines and comments. The Dest array must also be dimensioned as a long integer (e.g., Public Array(20) AS LONG).

Note that the execution time for an instruction may vary. For instance, it will take longer to execute instructions when the datalogger is communicating with another device.

PanelTemp (Dest, Integ)

This instruction measures the datalogger’s panel temperature, giving the result in °C.

Parameter & Data Type	Enter	
Dest Variable	The Variable in which to store the results of the instruction.	
Integ Constant	The time spent on integration in microseconds for each of the channels measured.	
	Entry	Integration
	250	250 μS
	_60Hz or 16667	16,667 μS (reject 60 Hz noise)
_50 Hz or 20000	20,000 μS (reject 50 Hz noise)	

NOTE This instruction must NOT be placed inside a conditional statement when running in pipeline mode.

7.7 Digital I/O

CheckPort (Port)

CheckPort is a function that returns the status of a control port. CheckPort returns True (-1) if the specified control port is high or False (0) if the control port is low. CheckPort can be used on the right side of an expression (e.g., Variable = CheckPort (Port)) or as an expression.

CheckPort has only one parameter, Port, the number of the port (1-8) to check.

CAUTION The value returned may not be valid if using the control port as a serial port or as a pulse counting port.

PeriodAvg (Dest, Reps, Range, SEChan, Threshold, Option, 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 92 ns, making the resolution of the period measurement 92 ns divided by the number of cycles chosen.

Parameter & Data Type	Enter																																
Dest <i>Variable or Array</i>	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 <i>Constant</i>	The number of repetitions for the instruction on adjacent channels.																																
Range <i>Constant</i>	<p>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.</p> <table border="1"> <thead> <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 Frequency²</th> </tr> <tr> <th>Min</th> <th>Max</th> </tr> </thead> <tbody> <tr> <td>mV1000</td> <td>1</td> <td>200 mV</td> <td>10.0 V</td> <td>2.5 μs</td> <td>200 kHz</td> </tr> <tr> <td>mV200</td> <td>5</td> <td>20 mV</td> <td>2.0 V</td> <td>5 μs</td> <td>100 kHz</td> </tr> <tr> <td>mV50</td> <td>20</td> <td>5 mV</td> <td>2.0 V</td> <td>10 μs</td> <td>50 kHz</td> </tr> <tr> <td>mV20</td> <td>50</td> <td>2 mV</td> <td>2.0 V</td> <td>25 μs</td> <td>20 kHz</td> </tr> </tbody> </table> <p>¹ Signals must cross threshold to trigger the voltage comparator. ² Maximum frequency equals 1/(Twice Minimum Pulse Width) for 50% duty cycle signals.</p>	Range Code	Gain	Signal (pk-pk) ¹		Minimum Pulse Width	Maximum Frequency ²	Min	Max	mV1000	1	200 mV	10.0 V	2.5 μ s	200 kHz	mV200	5	20 mV	2.0 V	5 μ s	100 kHz	mV50	20	5 mV	2.0 V	10 μ s	50 kHz	mV20	50	2 mV	2.0 V	25 μ s	20 kHz
Range Code	Gain			Signal (pk-pk) ¹				Minimum Pulse Width	Maximum Frequency ²																								
		Min	Max																														
mV1000	1	200 mV	10.0 V	2.5 μ s	200 kHz																												
mV200	5	20 mV	2.0 V	5 μ s	100 kHz																												
mV50	20	5 mV	2.0 V	10 μ s	50 kHz																												
mV20	50	2 mV	2.0 V	25 μ s	20 kHz																												
SEChan <i>Constant</i>	The single-ended channel number on which to make the first measurement.																																
Threshold <i>Constant</i>	The voltage in millivolts that the input must cross for a count to occur. For a signal centered around CR3000 ground (Figure 7.7-1) 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.																																
Option	<p>Specifies whether to output the period in μs or the frequency in Hz.</p> <table border="1"> <thead> <tr> <th>Numeric Code</th> <th>Voltage Range</th> </tr> </thead> <tbody> <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> </tbody> </table>	Numeric Code	Voltage Range	0	Period of the signal is returned	1	Frequency of the signal is returned																										
Numeric Code	Voltage Range																																
0	Period of the signal is returned																																
1	Frequency of the signal is returned																																
Cycles <i>Constant</i>	The number of cycles to be measured for the average calculation.																																
Timeout <i>Constant</i>	The maximum time duration (in msec) that the logger will wait for the number of Cycles to be measured for the average calculation.																																
Mult, Offset <i>Constant, Variable, Array, or Expression</i>	A multiplier and offset by which to scale the raw results of the measurement.																																

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 7.7-1, is recommended to center the signal at CR3000 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 7.7-1, 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 ± 50 mV range is 1 mV; consequently, 1 mV of noise on the input signal could cause extraneous counts. **For best results, select the largest input range (smallest gain) that will meet the minimum input signal requirements.**

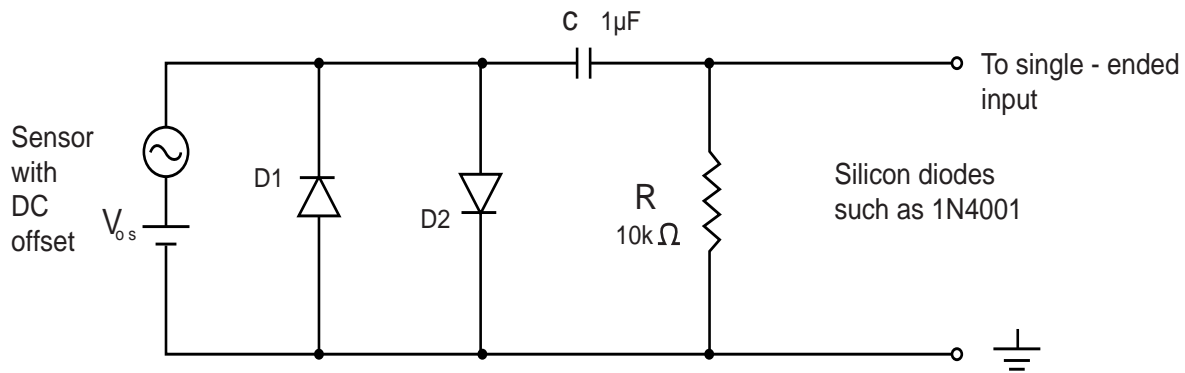


FIGURE 7.7-1. Input conditioning circuit for low-level and high level period averaging.

Figure 7.7-1 shows a circuit that capacitively couples an input signal to center 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 ($X_c = (2 \cdot \pi \cdot f \cdot C)^{-1}$) and resistor R1 form a voltage divider at low frequencies ($R1 / (R1 + X_c)$) 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 \neq ground terminals. Single-ended offset voltages of up to 2 $\mu\text{V}/\text{mA}$ of current that flows into the \neq ground terminals can be induced across the front panel. The back-to-back diodes can be tied into the G ground terminals, rather than \neq ground terminals, if this is a problem.

PortsConfig (Mask, Function)

The PortsConfig instruction is used to configure one or more control ports as either input or output.

By default, ports are configured as input. The PortsConfig instruction may be needed if a port is configured as output by a WriteIO or PortSet instruction and then subsequently needs to function as an input.

Parameter & Data Type	Enter
Mask <i>Binary Value (preceded by &B) or Integer (0-255)</i>	The Mask parameter is used to select which ports will be affected by this instruction. It is a binary representation of the ports (reading from left to right, the ports are represented as 8, 7, 6...1). If a port position is set to 1, the datalogger configures that port. Binary numbers are entered by preceding the number with "&B". Leading zeros can be omitted. As an example, if &B110 is entered for this parameter, ports 3 and 2 will be configured, based on the Function parameter.
Function <i>Binary Value (preceded by &B) or Integer (0-255)</i>	The function parameter is used to configure the port. A binary value is entered to set each port location. 0 configures the port for input; a 1 configures the port for output. Using the above example mask, if the Function parameter is set to &B110, ports 3 and 2 will be configured for output (port 1 uses the code for input, but it is not affected because of the mask).

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.

Parameter & Data Type	Enter
Dest <i>Variable</i>	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 <i>Constant</i>	The control port number (1-8) for which the status should be obtained.

PortSet (Port, State)

This Instruction will set the specified digital I/O port or switched 12V channel high or low.

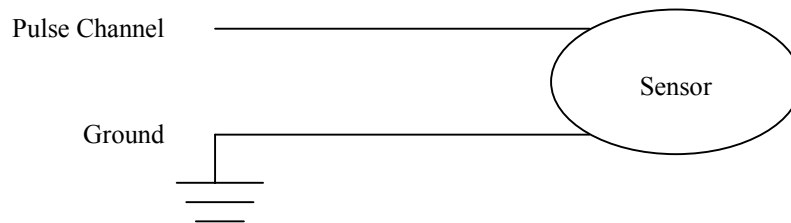
Parameter & Data Type	Enter		
Port <i>Constant, Variable, or Expression</i>	The number of the port or switched 12V channel to set with the instruction		
	Code	Description	
	1	Control Port 1	
	2	Control Port 2	
	3	Control Port 3	
	4	Control Port 4	
	5	Control Port 5	
	6	Control Port 6	
	7	Control Port 7	
	8	Control Port 8	
	9	Switched 12 V channel 1 (SW12V-1)	
10	Switched 12 V channel 2 (SW12V-2)		
State <i>Constant, Variable, or Expression</i>	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)

Makes a measurement on a pulse channel.

With the PulseCount instruction, a pulse channel can be configured as a high frequency input, a low level AC input, or as a switch closure input. Control ports can be configured as high frequency input or switch closure.

PulseCountReset is used to reset both the pulse counter and the running average values in the pulse count instruction. The 24-bit counter can count up to 16,777,215. Each scan of the datalogger causes the counter to output the number of pulses accumulated since the last scan. If the scans stop, as in a program with more than one scan loop, the counter continues to accumulate counts until an overrange occurs or a scan is initiated.



Parameter & Data Type	Enter																										
Dest <i>Variable or Array</i>	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 <i>Constant</i>	The number of repetitions for the measurement or instruction. Measurements are made on consecutive channels.																										
PChan <i>Constant</i>	The number of the pulse channel for the measurement. <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <tr><td>1</td><td>Pulse Channel 1</td></tr> <tr><td>2</td><td>Pulse Channel 2</td></tr> <tr><td>3</td><td>Pulse Channel 3</td></tr> <tr><td>4</td><td>Pulse Channel 4</td></tr> <tr><td>11</td><td>Control Port 1</td></tr> <tr><td>12</td><td>Control Port 2</td></tr> <tr><td>13</td><td>Control Port 3</td></tr> <tr><td>14</td><td>Control Port 4</td></tr> <tr><td>15</td><td>Control Port 5</td></tr> <tr><td>16</td><td>Control Port 6</td></tr> <tr><td>17</td><td>Control Port 7</td></tr> <tr><td>18</td><td>Control Port 8</td></tr> </tbody> </table>	Code	Description	1	Pulse Channel 1	2	Pulse Channel 2	3	Pulse Channel 3	4	Pulse Channel 4	11	Control Port 1	12	Control Port 2	13	Control Port 3	14	Control Port 4	15	Control Port 5	16	Control Port 6	17	Control Port 7	18	Control Port 8
Code	Description																										
1	Pulse Channel 1																										
2	Pulse Channel 2																										
3	Pulse Channel 3																										
4	Pulse Channel 4																										
11	Control Port 1																										
12	Control Port 2																										
13	Control Port 3																										
14	Control Port 4																										
15	Control Port 5																										
16	Control Port 6																										
17	Control Port 7																										
18	Control Port 8																										
PConfig <i>Constant</i>	A code specifying the type of pulse input to measure. Control ports can be used to measure high frequency or switch closure configurations, but not low level AC. <table border="1"> <thead> <tr> <th>Code</th> <th>Input Configuration</th> </tr> </thead> <tbody> <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> </tbody> </table>	Code	Input Configuration	0	High Frequency	1	Low Level AC	2	Switch Closure																		
Code	Input Configuration																										
0	High Frequency																										
1	Low Level AC																										
2	Switch Closure																										
POption <i>Constant</i>	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 border="1"> <thead> <tr> <th>Code</th> <th>Result</th> </tr> </thead> <tbody> <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> </tbody> </table>	Code	Result	0	Counts	1	Frequency (Hz) counts/scan interval in seconds	>1	Running average of frequency. The number entered is the time period over which the frequency is averaged in milliseconds.																		
Code	Result																										
0	Counts																										
1	Frequency (Hz) counts/scan interval in seconds																										
>1	Running average of frequency. The number entered is the time period over which the frequency is averaged in milliseconds.																										
Mult, Offset <i>Constant, Variable, Array, or Expression</i>	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.																										

NOTE

The PulseCount instruction can not be used in a Slow Sequence scan and must NOT be placed inside a conditional statement when running in pipeline mode.

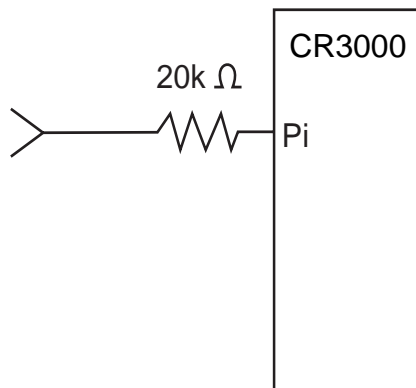


FIGURE 7.7-2. Conditioning Large Voltage Pulses

The maximum input voltage on a pulse channel is ± 20 V. Refer to Figure 7.7-2 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: 250 kHz
(50% Duty Cycle)
Lower Threshold: 0.9 V*
Upper Threshold: 2.2 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 250 kHz will be counted if centered around +2.5 V with deviations $\geq \pm 2.5$ V for ≥ 1.2 microseconds.

- Control Ports
A high frequency sensor is connected between C1..C8 and 5V.

Low Level AC (Pulse Channels Only)

Input Hysteresis: 16 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 20 Hz
200 mV	0.5 Hz to 200 Hz
2000 mV	0.3 Hz to 10 kHz
5000 mV	0.3 Hz to 20 kHz

Switch Closure

- **Pulse Channels**
A switch closure is connected between P1..P4 and analog ground. When the switch is open, the CR3000 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

- **Control Ports Channels**
A switch closure is connected between C1..C8 and 5V.

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 CR3000 reads the counts accumulated since the last scan and then resets the counter. If the scans stop, as in a program with more than 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.

PulsePort (Port, Delay)

This instruction toggles the state of a port, delays the specified amount of time, toggles the port, and then delays a second time. The second delay in the instruction allows it to be used to create a 50 percent duty cycle clock. The instruction has the following parameters:

Parameter & Data Type	Enter
Port <i>Constant</i>	The control port that should be set by the instruction. The number of the port (1-8) is entered:
Delay <i>Array</i>	The amount of time, in microseconds, that the instruction should delay after each toggle of the port

ReadIO (Dest, Mask)

ReadIO is used to read the status of selected control I/O channels (ports) on the CR3000 and store the results. 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
```

SDI12Recorder (Dest, SDIPort, SDIAddress, SDICommand, Multiplier, Offset)

The SDI12Recorder instruction is used to retrieve the results from an SDI-12 sensor.

Each execution of the SDI12Recorder instruction sends an (address)M! and then an (address)D0!

M! instructs the sensor to make the measurement; D0! is a request for the data.

Parameter & Data Type	Enter																		
Dest <i>Variable or Array</i>	The Dest parameter is a variable in which to store the results of the measurement. Dest must have enough elements to store all the data that is returned by the SDI-12 sensor or a 'variable out of range' error will result during the execution of the instruction.																		
SDIPort <i>Constant</i>	The SDIPort parameter is the port to which the SDI-12 sensor is connected. A numeric value is entered: <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Control Port 1</td> </tr> <tr> <td>3</td> <td>Control Port 3</td> </tr> <tr> <td>5</td> <td>Control Port 5</td> </tr> <tr> <td>7</td> <td>Control Port 7</td> </tr> </tbody> </table>	Code	Description	1	Control Port 1	3	Control Port 3	5	Control Port 5	7	Control Port 7								
Code	Description																		
1	Control Port 1																		
3	Control Port 3																		
5	Control Port 5																		
7	Control Port 7																		
SDIAddress <i>Constant or Variable</i>	The address of the SDI12 sensor that will be affected by this instruction. Valid addresses are 0 through 9, A through Z, and a through z. Alphabetical characters should be enclosed in quotes (e.g., "A").																		
SDICommand <i>String</i>	The command strings that will be sent to the sensor. The command should be enclosed in quotes. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th>Command</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>?!</td> <td>Address query</td> </tr> <tr> <td>An!</td> <td>Change address (where n = address)</td> </tr> <tr> <td>C!</td> <td>Initiate concurrent measurements</td> </tr> <tr> <td>CC!</td> <td>Initiate concurrent measurements (with checksum)</td> </tr> <tr> <td>I!</td> <td>Send identification (destination variable must be formatted as a string)</td> </tr> <tr> <td>M!</td> <td>Initiate measurements</td> </tr> <tr> <td>MC!</td> <td>Initiate measurements (with checksum)</td> </tr> <tr> <td>M1! - M9!</td> <td>Additional measurement commands specified by the SDI-12 sensor</td> </tr> </tbody> </table>	Command	Description	?!	Address query	An!	Change address (where n = address)	C!	Initiate concurrent measurements	CC!	Initiate concurrent measurements (with checksum)	I!	Send identification (destination variable must be formatted as a string)	M!	Initiate measurements	MC!	Initiate measurements (with checksum)	M1! - M9!	Additional measurement commands specified by the SDI-12 sensor
Command	Description																		
?!	Address query																		
An!	Change address (where n = address)																		
C!	Initiate concurrent measurements																		
CC!	Initiate concurrent measurements (with checksum)																		
I!	Send identification (destination variable must be formatted as a string)																		
M!	Initiate measurements																		
MC!	Initiate measurements (with checksum)																		
M1! - M9!	Additional measurement commands specified by the SDI-12 sensor																		

	<p>RC! Continuous measurement (with checksum) R0! - R9! Continuous measurement commands V! Initiate verify sequence X! Extended commands (destination variable must be formatted as a string)</p> <p>Values from the public table can be sent to the sensor using the following command:</p> <p>"XD%variable_name%!"</p> <p>Multiple values from an array can be sent by preceding the array name with the number of values to send (e.g., "XD%3variable%!" would send Variable(1), Variable(2), and Variable(3)).</p> <p>If a check summed command fails, a NAN will be returned and the command will be retried.</p>
Multiplier, Offset	The Multiplier and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement.

VibratingWire (Dest, Reps, Range, SEChan, ExChan, StartFreq, EndFreq, TSweep, Steps, DelMeas, NumCycles, DelReps, Multiplier, Offset)

The VibratingWire instruction is used to measure a vibrating wire sensor with a swept frequency (from low to high).

The period of the response is measured and $1/T^2$ is calculated. T is the period of the measured signal in milliseconds.

Parameter & Data Type	Enter										
Dest	The Dest parameter is a variable in which to store the results of the measurement.										
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.										
Range	The Range parameter is the voltage range for the measurement. An alphanumeric or the numeric code can be entered:										
	<table border="1"> <thead> <tr> <th>Alphanumeric</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>mV1000</td> <td>+1000 mV</td> </tr> <tr> <td>mV200</td> <td>+200 mV</td> </tr> <tr> <td>mV50</td> <td>+50 mV</td> </tr> <tr> <td>mV20</td> <td>+20 mV</td> </tr> </tbody> </table>	Alphanumeric	Description	mV1000	+1000 mV	mV200	+200 mV	mV50	+50 mV	mV20	+20 mV
	Alphanumeric	Description									
	mV1000	+1000 mV									
	mV200	+200 mV									
	mV50	+50 mV									
mV20	+20 mV										

Parameter & Data Type	Enter															
SEChan	The SEChan argument is the number of the single-ended channel on which to make the first measurement. If the Reps parameter is greater than 1, the additional measurements will be made on sequential channels. If the SEChan number is entered as a negative value, all Reps will be performed on the same channel.															
ExChan	The ExChan argument is used to specify the excitation channel to use for the first measurement. If the Reps parameter is greater than 1, the excitation channel used for subsequent measurements will be incremented with each measurement. An alphanumeric or numeric code can be entered: <table border="1" data-bbox="527 569 1122 730"> <thead> <tr> <th>Alphanumeric</th> <th>Numeric</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>VX1</td> <td>1</td> <td>Excitation channel 1</td> </tr> <tr> <td>VX2</td> <td>2</td> <td>Excitation channel 2</td> </tr> <tr> <td>VX3</td> <td>3</td> <td>Excitation channel 3</td> </tr> <tr> <td>VX4</td> <td>4</td> <td>Excitation channel 4</td> </tr> </tbody> </table>	Alphanumeric	Numeric	Description	VX1	1	Excitation channel 1	VX2	2	Excitation channel 2	VX3	3	Excitation channel 3	VX4	4	Excitation channel 4
Alphanumeric	Numeric	Description														
VX1	1	Excitation channel 1														
VX2	2	Excitation channel 2														
VX3	3	Excitation channel 3														
VX4	4	Excitation channel 4														
StartFreq	The StartFreq parameter is the frequency, in Hertz, at which to start the excitation. StartFreq must be greater than 20 Hz.															
EndFreq	The EndFreq parameter is the frequency, in Hertz, at which to end the excitation. EndFreq must be less than 5000 Hz.															
TSweep	TSweep is the duration, in milliseconds, of the frequency sweep.															
Steps	Steps is the number of step changes in frequency to sweep from the start frequency (StartFreq) to the end frequency (EndFreq). If -1 is entered for the Steps parameter, the datalogger will calculate and use the maximum number of Steps allowed for the selected StartFreq, StopFreq, and TSweep parameter values. This results in the finest possible resolution in Step size.															
DelMeas	The DelMeas parameter is used to specify the number of microseconds to delay before measuring the return signal after the frequency sweep is completed.															
NumCycles	NumCycles is the number of cycles to measure.															
DelReps	The DelReps parameter is used to specify the number of microseconds to delay between each measurement repetition.															
Mult, Offset	The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement. With a multiplier (mult) of 1 and an offset of 0, the output is $1/T^2$, where T is the period of the measured signal in milliseconds.															

For additional information on the sweep frequency operation for the VibratingWire instruction, refer to Sweep Frequency Operation in the CRBasic Help file.

WriteIO (Mask, Source)

WriteIO is used to set the status of selected control I/O channels (ports) on the CR3000. (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)	<i>' Set port 3 high.</i>
-------------------------------	---------------------------

Parameter & Data Type	Enter
Mask <i>Constant</i>	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 <i>Constant Variable</i>	The Variable or number that is to be written to the I/O ports.

7.8 Specific Sensors

CS110 (Dest, Leakage, Status, Integ, Mult, Offset)

The CS110 instruction is used to measure an electric field by means of a CS110 electric field meter.

See the CRBasic Editor or the CS110 manual for details on the instruction.

CS616 (Dest, Reps, SEChan, Port, MeasPerPort, Mult, Offset)

The CS616 instruction is used to enable and measure a CS616 water content reflectometer. This instruction outputs a period measurement in microseconds.

Parameter & Data Type	Enter
Dest	The Dest parameter is the variable or variable array in which to store the results of the measurement. Dest must be dimensioned to at least the number of Reps.
Reps	The Reps parameter is the number of measurements that should be made using this instruction. If Reps is greater than 1, Dest must be an array dimensioned to the size of Reps.
SEChan	The SEChan parameter is the number of the single-ended channel on which to make the first measurement. If the Reps parameter is greater than 1, the additional measurements will be made on sequential channels.
Port	The Port parameter is the control port (1-8) that will be used to enable the CS616 sensor.

Parameter & Data Type	Enter
MeasPerPort	The MeasPerPort parameter is the number of control ports to be used to control the CS616 sensor(s). If Repts is set to 4, MeasPerPort = 4 will result in the same port being used for all measurements. MeasPerPort = 1 will result in four sequential ports being used for the measurements. MeasPerPort = 2 will result in one port being used for the first two measurements, and the next port being used for the next two measurements.
Mult, Offset	The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement.

HydraProbe (Dest, SourceVolts, ProbeType, SoilType)

The HydraProbe instruction is used to process the values returned from a Stevens Vitel Hydra Probe sensor. Four single-ended voltage readings from the probe must be stored in an array prior to the HydraProbe instruction.

Parameter & Data Type	Enter
Dest Array	The variable array that will hold the values returned from the Hydra Probe sensor. This variable must be dimensioned to 11. The sensor returns the following measurements: soil type (1 = sand, 2 = silt, 3 = clay), real dielectric constant, imagined dielectric constant, temperature, real dielectric constant with temperature correction, imagined dielectric constant with temperature correction, water content (fraction by volume), salinity (grams of NaCl per liter), soil conductivity (S/m), soil conductivity with temperature correction (S/m), soil water conductivity with temperature correction (S/m).
SourceVolts Array	The variable array that will hold the voltages returned by the sensor (V1, V2, V3, and V4). This variable must be dimensioned to 4.
ProbeType Constant	A code identifying which version of the Hydra Probe is being measured.
	Code Description
	0 Standard Probe (5V output for V4) 1 Probe Type A (2.5 V output for V4)
SoilType Constant	A code to indicate the type of soil
	Code Description
	1 Sand 2 Silt 3 Clay
Multiplier Constant, Variable or Expression	Multiplier that will be applied to the returned temperature value.
Offset Constant, Variable or Expression	Offset that will be applied to the returned temperature value.

SlowAntenna

See CS110 manual.

Therm107 (Dest, Reps, SEChan, ExChan, SettlingTime, Integ, Mult, Offset)**Therm108 (Dest, Reps, SEChan, ExChan, SettlingTime, Integ, Mult, Offset)****Therm109 (Dest, Reps, SEChan, ExChan, SettlingTime, Integ, Mult, Offset)**

The Therm107, Therm108, and Therm109 instructions are used to measure the 107, 108, and 109 thermistors, respectively.

The syntax for all three instructions is the same.

The instruction makes a half bridge voltage measurement and processes the results using the Steinhart-Hart calculation. The output is temperature in degrees C.

Parameter & Data Type	Enter																				
Dest	Dest is the variable in which the results of the measurement will be stored. If the Reps parameter is greater than one, Dest must be an array dimensioned large enough to hold the results for all repetitions.																				
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.																				
SEChan	The SEChan argument is the number of the single-ended channel on which to make the first measurement (1-28). If the Reps parameter is greater than 1, the additional measurements will be made on sequential channels.																				
ExChan	The ExChan is the excitation channel number (1-4) to use to excite the thermistor. If multiple thermistors are measured with one instruction, all repetitions will use the same excitation channel. <table border="1"> <thead> <tr> <th>Alphanumeric</th> <th>Numeric</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>VX1</td> <td>1</td> <td>Excitation channel 1</td> </tr> <tr> <td>VX2</td> <td>2</td> <td>Excitation channel 2</td> </tr> <tr> <td>VX3</td> <td>3</td> <td>Excitation channel 3</td> </tr> <tr> <td>VX4</td> <td>4</td> <td>Excitation channel 4</td> </tr> </tbody> </table>	Alphanumeric	Numeric	Description	VX1	1	Excitation channel 1	VX2	2	Excitation channel 2	VX3	3	Excitation channel 3	VX4	4	Excitation channel 4					
Alphanumeric	Numeric	Description																			
VX1	1	Excitation channel 1																			
VX2	2	Excitation channel 2																			
VX3	3	Excitation channel 3																			
VX4	4	Excitation channel 4																			
SettlingTime	The SettlingTime parameter is the amount of time to delay after setting up a measurement and before making the measurement. Refer to the table below for default SettlingTimes. <table border="1"> <thead> <tr> <th>Entry</th> <th>Range</th> <th>Integration</th> <th>Settling Time</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>All</td> <td>250 ms</td> <td>450 ms (default)</td> </tr> <tr> <td>0</td> <td>All</td> <td>_50Hz</td> <td>3 ms (default)</td> </tr> <tr> <td>0</td> <td>All</td> <td>_60Hz</td> <td>3 ms (default)</td> </tr> <tr> <td>>100</td> <td>All</td> <td>All</td> <td>ms entered</td> </tr> </tbody> </table>	Entry	Range	Integration	Settling Time	0	All	250 ms	450 ms (default)	0	All	_50Hz	3 ms (default)	0	All	_60Hz	3 ms (default)	>100	All	All	ms entered
Entry	Range	Integration	Settling Time																		
0	All	250 ms	450 ms (default)																		
0	All	_50Hz	3 ms (default)																		
0	All	_60Hz	3 ms (default)																		
>100	All	All	ms entered																		
Integ	The Integ parameter is the amount of time, in microseconds, to integrate a signal for the channel being measured. <table border="1"> <thead> <tr> <th>Option</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>250</td> <td>Performs a 250 microsecond integration.</td> </tr> <tr> <td>_60Hz</td> <td>Performs a 16.667 millisecond integration; filters 60 Hz noise.</td> </tr> <tr> <td>_50Hz</td> <td>Performs a 20 millisecond integration; filters 50 Hz noise.</td> </tr> </tbody> </table>	Option	Description	250	Performs a 250 microsecond integration.	_60Hz	Performs a 16.667 millisecond integration; filters 60 Hz noise.	_50Hz	Performs a 20 millisecond integration; filters 50 Hz noise.												
Option	Description																				
250	Performs a 250 microsecond integration.																				
_60Hz	Performs a 16.667 millisecond integration; filters 60 Hz noise.																				
_50Hz	Performs a 20 millisecond integration; filters 50 Hz noise.																				
Mult, Offset	The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement. With a multiplier (mult) of 1 and an offset of 0, the output is temperature in degrees C. With a multiplier of 1.8 and an offset of 32, the output is temperature in degrees F.																				

7.9 Peripheral Devices

AM25T (Dest, Reps, Range, AM25TChan, DiffChan, TCTYPE, Tref, ClkPort, ResPort, ExChan, RevDiff, SettlingTime, Integ, Mult, Offset)

This Instruction controls the AM25T Multiplexer.

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	The number of channels to measure on the AM25T. Enter 0 to just measure reference temperature.		
Range Constant	Alpha Code	Voltage Range	
	mV5000	± 5000 mV	
	mV1000	± 1000 mV	
	mV200	± 200 mV	
	mV50	± 50 mV	
	mV20	± 20 mV	
	Autorange	mV2_5 – mV5000	Selects range (Sect. 3.1)
	mV200C	± 200 mV	The mV200C, mV50C, and mV20C
	mV50C	± 50 mV	ranges pull the channel into common mode
	mV20C	± 20 mV	range and check for open circuit
AutorangeC	mV2_5 – mV250	Selects C range	
Am25TChan Constant	The starting input channel on the multiplexer.		
DiffChan Constant	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.		
TCTYPE Constant	The Thermocouple Type Code.		
	Alpha Code	Numeric	Type
	mV	-1	Outputs a voltage, in millivolts
	TypeT	0	Copper Constantan
	TypeE	1	Chromel Constantan
	TypeK	2	Chromel Alumel
	TypeJ	3	Iron Constantan
	TypeB	4	Platinum Rhodium
TypeR	5	Platinum Rhodium	
TypeS	6	Platinum Rhodium	
Tref Variable	The variable in which to store/read the AM25T reference temperature.		
ClkPort Constant	The Digital Output port number that will be used to clock the AM25T. One clock port may be used with several AM25Ts.		
ResPort Constant	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		
ExChan Constant	The Excitation Channel number that will be used to provide excitation for the PRT reference temperature measurement.		
	Alpha Code	Numeric	Description
	0	0	Temperature not measured
	VX1	1	Excitation channel 1
	VX2	2	Excitation channel 2
	VX3	3	Excitation channel 3
VX4	4	Excitation channel 4	

Parameter & Data Type	Enter		
RevDiff <i>Constant</i>	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 <i>Constant</i>	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
	0	All	250 μS
	0	All	_50Hz, _60 Hz
	>=100	All	All
			450 μS (default)
			3 mS (default)
			μS entered
Integ <i>Constant</i>	The time spent on integration in microseconds for each of the channels measured.		
	Entry	Integration	
	250	250 μS	
	_60Hz or 16667	16,667 μS (reject 60 Hz noise)	
	_50 Hz or 20000	20,000 μS (reject 50 Hz noise)	
Mult, Offset <i>Constant, Variable, Array, or Expression</i>	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.		

CS7500 (Dest, Reps, SDMAAddress, CS7500Cmd)

Communicates with the CS7500 open path CO₂ and H₂O sensor. See CS7500 manual for more information.

Parameter & Data Type	Enter	
Dest <i>Variable or Array</i>	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
6	4	
Reps <i>Constant or Expression</i>	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	
SDMAAddress <i>Constant</i>	The SDMAAddress 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 SDMAAddress 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:														
	<table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Get CO2 & H2O molar density (mmol/m3)</td> </tr> <tr> <td>1</td> <td>Get CO2 & H2O absorptance</td> </tr> <tr> <td>2</td> <td>Get internal pressure estimate (kPa), auxiliary measurement A, auxiliary measurement B, and cooler voltage (V)</td> </tr> <tr> <td>3</td> <td>Get cell diagnostic value, output bandwidth (Hz), and programmed delay [230 + (delay * 6.579)] (msec)</td> </tr> <tr> <td>4</td> <td>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))</td> </tr> <tr> <td>5</td> <td>Get CO2 & H2O molar density (mmol/m3) and internal pressure estimate (kPa)</td> </tr> </tbody> </table>	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)
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, SDMAAddress, CSAT3Cmd, CSAT3Opt)

Communicates with the CSAT3 three-dimensional sonic anemometer. See CSAT3 manual for more information.

Parameter & Data Type	Enter
Dest <i>Variable Array</i>	The Dest parameter is a variable array 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 <i>Constant or Expression</i>	The Reps parameter is the number of times the measurement should be made. Measurements are made on consecutive channels.
SDMAAddress <i>Constant</i>	<p>The SDMAAddress 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.</p> <p>The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.</p>

Parameter & Data Type	Enter																										
CSAT3Cmd	<p>Commands 90 - 92 send a measurement trigger to the CSAT3 with the SDMAAddress 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 SDMAAddress parameter without triggering a new CSAT3 measurements. The CSAT() instruction must be preceded by the SDMTrigger() instruction in order to used Options 97 - 99.</p> <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </table>	Code	Description	90	Trigger and Get wind & speed of sound data	91	Trigger and Get wind & sonic temperature data	92	Trigger and Get wind & speed of sound data minus 340 m/s	97	Get wind & speed of sound data minus 340 m/s after a Group Trigger	98	Get wind & sonic temperature data after a Group Trigger	99	Get wind & speed of sound data after a Group Trigger												
Code	Description																										
90	Trigger and Get wind & speed of sound data																										
91	Trigger and Get wind & sonic temperature data																										
92	Trigger and Get wind & speed of sound data minus 340 m/s																										
97	Get wind & speed of sound data minus 340 m/s after a Group Trigger																										
98	Get wind & sonic temperature data after a Group Trigger																										
99	Get wind & speed of sound data after a Group Trigger																										
CSAT3Opt	<p>The CSAT3Opt 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.</p> <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </table>	Code	Description	1	Set Execution Parameter to 1 Hz	2	Set Execution Parameter to 2 Hz	3	Set Execution Parameter to 3 Hz	5	Set Execution Parameter to 5 Hz	6	Set Execution Parameter to 6 Hz	10	Set Execution Parameter to 10 Hz	12	Set Execution Parameter to 12 Hz	20	Set Execution Parameter to 20 Hz	30	Set Execution Parameter to 30 Hz	60	Set Execution Parameter to 60 Hz	61	Set Execution Parameter to 60 Hz to 10 Hz Oversample Mode	62	Set Execution Parameter to 60 Hz to 20 Hz Oversample Mode
Code	Description																										
1	Set Execution Parameter to 1 Hz																										
2	Set Execution Parameter to 2 Hz																										
3	Set Execution Parameter to 3 Hz																										
5	Set Execution Parameter to 5 Hz																										
6	Set Execution Parameter to 6 Hz																										
10	Set Execution Parameter to 10 Hz																										
12	Set Execution Parameter to 12 Hz																										
20	Set Execution Parameter to 20 Hz																										
30	Set Execution Parameter to 30 Hz																										
60	Set Execution Parameter to 60 Hz																										
61	Set Execution Parameter to 60 Hz to 10 Hz Oversample Mode																										
62	Set Execution Parameter to 60 Hz to 20 Hz Oversample Mode																										

SDMAO4 (Source, Reps, SDMAAddress)

This instruction is used to set the voltage to an SDM-AO4 four-channel analog output device.

The SDMAO4 instruction has the following parameters:

Parameter & Data Type	Enter
Source Variable	The Source parameter is the variable that holds the voltage, in millivolts, which 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 Constant or Expression	The Reps parameter determines the number of SDM-AO4 devices to supply a voltage using this instruction. The SDM-AO4s must have sequential SDM addresses if the Reps parameter is greater than 1.

Parameter & Data Type	Enter
SDMAddress <i>Constant</i>	<p>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.</p> <p>The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.</p>

SDMCAN (Dest, SDMAddress, TimeQuanta, TSEG1, TSEG2, ID, DataType, StartBit, NumBits, NumVals, Multiplier, Offset)

The SDMCAN instruction is used to measure and control the SDM-CAN interface.

Multiple SDM 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 <i>Variable or Array</i>	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 <i>Constant</i>	<p>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.</p> <p>The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.</p>

Parameter & Data Type	Enter
TimeQuanta <i>Integer</i>	<p>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:</p> $t_{\text{bit}} = t_q + t_{\text{TSEG1}} + t_{\text{TSEG2}}$ <p>The first time segment, the synchronization segment (S-SG), is defined by the TimeQuanta parameter. To calculate a suitable value for TimeQuanta, use the following equation:</p> $\text{TimeQuanta} = t_q * 8 * 10^6$ <p>where t_q = the TimeQuanta. There are between 8 and 25 time quanta in the bit time. The bit time is defined as 1/ baud rate.</p>
TSEG1 <i>Integer</i>	<p>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:</p> $T_{\text{TSEG1}} = t_{\text{TSEG1}} / t_q$
TSEG2 <i>Integer</i>	<p>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:</p> $T_{\text{TSEG2}} = t_{\text{TSEG2}} / t_q$ <p>The relative values of TSEG1 and TSEG2 determine when the SDM-CAN samples the data bit.</p>
ID	<p>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.</p>
DataType	<p>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:</p>

Parameter & Data Type	Enter	
	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.
	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.

Parameter & Data Type	Enter																																	
	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: <table border="1"> <thead> <tr> <th>Status</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>The SDM-CAN is involved in bus activities; error counters are less than 96.</td> </tr> <tr> <td>0001</td> <td>The SDM-CAN is involved in bus activities; one or more error counters is greater than or equal to 96.</td> </tr> <tr> <td>0002</td> <td>The SDM-CAN is not involved in bus activities; error counters are less than 96.</td> </tr> <tr> <td>0003</td> <td>The SDM-CAN is not involved in bus activities; one or more error counters is greater than or equal to 96.</td> </tr> </tbody> </table>	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.																						
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. <table border="1"> <thead> <tr> <th>Switch</th> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>A</td> <td>0</td> <td>Currently not used; set to 0.</td> </tr> <tr> <td rowspan="2">B</td> <td>0</td> <td>SDM-CAN returns the last value captured from the network, even if that value has been read before (default).</td> </tr> <tr> <td>1</td> <td>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.</td> </tr> <tr> <td rowspan="5">C</td> <td>2-9</td> <td>Currently not used.</td> </tr> <tr> <td>0</td> <td>Disable I/O interrupts (default).</td> </tr> <tr> <td>1</td> <td>Enable I/O interrupts, pulsed mode.</td> </tr> <tr> <td>2</td> <td>Enable I/O interrupts, fast mode.</td> </tr> <tr> <td>3-7</td> <td>Currently not used.</td> </tr> <tr> <td rowspan="3">D</td> <td>8</td> <td>Place the SDM-CAN into low power stand-by mode</td> </tr> <tr> <td>9</td> <td>Leave switch setting unchanged.</td> </tr> <tr> <td>0</td> <td>Listen only (error passive) mode. CAN transmissions are not confirmed.</td> </tr> <tr> <td></td> <td>1</td> <td>Transmit once. Data will not be retransmitted in case of error or loss of arbitration. Frames received without error are acknowledged.</td> </tr> </tbody> </table>	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).	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.	C	2-9	Currently not used.	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.	D	8	Place the SDM-CAN into low power stand-by mode	9	Leave switch setting unchanged.	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.
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).																																
	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.																																
C	2-9	Currently not used.																																
	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.																																
D	8	Place the SDM-CAN into low power stand-by mode																																
	9	Leave switch setting unchanged.																																
	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.																																

Parameter & Data Type	Enter	
		<p>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.</p> <p>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.</p> <p>4 Transmit once; self-test. The SDM-CAN will perform a successful transmission even if there is no acknowledgement from an external CAN node. Frames received correctly from an external node are acknowledged.</p> <p>5 Self-reception; self -test. The SDM-CAN will perform a successful transmission even if there is no acknowledgement from an external CAN node. Frames received correctly from an external node are acknowledged. SDM-CAN will receive its own transmission.</p> <p>6 Normal; self-test. The SDM-CAN will perform a successful transmission even if there is no acknowledgement from an external CAN node. Frames received correctly from an external node are acknowledged.</p> <p>7 Not defined.</p> <p>8 Not defined.</p> <p>9 Leave switch setting unchanged.</p> <p>33 Read SDM-CAN's internal switches. Place results in the array specified in the Dest parameter.</p>
StartBit	<p>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.</p>	

Parameter & Data Type	Enter
NumBits	<p>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).</p> <p>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.</p> <p>Note: This parameter may be overridden by a fixed number of bits, depending upon the data type selected.</p>
NumVals	<p>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.</p>
Mult, Offset	<p>The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement.</p>

NOTE If more than one Canbus Instruction 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 TSEG2=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
  SDMCAN(CANBLK1(), ADDR1, TQUANT, TSEG1, TSEG2, 217056256,
  DTYPE1, STBIT1, NBITS1, NVALS1, CMULT1, COSET1)
  Next Scan
EndProg
    
```

SDMCD16AC (Source, Repls, SDMAAddress)

The SDMCD16AC 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 SDMCD16AC instruction. A non-zero value will turn the port on; a zero value will turn it off. The values to be sent to the SDM-CD16AC are held in the Source array.

Parameter & Data Type	Enter
Source <i>Array</i>	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..
Repls <i>Constant</i>	The Repls parameter is the number of SDM-CD16AC devices to be controlled with this instruction.
SDMAAddress <i>Constant</i>	The SDMAAddress parameter is used to define the address of the SDM-CD16AC that will be controlled with this instruction. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. If the Repls parameter is greater than 1, the datalogger will increment the SDM address for each subsequent device with which it communicates.

SDMINT8 (Dest, SDMAAddress, 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 CR3000. The SDM-INT8 is a (S)ynchronous (D)evice for the (M)asurement of intervals, counts between events, frequencies, periods, and/or time since an event. See the SDM-INT8 manual for more information about its capabilities.

Parameter & Data Type	Enter																				
Dest <i>Variable or Array</i>	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.																				
SDMAddress <i>Constant</i>	the address of the INT8 with which to communicate. Valid SDM addresses are 0 through 14 (factory default is 0). Address 15 is reserved for the SDMTrigger instruction. If the Repts parameter is greater than 1, the datalogger will increment the SDM address for each subsequent device with which it communicates.																				
Config8_5 Config4_1 <i>Constants</i>	<p>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.</p> <table border="1"> <thead> <tr> <th>Digit</th> <th>Edge</th> </tr> </thead> <tbody> <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> </tbody> </table>	Digit	Edge	0	High level, rising edge	1	High level, falling edge	2	Low level, rising edge	3	Low level falling edge										
Digit	Edge																				
0	High level, rising edge																				
1	High level, falling edge																				
2	Low level, rising edge																				
3	Low level falling edge																				
Funct8_5 Funct4_1 <i>Constants</i>	<p>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.</p> <table border="1"> <thead> <tr> <th>Digit</th> <th>Results</th> </tr> </thead> <tbody> <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> </tbody> </table>	Digit	Results	0	None	1	Period (msec) between edges on this channel	2	Frequency (kHz) of edges on the channel	3	Time between an edge on the previous channel and the edge on this channel (msec)	4	time between an edge on channel 1 and the edge on this channel (msec)	5	Number of edges on channel 2 between the last edge on channel 1 and the edge on this channel using linear interpolation	6	Low resolution frequency (kHz) of edges on this channel	7	Total number of edges on this channel since last interrogation	8	Integer number of edges on channel 2 between the last edge on channel 1 and the edge on this channel.
Digit	Results																				
0	None																				
1	Period (msec) between edges on this channel																				
2	Frequency (kHz) of edges on the channel																				
3	Time between an edge on the previous channel and the edge on this channel (msec)																				
4	time between an edge on channel 1 and the edge on this channel (msec)																				
5	Number of edges on channel 2 between the last edge on channel 1 and the edge on this channel using linear interpolation																				
6	Low resolution frequency (kHz) of edges on this channel																				
7	Total number of edges on this channel since last interrogation																				
8	Integer number of edges on channel 2 between the last edge on channel 1 and the edge on this channel.																				
	<p>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.</p> <p>Note: the destination array must be dimensioned large enough to hold all the functions requested.</p>																				

Parameter & Data Type	Enter																								
OutputOpt	<p>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.</p> <table border="1" data-bbox="532 415 1333 1360"> <thead> <tr> <th data-bbox="532 415 618 447">Code</th> <th data-bbox="618 415 1333 447">Result</th> </tr> </thead> <tbody> <tr> <td data-bbox="532 447 618 590">0:</td> <td data-bbox="618 447 1333 590">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 data-bbox="532 590 618 758">32768</td> <td data-bbox="618 590 1333 758">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 data-bbox="532 758 618 926">nnnn</td> <td data-bbox="618 758 1333 926">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> <tr> <td data-bbox="532 926 618 1146">-nnnn</td> <td data-bbox="618 926 1333 1146">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 data-bbox="532 1146 618 1209">-9999</td> <td data-bbox="618 1146 1333 1209">Causes the INT8 to perform a self memory test. The signature of the INT8's PROM is returned to the datalogger.</td> </tr> <tr> <td data-bbox="618 1209 870 1241"></td> <td data-bbox="870 1209 1333 1241"> <table border="1" data-bbox="618 1209 1333 1360"> <thead> <tr> <th data-bbox="618 1209 870 1241">RESULT CODE</th> <th data-bbox="870 1209 1333 1241">DEFINITION</th> </tr> </thead> <tbody> <tr> <td data-bbox="618 1241 870 1272">0</td> <td data-bbox="870 1241 1333 1272">Bad ROM</td> </tr> <tr> <td data-bbox="618 1272 870 1304">-0</td> <td data-bbox="870 1272 1333 1304">Bad ROM, & bad RAM</td> </tr> <tr> <td data-bbox="618 1304 870 1335">positive integer</td> <td data-bbox="870 1304 1333 1335">ROM signature, good RAM</td> </tr> <tr> <td data-bbox="618 1335 870 1360">negative integer</td> <td data-bbox="870 1335 1333 1360">ROM signature, bad RAM</td> </tr> </tbody> </table> </td> </tr> </tbody> </table>	Code	Result	0:	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.	32768	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.	nnnn	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.	-nnnn	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.	-9999	Causes the INT8 to perform a self memory test. The signature of the INT8's PROM is returned to the datalogger.		<table border="1" data-bbox="618 1209 1333 1360"> <thead> <tr> <th data-bbox="618 1209 870 1241">RESULT CODE</th> <th data-bbox="870 1209 1333 1241">DEFINITION</th> </tr> </thead> <tbody> <tr> <td data-bbox="618 1241 870 1272">0</td> <td data-bbox="870 1241 1333 1272">Bad ROM</td> </tr> <tr> <td data-bbox="618 1272 870 1304">-0</td> <td data-bbox="870 1272 1333 1304">Bad ROM, & bad RAM</td> </tr> <tr> <td data-bbox="618 1304 870 1335">positive integer</td> <td data-bbox="870 1304 1333 1335">ROM signature, good RAM</td> </tr> <tr> <td data-bbox="618 1335 870 1360">negative integer</td> <td data-bbox="870 1335 1333 1360">ROM signature, bad RAM</td> </tr> </tbody> </table>	RESULT CODE	DEFINITION	0	Bad ROM	-0	Bad ROM, & bad RAM	positive integer	ROM signature, good RAM	negative integer	ROM signature, bad RAM
Code	Result																								
0:	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.																								
32768	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.																								
nnnn	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.																								
-nnnn	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.																								
-9999	Causes the INT8 to perform a self memory test. The signature of the INT8's PROM is returned to the datalogger.																								
	<table border="1" data-bbox="618 1209 1333 1360"> <thead> <tr> <th data-bbox="618 1209 870 1241">RESULT CODE</th> <th data-bbox="870 1209 1333 1241">DEFINITION</th> </tr> </thead> <tbody> <tr> <td data-bbox="618 1241 870 1272">0</td> <td data-bbox="870 1241 1333 1272">Bad ROM</td> </tr> <tr> <td data-bbox="618 1272 870 1304">-0</td> <td data-bbox="870 1272 1333 1304">Bad ROM, & bad RAM</td> </tr> <tr> <td data-bbox="618 1304 870 1335">positive integer</td> <td data-bbox="870 1304 1333 1335">ROM signature, good RAM</td> </tr> <tr> <td data-bbox="618 1335 870 1360">negative integer</td> <td data-bbox="870 1335 1333 1360">ROM signature, bad RAM</td> </tr> </tbody> </table>	RESULT CODE	DEFINITION	0	Bad ROM	-0	Bad ROM, & bad RAM	positive integer	ROM signature, good RAM	negative integer	ROM signature, bad RAM														
RESULT CODE	DEFINITION																								
0	Bad ROM																								
-0	Bad ROM, & bad RAM																								
positive integer	ROM signature, good RAM																								
negative integer	ROM signature, bad RAM																								
CaptureTrig <i>Constant, Variable, or Expression</i>	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 <i>Constant, Variable, Array, or Expression</i>	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.																								

SDMIO16 (Dest, Status, Address, Command, Mode Ports 16-13, Mode Ports 12-9, Mode Ports 8-5, Mode Ports 4-1, Mult, Offset)

The SDMIO16 instruction is used to set up and measure an SDM-IO16 control port expansion device.

The ports on the SDM-IO16 can be configured for either input or output. When configured as input, the SDM-IO16 can measure the logical state of each port, count pulses, and measure the frequency of and determine the duty cycle of applied signals. The module can also be programmed to generate an interrupt signal to the datalogger when one or more input signals change state. When configured as an output, each port can be set to 0 or 5 V by the datalogger. In addition to being able to drive normal logic level inputs, when an output is set high a 'boost' circuit allows it to source a current of up to 100 mA, allowing direct control of low voltage valves, relays, etc.

Parameter & Data Type	Enter	
Dest <i>Variable Array</i>	The variable or variable array in which to store the results of the measurement (Command codes 1 - 69, 91, 92, 99) or the Source value for the Command Codes (70 - 85, 93 - 98). The variable array for this parameter must be dimensioned to accommodate the number of values returned (or sent) by the instruction.	
Status <i>Variable</i>	The variable in which to store the result of the command issued by the instruction. If the command is successful a 0 is returned; otherwise, the value is incremented by 1 with each failure.	
SDMAddress <i>Constant</i>	The SDMAddress parameter defines the address of the SDM-IO16 with which to communicate. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. If the Repts parameter is greater than 1, the datalogger will increment the SDM address for each subsequent SDM-IO16 with which it communicates.	
Command <i>Constant or Variable</i>	The following are valid command options:	
	Command Code	Description
	1	Read port 1's accumulated counts into Dest
	2	Read port 2's accumulated counts into Dest
	3	Read port 3's accumulated counts into Dest
	4	Read port 4's accumulated counts into Dest
	5	Read port 5's accumulated counts into Dest
	6	Read port 6's accumulated counts into Dest
	7	Read port 7's accumulated counts into Dest
	8	Read port 8's accumulated counts into Dest
	9	Read port 9's accumulated counts into Dest
	10	Read port 10's accumulated counts into Dest
	11	Read port 11's accumulated counts into Dest
	12	Read port 12's accumulated counts into Dest
	13	Read port 13's accumulated counts into Dest
	14	Read port 14's accumulated counts into Dest
	15	Read port 15's accumulated counts into Dest
16	Read port 16's accumulated counts into Dest	

Parameter & Data Type	Enter	
	17	Read ports 1-4's accumulated counts into Dest (Dest must be dimensioned to 4)
	18	Read ports 5-8's accumulated counts into Dest (Dest must be dimensioned to 4)
	19	Read ports 9-12's accumulated counts into Dest (Dest must be dimensioned to 4)
	20	Read ports 13-16's accumulated counts into Dest (Dest must be dimensioned to 4)
	21	Read ports 1-8's accumulated counts into Dest (Dest must be dimensioned to 8)
	22	Read ports 9-16's accumulated counts into Dest (Dest must be dimensioned to 8)
	23	Read ports 1-16's accumulated counts into Dest (Dest must be dimensioned to 16)
	24	Read port 1's frequency into Dest
	25	Read port 2's frequency into Dest
	26	Read port 3's frequency into Dest
	27	Read port 4's frequency into Dest
	28	Read port 5's frequency into Dest
	29	Read port 6's frequency into Dest
	30	Read port 7's frequency into Dest
	31	Read port 8's frequency into Dest
	32	Read port 9's frequency into Dest
	33	Read port 10's frequency into Dest
	34	Read port 11's frequency into Dest
	35	Read port 12's frequency into Dest
	36	Read port 13's frequency into Dest
	37	Read port 14's frequency into Dest
	38	Read port 15's frequency into Dest
	39	Read port 16's frequency into Dest
	40	Read ports 1-4's frequency into Dest (Dest must be dimensioned to 4)
	41	Read ports 5-8's frequency into Dest (Dest must be dimensioned to 4)
	42	Read ports 9-12's frequency into Dest (Dest must be dimensioned to 4)
	43	Read ports 13-16's frequency into Dest (Dest must be dimensioned to 4)
	44	Read ports 1-8's frequency into Dest (Dest must be dimensioned to 8)
	45	Read ports 9-16's frequency into Dest (Dest must be dimensioned to 8)
	46	Read ports 1-16's frequency into Dest (Dest must be dimensioned to 16)
	47	Read port 1's duty cycle into Dest
	48	Read port 2's duty cycle into Dest
	49	Read port 3's duty cycle into Dest

Parameter & Data Type	Enter	
	50	Read port 4's duty cycle into Dest
	51	Read port 5's duty cycle into Dest
	52	Read port 6's duty cycle into Dest
	53	Read port 7's duty cycle into Dest
	54	Read port 8's duty cycle into Dest
	55	Read port 9's duty cycle into Dest
	56	Read port 10's duty cycle into Dest
	57	Read port 11's duty cycle into Dest
	58	Read port 12's duty cycle into Dest
	59	Read port 13's duty cycle into Dest
	60	Read port 14's duty cycle into Dest
	61	Read port 15's duty cycle into Dest
	62	Read port 16's duty cycle into Dest
	63	Read ports 1-4's duty cycle into Dest (Dest must be dimensioned to 4)
	64	Read ports 5-8's duty cycle into Dest (Dest must be dimensioned to 4)
	65	Read ports 9-12's duty cycle into Dest (Dest must be dimensioned to 4)
	66	Read ports 13-16's duty cycle into Dest (Dest must be dimensioned to 4)
	67	Read ports 1-8's duty cycle into Dest (Dest must be dimensioned to 8)
	68	Read ports 9-16's duty cycle into Dest (Dest must be dimensioned to 8)
	69	Read ports 1-16's duty cycle into Dest (Dest must be dimensioned to 16)
	70	Set port 1's debounce time from Dest
	71	Set port 2's debounce time from Dest
	72	Set port 3's debounce time from Dest
	73	Set port 4's debounce time from Dest
	74	Set port 5's debounce time from Dest
	75	Set port 6's debounce time from Dest
	76	Set port 7's debounce time from Dest
	77	Set port 8's debounce time from Dest
	78	Set port 9's debounce time from Dest
	79	Set port 10's debounce time from Dest
	80	Set port 11's debounce time from Dest
	81	Set port 12's debounce time from Dest
	82	Set port 13's debounce time from Dest
	83	Set port 14's debounce time from Dest
	84	Set port 15's debounce time from Dest
	85	Set port 16's debounce time from Dest
	86	Set port 16-13 from Mode parameter
	87	Set port 12-9 from Mode parameter
	88	Set port 8-5 from Mode parameter
	89	Set port 4-1 from Mode parameter

Parameter & Data Type	Enter	
	90	Set port 16-1 from Mode parameters
	91	Read state of ports 1-16 into one variable (Dest). The result is a 16 bit decimal representation from 0-65535.
	92	Read state of ports 1-16 into 16 separate variables (Dest must be dimensioned to 16). Dest(1) holds the state of port 1, Dest(2) port 2, etc. State is represented by 0 or 1.
	93	Set state of ports 1-16 from a single variable (Dest). Dest should be a 16 bit decimal representation from 0-65535.
	94	Set state of ports 1-16 from 16 separate variables (Dest must be dimensioned to 16). Dest(1) sets the state of port 1, Dest(2) port 2, etc. State is represented by 0 or 1.
	95	Set direction of ports 1-16 from a single variable (Dest). Dest should be a 16 bit decimal representation from 0-65535.
	96	Set direction of ports 1-16 from 16 separate variables (Dest must be dimensioned to 16). Dest(1) sets the direction of port 1, Dest(2) port 2, etc. Direction is represented by 0 or 1.
	97	Set interrupt mask of ports 1-16 from a single variable (Dest). Dest should be a 16 bit decimal representation from 0-65535.
	98	Set interrupt mask of ports 1-16 from 16 separate variables (Dest should be dimensioned to 16). Dest(1) sets port 1, Dest(2) port 2, etc. The mask is represented by 0 or 1.
	99	Read the OS signature, OS version and counters for watchdog resets and communication errors into 4 separate variables (Dest must be dimensioned to 4). Using this command also resets the counters.
Mode <i>Constant or Variable</i>	The Mode parameter configures a bank of four ports when a Command code 86 through 90 is used (if any other Command Code is used, enter 0 for the Mode). Mode is entered as four digits, where each digit indicates the setting for a port. Ports are represented from the highest port number to the lowest, from left to right (e.g., 16 15 14 13; 12 11 10 9; 8 7 6 5; 4 3 2 1). There is a Mode for Ports 16 - 13, 12 - 9, 8 - 5, and 4 - 1. The valid codes are:	
	Code	Description
	0	Output logic low
	1	Output logic high
	2	Input digital, no debounce filter
	3	Input switch closure 3.17 msec debounce filter
	4	Input digital interrupt enabled, no debounce filter
	5	Input switch closure interrupt enabled 3.17 msec, debounce filter
	6	Undefined
	7	Undefined
	8	Undefined
	9	No change
Mult, Offset	The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement.	

SDMSIO4 (Dest, Reps, SDMAAddress, Mode, Command, Param1, Param2, ValuesPerRep, Multiplier, Offset)

The SDMSIO4 instruction is used to control and transmit/retrieve data from a Campbell Scientific SDM-SIO4 Interface (4 Channel Serial Input/Output device). See the SDM-SIO4 Serial Input Interface manual for operation details.

For instructions used for serial input and output without the SDM-SIO4, see Section 12.

Parameter & Data Type	Enter																		
Dest <i>Variable or Array</i>	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.																		
Reps <i>Constant or Expression</i>	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 fourth through sixth elements of the Dest array.																		
SDMAAddress <i>Constant</i>	The SDMAAddress parameter defines the address of the SIO4 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 SIO4 that it communicates with. The SDM address is entered as a base 10 number, unlike older, jumper-settable SDM instruments that used base 4.																		
Mode	The mode parameter defines which port the instruction will affect. <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </table>	Code	Description	1	Send/Receive Port 1	2	Send/Receive Port 2	3	Send/Receive Port 3	4	Send/Receive Port 4	5	Send to all Ports (global)						
Code	Description																		
1	Send/Receive Port 1																		
2	Send/Receive Port 2																		
3	Send/Receive Port 3																		
4	Send/Receive Port 4																		
5	Send to all Ports (global)																		
Command	The Command parameter is used to configure the SIO4. The commands are listed briefly below. See the SDM-SIO4 manual for details. <table border="1"> <thead> <tr> <th>Code</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </table>	Code	Description	1	Poll of available data.	2	Get EPROM and memory signatures.	3	Flush all receive buffers.	4	Send data to datalogger.	5	Return number of watchdog errors, invalid command executed, and lithium battery voltage.	6	Flush transmit buffer.	7	Activate command line.	8	Poll TX buffers for data.
Code	Description																		
1	Poll of available data.																		
2	Get EPROM and memory signatures.																		
3	Flush all receive buffers.																		
4	Send data to datalogger.																		
5	Return number of watchdog errors, invalid command executed, and lithium battery voltage.																		
6	Flush transmit buffer.																		
7	Activate command line.																		
8	Poll TX buffers for data.																		

Parameter & Data Type	Enter
	9 Flush converted data buffer. 66 Send single-byte data to datalogger. 67 Get return code. 320 Send byte data to SDM-SIO4. 321 Execute command line command. 1024 Send string to SIO4. 1025 Transmit a byte. 1026 Serial port status. 1027 Manual handshake mode. 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.

SDMSpeed (BitPeriod)

Changes the rate that the CR3000 uses to clock the SDM data. Slowing down the clock rate may be necessary when long cables lengths are used to connect the CR3000 and SDM devices.

Parameter & Data Type	Enter
BitPeriod <i>Constant or variable</i>	The time per bit, in microseconds. Initial Setting (default): 26.04 μ S Resolution: 4.34 μ S Min Setting: 8.68 μ S Max Setting: 2.2 mS SDMSpeed(30) gives: 26.04 μ S SDMSpeed(k) gives: $bit_rate = INT((k*144)/625) * Resolution$ When calculating SDMSpeed(k), the loggers round down to the next higher bit rate.

SDMSW8A (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 <i>Variable or Array</i>	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 <i>Constant</i>	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 <i>Constant</i>	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.															
FuncOp <i>Constant</i>	The FuncOp is used to determine the result that will be returned by the SW8A.															
	<table border="1"> <thead> <tr> <th>Numeric Code</th> <th>Function</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>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.</td> </tr> <tr> <td>1</td> <td>Returns the duty cycle of the signal. The result is the percentage of time the signal is high during the scan interval.</td> </tr> <tr> <td>2</td> <td>Returns a count of the number of positive transitions of the signal.</td> </tr> <tr> <td>3</td> <td>Returns a value indicating the condition of the module: <table border="1"> <tr> <td>positive integer:</td> <td>ROM and RAM are good</td> </tr> <tr> <td>negative value:</td> <td>RAM is bad</td> </tr> <tr> <td>Zero:</td> <td>ROM is bad</td> </tr> </table> </td> </tr> </tbody> </table>	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: <table border="1"> <tr> <td>positive integer:</td> <td>ROM and RAM are good</td> </tr> <tr> <td>negative value:</td> <td>RAM is bad</td> </tr> <tr> <td>Zero:</td> <td>ROM is bad</td> </tr> </table>	positive integer:	ROM and RAM are good	negative value:	RAM is bad	Zero:
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: <table border="1"> <tr> <td>positive integer:</td> <td>ROM and RAM are good</td> </tr> <tr> <td>negative value:</td> <td>RAM is bad</td> </tr> <tr> <td>Zero:</td> <td>ROM is bad</td> </tr> </table>	positive integer:	ROM and RAM are good	negative value:	RAM is bad	Zero:	ROM is bad									
positive integer:	ROM and RAM are good															
negative value:	RAM is bad															
Zero:	ROM is bad															
StartChan <i>Constant</i>	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 <i>Constant, Variable, Array, or Expression</i>	The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement.															

SDMTrigger

When SDMTrigger is executed, the CR3000 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 CR3000 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.

SDMX50 (SDMAddress, Channel)

SDMX50 allows individual multiplexer switches to be activated independently of the TDR100 Instruction.

SDMX50 is useful for selecting a particular probe to troubleshoot or to determine the apparent cable length.

Because it is usually easy to hear the multiplexer(s) switch, the SDMX50 instruction is a convenient method to test the addressing and wiring of a level of multiplexers: Program the datalogger to scan every few seconds with the SDM address for the multiplexer(s) and channel 8. The Instruction always starts with channel 1 and switches through the channels to get to the programmed channel. Switching to channel 8 will cause the most prolonged noise.

Remember each multiplexer level has a different SDM Address. Level 1 multiplexers should be set to the address 1 greater than the TDR100, Level 2 multiplexers should be set to the address 2 greater than the TDR100 and Level 3 multiplexers should be set to the address 3 greater than the TDR100. If the SDMX50 multiplexers for a given level are connected and have their addresses set correctly they should all switch at the same time.

Parameter & Data Type	Enter
SDMAddress <i>Constant</i>	The SDMAddress of the SDMX50 to switch. Valid SDM addresses are 0 through 14.
Channel <i>Constant</i>	The SDMX50 channel to switch to (1-8)

TDR100 (Dest, SDMAddress, Option, Mux/ProbeSelect, WaveAvg, Vp, Points, CableLength, WindowLength, ProbeLength, ProbeOffset, Mult, Offset)

This instruction can be used to measure one TDR probe connected to the TDR100 directly or multiple TDR probes connected to one or more SDMX50 multiplexers.

Parameter & Data Type	Enter
Dest <i>Variable or Array</i>	The Dest parameter is a variable or variable array in which to store the results of the measurement. The variable must be dimensioned to accommodate all of the values returned by the instruction, which is determined by the Option parameter.
SDMAddress <i>Constant</i>	The SDMAddress parameter defines the address of the TDR100 with which to communicate. Valid SDM addresses are 0 through 14. Address 15 is reserved for the SDMTrigger instruction. If the Repts parameter is greater than 1, the datalogger will increment the SDM address for each subsequent TDR100 that it communicates with. Note: CRBasic dataloggers are programmed using the base 10 address (0-14) Edlog programmed dataloggers (e.g., CR10X, CR23X) used base 4

Parameter & Data Type	Enter
Option <i>Constant or Variable</i>	The Option parameter determines the output of the instruction.
	Code Description
	0 Measure La/L (ratio of apparent to physical probe rod length)
	1 Collect Waveform values - Outputs reflection waveform values as an array of floating point numbers with a range of -1 to 1. The waveform values are prefaced by a header containing values of key parameters for this instruction (averaging, propagation velocity, points, cable length, window length, probe length, probe offset, multiplier, offset)
2 Collect Waveform plus First Derivative - Returns $(2*n-5)+9$ values where n is the number of waveform reflection values specified by the Points parameter.	
3 Measure Electrical Conductivity - Outputs a value that when multiplied by the Multiplier parameter determines soil bulk electrical conductivity in S/m.	
Mux/ProbeSelect	The Mux/Probe Select parameter is used to define the setup of any multiplexers and attached probes in the system. The addressing scheme used is ABCR, where A = level 1 multiplexer channel, B = level 2 multiplexer channel, C = level 3 multiplexer channel, and R = the number of consecutive probes to be read, starting with the channel specified by the ABC value (maximum of 8). 0 is entered for any level not used.
WaveAvg <i>Constant</i>	The WaveAvg parameter is used to define the number of waveform reflections averaged by the TDR100 to give a single result. A waveform averaging value of 4 provides good signal-to-noise ratio under typical applications. Under high noise conditions averaging can be increased. The maximum averaging possible is 128.
Vp <i>Constant</i>	The Vp parameter allows you to enter the propagation velocity of a cable when using the instruction to test for cable lengths or faults. Vp adjustment is not necessary for soil water content or electrical conductivity measurement and should be set to 1.0 for output Option 1, 2, or 3.
Points <i>Constant or Variable</i>	The Points parameter is used to define the number of values in the displayed or collected waveform (20 to 2048). An entry of 251 is recommended for soil water measurements. The waveform consists of the number of Points equally spaced over the WindowLength.
CableLength <i>Constant or Variable</i>	<p>The CableLength parameter is used to specify the cable length, in meters, of the TDR probes. If a 0 is entered for the Option parameter, cable length is used by the analysis algorithm to begin searching for the TDR probe. If a 1 or 2 is entered for the Option parameter, cable length is the distance to the start of the collected waveform.</p> <p>The value used for CableLength is best determined using PCTDR100 with the $V_p = 1.0$. Adjust the CableLength and WindowLength values in PCTDR100 until the probe reflection can be viewed. Subtract about 0.5 meters from the distance associated with the beginning of the probe reflection.</p> <p>Note that the specified CableLength applies to all probes read by this instruction; therefore, all probes must have the same cable lengths.</p>

Parameter & Data Type	Enter
WindowLength <i>Constant or Variable</i>	The WindowLength parameter specifies the length, in meters, of the waveform to collect or analyze. The waveform begins at the CableLength and ends at the CableLength + WindowLength. This is an apparent length because the value set for Vp may not be the actual propagation velocity. For water content measurements, the WindowLength must be large enough to contain the entire probe reflection for probes with 20 to 30 cm rods. A Vp = 1 and Window length = 5 is recommended.
ProbeLength <i>Constant or Variable</i>	The ProbeLength parameter specifies the length, in meters, of the probe rods that are exposed to the medium being measured. The value of this parameter only has an affect when Option 0, La/L, is used for the measurement.
ProbeOffset <i>Constant or Variable</i>	The ProbeOffset is an apparent length value used to correct for the portion of the probe rods that may be encapsulated in epoxy and not surrounded by soil or other medium being measured. This value is supplied by Campbell Scientific for the probes we manufacture. The value of this parameter only has an affect when Option 0, La/L, is used for the measurement.
Mult, Offset	The Mult and Offset parameters are each a constant, variable, array, or expression by which to scale the results of the measurement.

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 (Number)

Returns the absolute value of a number.

Syntax

x = **ABS** (*number*)

Remarks

Number 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 Value, X1, X2           'Declare variables.
Public Volt1, X, Public Precision 'Declare Public Variables
BeginProg
  Precision = 0.01          'Set Precision
  Scan(1,Sec,0,0)          'Scan at 1 Hz
  VoltDiff(Volt1,1,mV50,1,False,250,250,1.8,32)
  Value = Volt1             'Volt1 will be evaluated.
  X1 = 0
  X2 = Value                'Make first two guesses.
  'Loop until difference between guesses is less than precision.
  Do Until ABS(Volt1 - X * X * X) < Precision
  X = (X1 + X2) / 2
  If X * X * X - Value < 0 Then 'Adjust guesses.
    X1 = X
  Else
    X2 = X
```

```

EndIf
Loop
'X is now the cube root of Volt1
NextScan
EndProg

```

ACOS (Number)

The ACOS function returns the arc cosine of a number.

Syntax

x = ACOS (*Number*)

Remarks

The number 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 COS, 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.

```

AddPrecise (PrecisionVariable, X)

The AddPrecise function allows you to do high precision totalizing of variables or manipulation of high precision variables.

Syntax

AddPrecise (PrecisionVariable, X)

Remarks

In this function, the variable X is added to the PrecisionVariable. Every reference to the PrecisionVariable will cause a 32 bit extension of its mantissa to be saved and used internally. A normal single precision float has 24 bits of mantissa; therefore, this new precision is 56 bits. This function can be useful when trying to find the difference between two high precision variables.

PrecisionVariable The PrecisionVariable is the variable that will be affected by the precision add.

X The X variable is the value that will be added to the PrecisionVariable. It may or may not be a high precision variable, depending upon whether it has been declared as such in a previous AddPrecise or MovePrecise instruction.

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 <i>expr1</i> is	And <i>expr2</i> 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 <i>expr1</i> is	And bit in <i>expr2</i> 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	<i>'Declare variables.</i>
A = 10: B = 8: C = 6	<i>'Assign values.</i>
If A > B And B > C Then	<i>'Evaluate expressions.</i>
Msg = True	
Else	
Msg = False	
End If	

ASIN (Number)

The ASIN function returns the arc sine of a number.

Syntax

x = **ASIN** (*number*)

Remarks

Number 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	<i>'Declare variables.</i>
Pi = 2 * ASin(1)	<i>'Calculate Pi.</i>

ATN (Number)

Returns the arc tangent of a number.

Syntax

x = **ATN** (*number*)

Remarks

Number 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 $-\pi/2$ to $\pi/2$ radians.

To convert degrees to radians, multiply degrees by $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

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 ($1/\text{TAN}$).

ATN Function Example

The example uses ATN to calculate π . By definition, a full circle is 2π radians. ATN(1) is $\pi/4$ radians (45 degrees).

Dim Pi	<i>'Declare variables.</i>
Pi = 4 * Atn(1)	<i>'Calculate Pi.</i>

ATN2(Y, X)

The ATN2 function returns the arc tangent of y/x .

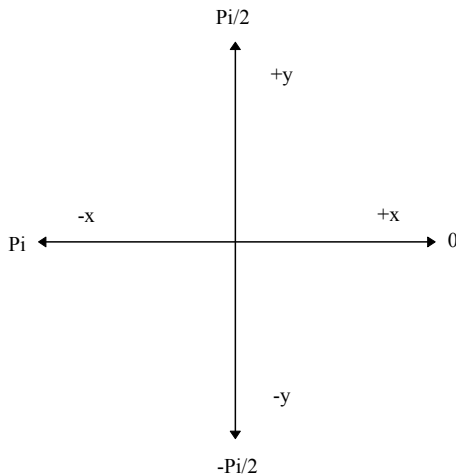
Syntax

x = ATN2 (Y, X)

Remarks

ATN2 function calculates the arctangent of Y/X returning a value in the range from π 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 $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.



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 ($1/\text{TAN}$).

ATN2 Function Example

The example uses ATN2 to calculate π . By definition, a full circle is 2π radians. ATN2(1,1) is $\pi/4$ radians (45 degrees).

Dim Pi	<i>'Declare variables.</i>
Pi = 4 * ATN2(5, 5)	<i>'Calculate Pi.</i>

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 <i>Variable or Array</i>	The variable or array in which to store the average(s).
Reps <i>Constant</i>	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 <i>Constant</i>	The number of values to include in the running average.
Source <i>Array</i>	The name of the variable or array that is to be averaged.

Example

```

Public PTemp, TCTemp(5), RunAvgTC(5) 'Declare Public Variables
BeginProg
Scan(1,Sec,0,0) 'Scan at 1 Hz
PanelTemp (PTemp,250)
TCDiff (TCTemp(),5,mV20,1,TypeT,PTemp,True ,0,250,1.0,0)
AvgRun (RunAvgTC(),5,TCTemp(),60)
NextScan
EndProg 'Program ends here
    
```

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 <i>Variable</i>	The variable in which to store the results of the instruction.
Swath <i>Constant</i>	The number of values of the source array to average.
Source <i>Array</i>	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))
```

COS (Angle)

Returns the cosine of an angle specified in radians.

Syntax

$x = \text{COS}(\text{angle})$

Remarks

Angle 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 $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

COS Function Example

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 (Expression)

The COSH function returns the hyperbolic cosine of an expression or value.

Syntax

x = COSH (*expression*)

Remarks

The COSH function takes a value and returns the hyperbolic cosine [COSH(x) = 0.5(e^x + 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	<i>'Declare variables.</i>
BeginProg	
Scan (1,Sec,3,0)	
VoltDiff (Volt1,1,mV5000,1,True ,200,500,1.0,0)	<i>'Return voltage on</i>
	<i>DiffChan1</i>
Ans = COSH(Volt1)	
NextScan	
EndProg	

CovSpa

The CovSpa instruction computes the covariance(s) of sets of data that are loaded into arrays.

Syntax

CovSpa(Dest, NumOfCov, SizeOfSets, CoreSet; DataSets)

CovSpa calculates the covariance(s) between the data in the CoreSet and one or more data sets in the DataSets. The covariance of the sets of data X and Y is calculated as:

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

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 <i>Variable or Array</i>	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 <i>Constant</i>	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 <i>Constant</i>	The number of values in the data sets for the covariance calculations.
CoreSet <i>Array</i>	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.
DataSets <i>Array</i>	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 $4 \times 100 = 400$. 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.

CSGN (Number)

Changes the sign of a number or variable.

This function will change the sign (+ or -) of a value. Typically, it is used in the form of $\text{VariableA} = \text{CSGN}(\text{VariableB})$.

DewPoint (Dest, Temp, RH)

Parameter & Data Type	Enter
Dest <i>Variable</i>	The variable in which to store the dew point temperature (°C).
Temp <i>Variable</i>	The variable that contains air temperature (°C).
RH <i>Variable</i>	The variable that contains RH (%).

The dew point instruction calculates the dew point temperature from previously measured values of RH and air temperature. While end results may not be quite as accurate as those from a dedicated dew point sensor, they are acceptable for a wide range of applications.

Calculating Dew Point

Measure the relative humidity (RH) and air temperature (T_a ; units °C) with the appropriate instruction for the sensors you are using.

Dew point temperature is calculated as follows:

1. The saturation vapor pressure (S_{vp} ; units kPa) is calculated using Lowe's equation (see SatVP).
2. The vapor pressure (V_p ; units kPa) is calculated from $V_p = RH * S_{vp} / 100$.
3. The dew point (T_d ; units °C) is calculated from the inverse of a version of Tetens' equation, optimized for dewpoints in the range -35 to 50°C:

$$T_d = (C_3 * \ln(V_p / C_1)) / (C_2 * \ln(V_p / C_1))$$

where:

$$C_1 = 0.61078$$

$$C_2 = 17.558$$

$$C_3 = 241.88$$

Error in the Estimation of Dew Point

Tetens' equation is an approximation of the true variation of saturated vapor pressure as a function of temperature. However, the errors in using the inverted form of the equation result in dew point errors much less than 0.1°C.

The largest component of error, in reality, comes from errors in the absolute calibration of the temperature and RH sensor.

Figure 8-1 shows how dew point varies as a function of temperature and humidity. It can be seen that the response is non-linear with respect to both variables. Errors in the measurement of RH and temperature thus form a complex function in relation to the resultant error in estimated dew point. In practice, the effect of errors in the calibration of air temperature can be taken to translate to an equivalent error in dew point, e.g. if the air temperature sensor is 0.2°C high, then the estimated dew point is approximately 0.2°C high. Figure 8-2 shows the errors in dew point as a function of a 'worst case' 5% error in the calibration of the RH sensor.

For sensors installed in the field there are additional errors associated with exposure of the sensor, e.g. sensors in un aspirated shields get slightly warmer than true air temperature in conditions of low wind speeds and high solar radiation. However, if the RH and air temperature sensors are installed in the same shield and are thus exposed identically, the estimate of dew point is not subject to the same error as the measurement of air temperature would be. This is because the temperature sensor will measure the actual temperature of the RH sensor, which is what is required for the derivation of air vapor pressure and thereby dew point.

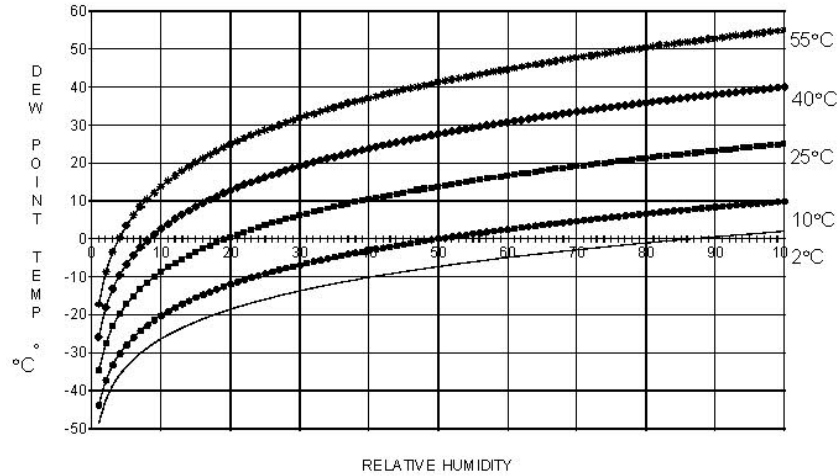


FIGURE 8-1. Dew Point Temperature over the RH Range for Selected Air Temperatures

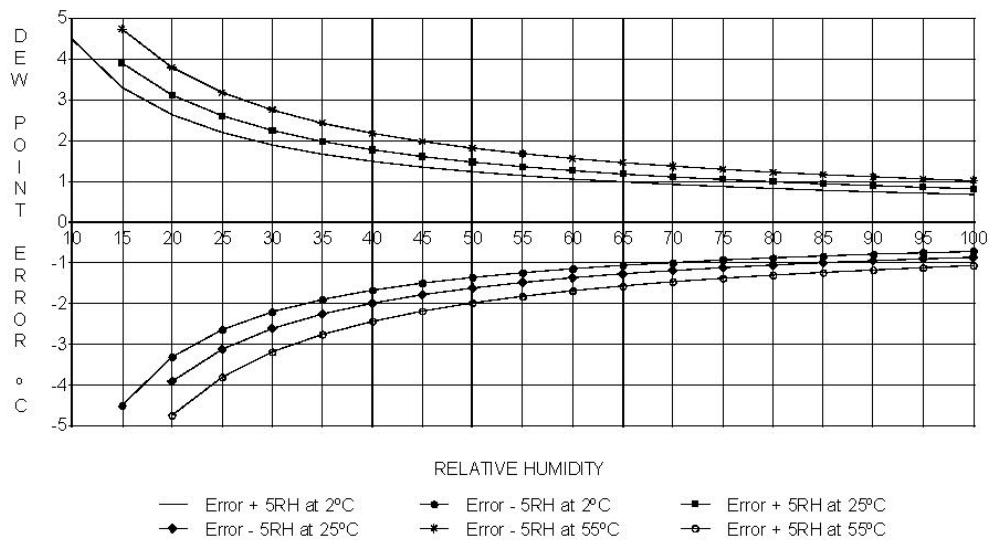


FIGURE 8-2. Effect of RH Errors on Calculated Dew Point (± 5 RH Unit Error at Three Air Temperatures)

EXP

Returns e (the base of natural logarithms) raised to a power.

Syntax

$x = \text{EXP}(\text{number})$

Remarks

If the value of the number argument 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 Function Example

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 e¹ or e.

```
Dim ValueOfE           'Declare variables.
BeginProg
ValueOfE = Exp(1)      'Calculate value of e.
EndProg
```

FFTSpa (Dest, N, SrcArray, SampleInterval, 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 <i>Array</i>	The array in which to store the results of FFT.		
N <i>Constant</i>	Number of points in the original time series. The number of points must be a power of 2 (i.e., 512, 1024, 2048, etc.).		
SrcArray <i>Variable</i>	The name of the Variable array that contains the input data for the FFT		
SampleInterval <i>Constant</i>	The sampling interval of the time series.		
Units <i>Constant</i>	The units for SampleInterval.		
	Alpha Code	Numeric Code	Units
	MSEC	1	milliseconds
	SEC	2	seconds
MIN	3	minutes	

Parameter & Data Type	Enter	
Options <i>Constant</i>	A code to indicate what values to calculate and output.	
	Code	Result
	0	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.
	1	Amplitude spectrum. The output is N/2 magnitudes. With $\text{Acos}(wt)$; A is magnitude.
	2	Amplitude and Phase Spectrum. The output is N/2 pairs of magnitude and phase; with $\text{Acos}(wt - \phi)$; A is amplitude, ϕ is phase $(-\pi, \pi)$.
	3	Power Spectrum. The output is N/2 values normalized to give a power spectrum. With $\text{Acos}(wt - \phi)$, the power is $A^2 / 2$. The summation of the N/2 values yields the total power in the time series signal.
4	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 $T = N \cdot \tau$ 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.	
5	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 FFT real time series. The output is the time series whose FFT would result in the input array.	

$T = N \cdot \text{SampleInterval}$: the length, in seconds, of the time series.
Processing field: "FFT,N,SampleInterval,option". Tick marks on the x axis are $1/(N \cdot \text{SampleInterval})$ Herz. N/2 values, or pairs of values, are output, depending upon the option code.

Normalization details:

Complex FFT result i , $i = 1 \dots N/2$: $a_i \cdot \cos(w_i \cdot t) + b_i \cdot \sin(w_i \cdot t)$.

$w_i = 2\pi(i-1)/T$.

$\phi_i = \text{atan2}(b_i, a_i)$ (4 quadrant arctan)

Power(1) = $(a_1^2 + b_1^2)/N^2$ (DC)

Power(i) = $2 \cdot (a_i^2 + b_i^2)/N^2$ ($i = 2 \dots N/2$, AC)

PSD(i) = Power(i) * T = Power(i) * N * tau

A1 = $\sqrt{a_1^2 + b_1^2}/N$ (DC)

Ai = $2 \cdot \sqrt{a_i^2 + b_i^2}/N$ (AC)

Notes:

- Power is independent of the sampling rate ($1/\text{SampleInterval}$) and of the number of samples (N).
- The PSD is proportional to the length of the sampling period ($T = N \cdot \text{SampleInterval}$), 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.

FIX (Number)

Used to return the integer portion of a number.

See INT.

Frac (Number)

Returns the fractional part of a number.

Syntax

x = FRAC (Number)

Remarks

Returns the fractional portion of the value in the number argument.

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 <i>Array</i>	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 <i>name</i>	The name of the data table from which to retrieve the record.
RecsBack <i>Const. or variable</i>	The number of records to back up and retrieve data (the most recent record is considered record 1).

IfTime

The IfTime instruction is used to return a number indicating True (-1) or False (0) based on the datalogger's real-time clock.

See also **TimeIntoInterval**.

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 <i>constant</i>	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 <i>constant</i>	The Interval is how often IfTime will be True.																
Units <i>Constant</i>	<table border="1"> <thead> <tr> <th colspan="2">The time units for TintoInt and Interval</th> </tr> <tr> <th>Alpha Code</th> <th>Units</th> </tr> </thead> <tbody> <tr> <td>Usec</td> <td>microseconds</td> </tr> <tr> <td>Msec</td> <td>Milliseconds</td> </tr> <tr> <td>Sec</td> <td>Seconds</td> </tr> <tr> <td>Min</td> <td>Minutes</td> </tr> <tr> <td>Hr</td> <td>Hours</td> </tr> <tr> <td>Day</td> <td>Days</td> </tr> </tbody> </table>	The time units for TintoInt and Interval		Alpha Code	Units	Usec	microseconds	Msec	Milliseconds	Sec	Seconds	Min	Minutes	Hr	Hours	Day	Days
The time units for TintoInt and Interval																	
Alpha Code	Units																
Usec	microseconds																
Msec	Milliseconds																
Sec	Seconds																
Min	Minutes																
Hr	Hours																
Day	Days																

IIF (Expression, TrueValue, FalseValue)

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 <i>Expression or Variable</i>	The Variable or expression to test.
	Value Result
	≠0 True: return TrueValue
	0 False: return FalseValue
TrueValue <i>Constant Variable or Expression</i>	The Value (or expression determining the value) to return if the test condition is true
FalseValue <i>Constant Variable or Expression</i>	The Value (or expression determining the value) to return if the test condition is False

IMP

Performs 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
False	True	True
False	False	True
False	Null	True

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

Returns the integer portion of a number.

Syntax

x = **INT** (*Number*)

x = **FIX** (*Number*)

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 (NAN).

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.

INT and FIX Function Example

This example illustrates the use of INT and FIX.

Dim A, B, C, D	<i>'Declare variables.</i>
BeginProg	
A = INT(-99.8)	<i>'Returns -100</i>
B = FIX(-99.8)	<i>'Returns -99</i>
C = INT(99.8)	<i>'Returns 99</i>
D = FIX(99.8)	<i>'Returns 99</i>
EndProg	

LN (Number)

LOG (Number)

Returns the natural logarithm of a number. Ln and Log perform the same function.

Syntax

x = **LOG** (*Number*)

x = **LN** (*Number*)

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:

$$\text{Log}_n(x) = \text{Log}(x) / \text{Log}(n)$$

The following example illustrates a procedure that calculates base-10 logarithms:

$$\text{Log10} = \mathbf{Log}(X) / \mathbf{Log}(10)$$

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

$$x = \mathbf{LOG10}(\textit{number})$$

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:

$$\mathbf{LOGN}(x) = \mathbf{LOG10}(x) / \mathbf{LOG10}(n)$$

LOG10 Function Example

This example uses the LOG10 instruction to calculate the log base 2 of 1000.

```

Dim LOG2_1000                            'Declare variables.
BeginProg
LOG2_1000 = LOG10(1000)/LOG10(2)
EndProg
    
```

MaxSpa (Dest, Swath, Source)

Finds the maximum value in an array.

Syntax

$$\mathbf{MaxSpa}(\textit{Dest}, \textit{Swath}, \textit{Source})$$

Remarks

Finds the maximum value in the given array and places 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

Finds the minimum value in the given array and places 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 $A = 19 \text{ Mod } 6.7$, A (which is result) equals 5.6. 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, DestReps, Source, SourceReps)

Moves a block or fills an array.

Syntax

Move(Dest, DestReps, Source, SourceReps)

Remarks

Block Move or fill array.

Parameter & Data Type	Enter
Dest <i>Variable or Array</i>	The first variable of an array in which to store the variables being moved.
DestReps <i>Constant</i>	The number of elements in the destination array to fill.
Source <i>Array or Expression</i>	The name of the variable array or expression that is the source of the values to move.
SourceReps <i>Constant</i>	The SourceReps parameter is the number of variables that should be moved into the Dest array. This parameter should be set equal to the DestReps parameter, or, as a special case, set to 1. If this parameter is set to 1, the same value will be placed in each variable of the Dest array. If SourceReps is less than DestReps, the remainder of the Dest array is filled with the last value from Source.

Move Function Example

The example uses the Move function.

Move (x, 20, y, 20)	<i>'move array y into array x</i>
Move (x, 20, 0.0, 1)	<i>'fill x with 0.0.</i>

MovePrecision (PrecisionVariable, X)

Moves a high precision variable into another input location.

Syntax

MovePrecision (PrecisionVariable, X)

Remarks

In this function, the variable X is moved into the PrecisionVariable as a high precision value. Every reference to the PrecisionVariable will cause a 32-bit extension of its mantissa to be saved and used internally. A normal single precision float has 24 bits of mantissa; therefore, this new precision is 56 bits.

Parameter & Data Type	Enter
PrecisionVariable <i>Variable</i>	The variable that will be affected by the precision move.
X <i>Variable</i>	The value that will be moved to the PrecisionVariable. It may or may not be a high precision variable, depending upon whether it has been declared as such in a previous AddPrecision or MovePrecision instruction.

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 <i>expr1</i> 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 <i>expr1</i> is	And <i>expr2</i> 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 <i>expr1</i> is	And bit in <i>expr2</i> 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	<i>'Declare variables.</i>
A = 10: B = 8: C = 11	<i>'Assign values.</i>
If A > B OR B > C Then	<i>'Evaluate expressions.</i>
Msg = True	
Else	
Msg = False.	
End If	

PeakValley (DestPV, DestChange, Reps, Source, Hysteresis)

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 <i>Variable or array</i>	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 <i>Variable or array</i>	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., <i>changevar</i> (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 <i>Constant</i>	The number of inputs for which to track the peaks and valleys. 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 <i>Variable or Array</i>	The variable or array containing the inputs to check for peaks and valleys.
Hysteresis <i>Constant Variable or expression</i>	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 Dim XY(2)
Public Change(3), Deg, PeakV(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)

Calculates 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 R_s/R_0 , where R_s is the RTD resistance and R_0 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 <i>Variable or Array</i>	The variable in which to store the temperature in degrees C.
Reps <i>Constant</i>	The number of repetitions for the measurement or instruction.
Source <i>Variable or Array</i>	The name of the variable or array that contains the R_s/R_0 value(s).
Mult, Offset <i>Constant, Variable, Array, or Expression</i>	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 (Number)

Initializes the random-number generator.

Syntax

Randomize (*Number*)

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 day of year from the CR3000 clock.

Syntax

RealTime(Dest)

Remarks

The destination array must be dimensioned to 9.

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)	<i>'declare as public and dimension rTime to 9</i>
Alias rTime(1) = Year	<i>'assign the alias Year to rTime(1)</i>
Alias rTime(2) = Month	<i>'assign the alias Month to rTime(2)</i>
Alias rTime(3) = Day	<i>'assign the alias Day to rTime(3)</i>
Alias rTime(4) = Hour	<i>'assign the alias Hour to rTime(4)</i>
Alias rTime(5) = Minute	<i>'assign the alias Minute to rTime(5)</i>
Alias rTime(6) = Second	<i>'assign the alias Second to rTime(6)</i>
Alias rTime(7) = uSecond	<i>'assign the alias uSecond to rTime(7)</i>
Alias rTime(8) = WeekDay	<i>'assign the alias WeekDay to rTime(8)</i>
Alias rTime(9) = Day_of_Year	<i>'assign the alias Day_of_Year to rTime(9)</i>
DataTable (VALUES, 1, 100)	<i>'set up data table</i>
DataInterval(0, 1, mSec, 0)	<i>'set up data table</i>
' Sample(1, Year, IEEE4)	<i>'place Year in VALUES table</i>
' Sample(1, Month, IEEE4)	<i>'place Month in VALUES table</i>
' Sample(1, Day, IEEE4)	<i>'place Day in VALUES table</i>
' Sample(1, Hour, IEEE4)	<i>'place Hour in VALUES table</i>
' Sample(1, Minute, IEEE4)	<i>'place Minute in VALUES table</i>
' Sample(1, Second, IEEE4)	<i>'place Second in VALUES table</i>
' Sample(1, uSecond, IEEE4)	<i>'place uSecond in VALUES table</i>
' Sample(1, WeekDay, IEEE4)	<i>'place WeekDay in VALUES table</i>
' Sample(1, Day_of_Year, IEEE4)	<i>'place Day_of_Year in VALUES table</i>
Sample(9, rTime(), IEEE4)	<i>'place all 9 segments in VALUES table</i>
EndTable	
BeginProg	
Scan (1, Sec, 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($\pm \pi$), 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 root mean square (RMS) value of an array.

Syntax

RMSSpa(Dest, Swath, Source)

Remarks

Spatial RMS, Calculate the root mean square of values in an array.

$$Dest = \sqrt{\frac{\sum_{i=j}^{i=j+swath} (X(i))^2}{swath}}$$

Where $X(j)$ = Source

Parameter & Data Type	Enter
Dest <i>Variable</i>	The variable in which to store the RMS value.
Swath <i>Constant</i>	The number of values of the array to include in the RMS calculation.
Source <i>Array</i>	The name of the variable array that is the input for the instruction.

RND

The RND function is used to generate a random number.

Syntax

Variable = RND

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. 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:

$$\text{INT}((\text{upperbound} - \text{lowerbound} + 1) * \text{RND} + \text{lowerbound})$$

Here, upperbound is the highest number in the range, and lowerbound is the lowest number in the range.

SatVP (Dest, Temp)

Calculates saturation vapor pressure over water (Svpw) in kilopascals from the air temperature (°C) and places it in the destination variable. The algorithm for obtaining Svpw from air temperature (°C) is taken from: Lowe, Paul R.: 1977, "An approximating polynomial for computation of saturation vapor pressure," *J. Appl. Meteor*, **16**, 100-103.

Saturation vapor pressure over ice (Svpi) in kilopascals for a 0°C to -50°C range can be obtained using SatVP and the relationship

$$Svpi = -.00486 + .85471 Svp + .2441 Svp^2$$

where Svpw is derived by SatVP. This relationship was derived by Campbell Scientific from the equations for the Svpw and the Svpi given in Lowe's paper.

Parameter & Data Type	Enter
Dest	Variable in which to store saturation vapor pressure (kPa).
Temp	Variable containing air temperature (°C).

Example: The following program uses the SatVP instruction to derive saturation vapor pressure, in kilopascals, from a temperature measurement in the program.

```
Dim XY(2),Polar(2),Deg,AnglDeg
Public Temp, SatVPkPa
DataTable (EnvData,1,-1)
  DataInterval (0,1,Min,10)
  Sample (1,Temp,FP2)
  Sample (1,SatVPkPa,FP2)
EndTable
BeginProg
  Scan (1,Sec,3,0)
  VoltSE(Temp,1,mV5000,2,0,0,_60Hz,0.1,-40.0)
  SatVP (SatVPkPa,Temp)
  CallTable EnvData
  NextScan
EndProg
```

SGN (Number)

Used to find the sign value of a number.

Syntax

x = **SGN** (*Number*)

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 $\text{Sgn}(X) = 0$.

If $X < 0$, then $\text{Sgn}(X) = -1$.

SGN Function Example

The example uses SGN to determine the sign of a number.

Dim Msg, Number, Volt	<i>'Declare variables.</i>
BeginProg	
Number = Volt(1)	<i>'Get user input.</i>
Select Case Sgn(Number)	<i>'Evaluate Number.</i>
Case 0	<i>'Zero.</i>
Msg = 0	
Case 1	<i>'Positive.</i>
Msg = 1	
Case -1	<i>'Negative.</i>
Msg = -1	
EndSelect	
EndProg	

SIN (Angle)

Returns the sine of an angle.

Syntax

$x = \text{SIN}(\text{Angle})$

Remarks

The *Angle* argument 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 $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

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	<i>'Declare variables.</i>
Pi = 4 * ATN(1)	<i>'Calculate Pi.</i>
Degrees = Volt(1)	<i>'Get input.</i>
Radians = Degrees * (Pi / 180)	<i>'Convert to radians.</i>
Ans = SIN (Radians)	<i>'The Sine of Degrees.</i>

SINH (NumericExpression)

The SINH function returns the hyperbolic sine of an expression or value.

Syntax

$x = \text{SINH}(\text{NumericExpression})$

Remarks

The SINH function returns the hyperbolic sine [$\text{SINH}(x) = 0.5(e^x - e^{-x})$] for the value contained in the NumericExpression 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)
    'Measure voltage on channel 1
    VoltDiff(Volt1,1,mV5000,1,True,100,500,1,0)
    Ans = SINH( Volt1 )    'The Hyperbolic Sine of Volt1
  NextScan
EndProg
```

SQR (Number)

Returns the square root of a *number*.

Syntax

$x = \text{SQR}(\text{number})$

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.
Public Volt1
BeginProg
  Scan ( 1, min, 3, 0)
    VoltDiff(Volt1,1,mV5000,1,True,100,500,1,0) 'Get input
    Number = Volt(1)
    IF Number < 0 Then
      Nsg = 0           'Cannot determine the square root of a
                        'negative number
    Else
      Msg = SQR(Number)
    NextScan
EndProg
```

StdDevSpa (Dest, Swath, Source)

Used to find the standard deviation of an array.

Syntax

StdDevSpa(Dest, Swath, Source)

Remarks

The spatial standard deviation is calculated as

$$Dest = \left(\left(\sum_{i=j}^{i=j+swath} X(i)^2 - \left(\sum_{i=j}^{i=j+swath} X(i) \right)^2 / swath \right) / swath \right)^{\frac{1}{2}}$$

Where $X(j) = \text{Source}$

Parameter & Data Type	Enter
Dest <i>Variable or Array</i>	The variable in which to store the results of the instruction.
Swath <i>Constant</i>	The number of elements to include in the calculation.
Source <i>Array</i>	The name of the first variable in the array for which the standard deviation should be calculated.

StrainCalc (Dest, Reps, Source, BrZero, BrConfig, GageFactpr, PoissonFactor)

Converts the output of a bridge measurement instruction to microstrain.

Syntax

StrainCalc (Dest, Reps, Source, BrZero, BrConfig, GageFactpr, PoissonFactor)

Remarks

Calculates microstrain, $\mu\epsilon$, 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 <i>Variable or Array</i>	Variable to store strain in.
Reps <i>Constant</i>	Number of strains to calculate, Destination, Source, and BrZero variables must be dimensioned accordingly.
Source <i>Variable</i>	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.

Parameter & Data Type	Enter														
BrZero <i>Variable</i>	The variable array that holds the unstrained reading(s) in millivolts out per volt in. This value will be subtracted from subsequent strain readings so that the strain output values will be relative to the zero condition.														
BrConfig <i>Constant</i>	<p>Bridge configuration code for strain gages The bridge configuration code can be entered as a positive or negative number: + code: $V_r = 0.001(\text{Source} - \text{Zero})$; bridge configured so its output decreases with increasing strain. - code: $V_r = -0.001(\text{Source} - \text{Zero})$; 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.)</p> <table border="1"> <thead> <tr> <th>Code</th> <th>Configuration</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>Quarter bridge strain gauge $\mu\varepsilon = \frac{-4 \cdot 10^6 V_r}{GF(1 + 2V_r)}$</td> </tr> <tr> <td>2</td> <td>Half bridge strain gauge, one gage parallel to strain, the other at 90° to strain: $\mu\varepsilon = \frac{-4 \cdot 10^6 V_r}{GF[(1 + \nu) - 2V_r(\nu - 1)]}$</td> </tr> <tr> <td>3</td> <td>Half bridge strain gauge, one gage parallel to +ε, the other parallel to -ε: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF}$</td> </tr> <tr> <td>4</td> <td>Full bridge strain gauge, 2 gages parallel to +ε, the other 2 parallel to -ε: $\mu\varepsilon = \frac{-10^6 V_r}{GF}$</td> </tr> <tr> <td>5</td> <td>Full bridge strain gauge, half the bridge has 2 gages parallel to +ε and -ε: the other half +νε and -νε: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF(\nu + 1)}$</td> </tr> <tr> <td>6</td> <td>Full bridge strain gauge, one half +ε and -νε, the other half -νε and +ε.: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF[(\nu + 1) - V_r(\nu - 1)]}$</td> </tr> </tbody> </table>	Code	Configuration	1	Quarter bridge strain gauge $\mu\varepsilon = \frac{-4 \cdot 10^6 V_r}{GF(1 + 2V_r)}$	2	Half bridge strain gauge, one gage parallel to strain, the other at 90° to strain: $\mu\varepsilon = \frac{-4 \cdot 10^6 V_r}{GF[(1 + \nu) - 2V_r(\nu - 1)]}$	3	Half bridge strain gauge, one gage parallel to +ε, the other parallel to -ε: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF}$	4	Full bridge strain gauge, 2 gages parallel to +ε, the other 2 parallel to -ε: $\mu\varepsilon = \frac{-10^6 V_r}{GF}$	5	Full bridge strain gauge, half the bridge has 2 gages parallel to +ε and -ε: the other half +νε and -νε: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF(\nu + 1)}$	6	Full bridge strain gauge, one half +ε and -νε, the other half -νε and +ε.: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF[(\nu + 1) - V_r(\nu - 1)]}$
Code	Configuration														
1	Quarter bridge strain gauge $\mu\varepsilon = \frac{-4 \cdot 10^6 V_r}{GF(1 + 2V_r)}$														
2	Half bridge strain gauge, one gage parallel to strain, the other at 90° to strain: $\mu\varepsilon = \frac{-4 \cdot 10^6 V_r}{GF[(1 + \nu) - 2V_r(\nu - 1)]}$														
3	Half bridge strain gauge, one gage parallel to +ε, the other parallel to -ε: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF}$														
4	Full bridge strain gauge, 2 gages parallel to +ε, the other 2 parallel to -ε: $\mu\varepsilon = \frac{-10^6 V_r}{GF}$														
5	Full bridge strain gauge, half the bridge has 2 gages parallel to +ε and -ε: the other half +νε and -νε: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF(\nu + 1)}$														
6	Full bridge strain gauge, one half +ε and -νε, the other half -νε and +ε.: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF[(\nu + 1) - V_r(\nu - 1)]}$														
GageFactor <i>Variable</i>	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 <i>arrayname()</i> , with no element number in the parentheses.														
PoissonFactor <i>Variable</i>	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,IIEEE4,0)
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
  CallTable ZERO_1                    'Zero_1 outputs on last scan
  Next Scan                           (Count=100)
  ZStrain = ZERO_1.ZStrain_Avg(1,1)   'Set ZStrain = averaged
  Flag(2) = True                       value
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
```

TAN (Angle)

Returns the tangent of an angle.

Syntax

x = TAN (*Angle*)

Remarks

The angle argument 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 $\pi/180$. To convert radians to degrees, multiply radians by $180/\pi$.

TAN Function Example

The example uses TAN to calculate the tangent of an angle from a Volt(1) input.

Dim Degrees, Pi, Radians, Ans	<i>'Declare variables.</i>
Pi = 4 * Atn(1)	<i>'Calculate Pi.</i>
Degrees = Volt(1)	<i>'Get user input.</i>
Radians = Degrees * (Pi / 180)	<i>'Convert to radians.</i>
Ans = TAN(Radians)	<i>'The Tangent of Degrees.</i>

TANH (Expr)

The TANH function returns the hyperbolic tangent of an expression or value.

Syntax

x = TANH (*Expr*)

Remarks

The TANH function returns the hyperbolic tangent [$\tanh(x) = \sinh(x)/\cosh(x)$] for the value defined in Expr.

TANH Function Example

The example uses TANH to calculate the hyperbolic tangent of a voltage input.

Public Volt1, Ans	<i>'Declare variables.</i>
VoltDiff(Volt1,1,mV5000,1,True,100,500,1,0)	
<i>'Returns voltage on Channel(1) to Volt(1)</i>	
Ans = TANH(Volt1)	<i>'The Hyperbolic Tangent of Volt1.</i>

TimeIntoInterval (TintoInt, Interval, Units)

The TimeIntoInterval (or IfTime) instruction is used to return a logic level of True or False based on the datalogger's real-time clock.

See also **IfTime**.

Syntax

Variable = TimeIntoInterval(TintoInt, Interval, Units)

or

If TimeIntoInterval (TintoInt, Interval, Units)

Remarks

When encountered by the datalogger program, the TimeIntoInterval statement is evaluated True (-1) or False (0) based on the datalogger's real-time clock. Time is kept internally by the datalogger as the elapsed time since January 1, 1990, at 00:00:00 hours. When the Interval divides evenly into this elapsed time, the TimeIntoInterval is set True. The TimeIntoInterval instruction can be used to set the value of a variable to -1 or 0 (first syntax example), or it can be used as an expression for a Condition (second syntax example).

The TimeIntoInterval instruction has the following parts:

TintoInt The TintoInt, or time into interval, argument allows the programmer to define an offset from the Interval at which the TimeIntoInterval statement will be evaluated true. For example, if the Interval is set at 60 minutes, and TintoInt is set to 5, TimeIntoInterval 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 TimeIntoInterval statement is True at the top of the hour.

Interval The Interval is how frequently the TimeIntoInterval statement will be evaluated True, based on the datalogger's real-time clock.

Units The Units argument is used to specify the units on which the TintoInt and Interval arguments will be based. The options are microseconds, milliseconds, seconds, minutes, hours, or days.

Notes:

TimeIntoInterval must be placed within a scan to function.

This instruction is also known as IfTime. Either keyword can be used within the program.

VaporPressure (Dest, Temp, RH)

The VaporPressure instruction calculates the ambient vapor pressure (Vp) from previously measured values for air temperature and RH. The instruction first calculates saturation vapor pressure from air temperature using Lowe's equation (see **SatVP**). Vapor pressure is then calculated by multiplying by the fractional RH:

$$V_p = \text{SatVp} \times \text{RH}/100$$

Parameter & Data Type	Enter
Dest <i>Variable or Array</i>	The variable in which to store the results of the instruction.
Temp <i>Variable</i>	The program variable that contains the value for the temperature sensor. The temperature measurement must be in degrees C.
RH <i>Variable</i>	The program variable that contains the value for the relative humidity sensor. The RH measurement must be in units of percent.

WetDryBulb (Dest, DryTemp, WetTemp, Pressure)

Calculates vapor pressure in kilopascals from the wet and dry-bulb temperatures in °C. The WetDryBulb instruction uses an equation given by List, Robert. J.: 1966, Smithsonian Meteorological Tables, Sixth Revised Edition, Smithsonian Institution, Washington, D.C., page 365. This equation is also used by the National Weather Service. The results are in kPa.

$$V_p = \text{Svp}_{\text{wet}} - A (1 + B \cdot T_w)(T_a - T_w) P$$

V_p = ambient vapor pressure in kilopascals

Svp_{wet} = saturation vapor pressure at the wet-bulb temperature in kilopascals

T_w = wet-bulb temperature, °C

T_a = ambient air temperature, °C

P = air pressure in kilopascals

A = 0.000660

B = 0.00115

Although the algorithm requires an air pressure entry, the daily fluctuations are small enough that for most applications a fixed entry of the standard pressure at the site elevation will suffice. If a pressure sensor is employed, the current pressure can be used.

Parameter & Data Type	Enter
Dest <i>Variable or Array</i>	The variable in which to store Vp (kPA).
DryTemp <i>Variable</i>	The variable containing air temperature (dry-bulb °C).
WetTemp <i>Variable</i>	The variable containing wet-bulb temperature (°C).

Parameter & Data Type	Enter
Pressure Variable or Constant	The variable containing atmospheric pressure (kPa).

XOR

Performs 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 <i>expr1</i> is	And <i>expr2</i> 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 <i>expr1</i> is	And bit in <i>expr2</i> 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	$\text{Sec} = 1 / \text{Cos}(X)$
Cosecant	$\text{Cosec} = 1 / \text{Sin}(X)$
Cotangent	$\text{Cotan} = 1 / \text{Tan}(X)$
Inverse Secant	$\text{Arcsec} = \text{Atn}(X / \text{Sqr}(X * X - 1)) + \text{Sgn}(\text{Sgn}(X) - 1) * 1.5708$
Inverse Cosecant	$\text{Arccosec} = \text{Atn}(X / \text{Sqr}(X * X - 1)) + (\text{Sgn}(X) - 1) * 1.5708$
Inverse Cotangent	$\text{Arccotan} = \text{Atn}(X) + 1.5708$
Hyperbolic Secant	$\text{HSec} = 2 / (\text{Exp}(X) + \text{Exp}(-X))$
Hyperbolic Cosecant	$\text{HCosec} = 2 / (\text{Exp}(X) - \text{Exp}(-X))$
Hyperbolic Cotangent	$\text{HCotan} = (\text{Exp}(X) + \text{Exp}(-X)) / (\text{Exp}(X) - \text{Exp}(-X))$
Inverse Hyperbolic Sine	$\text{HArcsin} = \text{Log}(X + \text{Sqr}(X * X + 1))$
Inverse Hyperbolic Cosine	$\text{HArccos} = \text{Log}(X + \text{Sqr}(X * X - 1))$
Inverse Hyperbolic Tangent	$\text{HArctan} = \text{Log}((1 + X) / (1 - X)) / 2$
Inverse Hyperbolic Secant	$\text{HArcsec} = \text{Log}((\text{Sqr}(-X * X + 1) + 1) / X)$
Inverse Hyperbolic Cosecant	$\text{HArccosec} = \text{Log}((\text{Sgn}(X) * \text{Sqr}(X * X + 1) + 1) / X)$
Inverse Hyperbolic Cotangent	$\text{HArccotan} = \text{Log}((X + 1) / (X - 1)) / 2$
Logarithm	$\text{LogN} = \text{Log}(X) / \text{Log}(N)$

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.

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(Arguments)

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. The name is limited to eight characters.
Arguments	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 CR3000 clock from the values in an array. The most likely use for this is where the CR3000 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 CR3000 clock.

Source <i>Array</i>	The source must be a seven-element array . <i>array(1)..array(7)</i> should hold respectively year, month, day, hours, minutes, seconds, and microseconds..
-------------------------------	---

Delay (Option, Delay, Units)

Used to delay the program.

Syntax

Delay(Option, 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.

Parameter & Data Type	Enter		
DelayOption Constant	Code	Result	
	0	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.	
	1	Delay will affect processing. Measurements will continue as called for by the task sequencer.	
Delay Constant	The numeric value for the time delay.		
Units Constant	The units for the delay.		
	Alpha Code	Numeric Code	Units
	USEC	0	microseconds
	MSEC	1	milliseconds
	SEC	2	seconds
	MIN	3	minutes

Do ... Loop

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 <i>condition</i> is true.
Until	Indicates that the loop is executed until <i>condition</i> is true.
<i>condition</i>	Numeric expression that evaluates true (nonzero) or false (0 or Null).
<i>statementblock</i>	Program lines between the Do and Loop statements that are repeated while or until <i>condition</i> 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.
    EndIf
```

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 ("Device: FileName", Attribute)

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 FileManager instruction has the following parameters:

Parameter & Data Type	Enter																				
Device; Filename <i>Text</i>	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. A device of USR can also be specified. The USR device is an area of memory that can be set up by the user by assigning a value to the datalogger's UsrDriveSize setting in the Status table. This drive must be set to at least 8192 bytes, in 512 byte increments (if the value entered is not a multiple of 512 bytes, the size will be rounded up).																				
Attribute Constant	The Attribute is a numeric code to determine what should happen to the file affected by the FileManager instruction. The Attribute codes are actually a bit field. The codes are as follows:																				
	<table border="1"> <thead> <tr> <th>Bit</th> <th>Decimal</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>bit 0</td> <td>1</td> <td>Program not active</td> </tr> <tr> <td>bit 1</td> <td>2</td> <td>Run on power up</td> </tr> <tr> <td>bit 2</td> <td>4</td> <td>Run now</td> </tr> <tr> <td>bits 1 & 2</td> <td>6</td> <td>Run now and on power up</td> </tr> <tr> <td>bit 3</td> <td>8</td> <td>Delete</td> </tr> <tr> <td>bit 4</td> <td>16</td> <td>Delete all</td> </tr> </tbody> </table>	Bit	Decimal	Description	bit 0	1	Program not active	bit 1	2	Run on power up	bit 2	4	Run now	bits 1 & 2	6	Run now and on power up	bit 3	8	Delete	bit 4	16
Bit	Decimal	Description																			
bit 0	1	Program not active																			
bit 1	2	Run on power up																			
bit 2	4	Run now																			
bits 1 & 2	6	Run now and on power up																			
bit 3	8	Delete																			
bit 4	16	Delete all																			

FileManager Example

The statement below uses FileManager to run TEMPS.CR3, which is stored on the datalogger's CPU, when Flag(2) becomes high.

```
If Flag(2) then FileManager( "CPU:TEMPS.CR3" 4 ) '4 means Run Now
```

FileMark (TableName)

Parameter & Data Type	Enter
TableName <i>name</i>	The name of the data table in which to insert the filemark..

Inserts a filemark into a data file. After the FileMark instruction is encountered, a filemark will be added to the next record written to the specified Table. The filemark inserted by this instruction indicates to the software processing the file that a new file should be started after the marked record. To make use of FileMarks, the DataTable must be processed using the CardConvert binary file utility.

FileMark is placed within a conditional statement in order to write the filemark at the desired time.

For ... Next

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.
<i>counter</i>	Numeric variable used as the loop counter. The variable cannot be an array element or a record element.
<i>start</i>	Initial value of <i>counter</i> .
To	Separates <i>start</i> and <i>end</i> values.
<i>end</i>	Final value of <i>counter</i> .
Step	Indicates that <i>increment</i> is explicitly stated.
<i>increment</i>	Amount <i>counter</i> is changed each time through the loop. If you do not specify Step , <i>increment</i> defaults to one.
<i>statementblock</i>	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 <i>increment</i> to be added to <i>counter</i> .

The *Step* value controls loop execution as follows:

When <i>Step</i> is	Loop executes if
Positive or 0	$counter \leq end$
Negative	$counter \geq end$

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** *thenstatements* [**Else** *elsestatements*]

Syntax 2
If *condition1* **Then**
 [*statementblock-1*]
[Elseif *condition2* **Then**
 [*statementblock-2*]
[Else
 [*statementblock-n*]
EndIf

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.
<i>condition</i>	An expression that evaluates true (nonzero) or false (0 and Null).

Then	Identifies actions to be taken if <i>condition</i> is satisfied.
<i>thenpart</i>	Statements or branches performed when <i>condition</i> is true.
Else	Identifies actions taken if <i>condition</i> is not satisfied. If the Else clause is not present, control passes to the next statement in the program.
<i>elsepart</i>	Statements or branches performed when <i>condition</i> is false.

The *thenstatements* and the *elsestatements* fields both have this syntax:

{statements |

Part	Description
-------------	--------------------

<i>statements</i>	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.
-----------	--

<i>condition1</i>	Same as <i>condition</i> used in the single-line form shown above.
-------------------	--

Then	Keyword used to identify the actions to be taken if a condition is satisfied.
-------------	---

<i>statementblock-1</i>	One or more CRBasic statements executed if <i>condition1</i> is true.
-------------------------	---

ElseIf	Keyword indicating that alternative conditions must be evaluated if <i>condition1</i> is not satisfied.
---------------	---

<i>condition2</i>	Same as <i>condition</i> used in the single-line form shown above.
-------------------	--

<i>statementblock-2</i>	One or more CRBasic statements executed if <i>condition2</i> is true.
-------------------------	---

Else	Keyword used to identify the actions taken if none of the previous conditions are satisfied.
-------------	--

<i>statementblock-n</i>	One or more CRBasic statements executed if <i>condition1</i> and <i>condition2</i> are both false.
-------------------------	--

EndIf	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 **EndIf**.

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 **EndIf** parts of the statement can have nothing but spaces in front of them. The block **If** must end with an **EndIf** statement.

For Example

```

If a > 1 And a <= 100 Then
    ...
ElseIf a = 200 Then
    ...
EndIf
    
```

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.
EndIf
. . . .                       'Run some code
. . . .                       'Run some code
    
```

RunProgram ("Device:FileName", Attrib)

Used to run a datalogger program file from the active program file.

Syntax
 RunProgram ("Device:FileName", Attrib)

Remarks
 The RunProgram 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.

A device of USR can also be specified. The USR device is an area of memory that can be set up by the user by assigning a value to the datalogger's `UsrDriveSize` setting in the Status table. This drive must be set to at least 8192 bytes, in 512 byte increments (if the value entered is not a multiple of 512 bytes, the size will be rounded up).

Attribute The Attribute is a numeric code to determine what should happen to the file called by the `RunProgram` 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

RunProgram Example

The statement below uses `RunProgram` to run `TEMPS.CR3`, which is stored on the datalogger's CPU, when `Flag(2)` becomes high.

```
If Flag(2) then RunProgram ( "CPU:TEMPS.CR3" 4 )      '4 means Run Now
```

ResetTable (TableName)

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 ... NextScan

Used to establish the program scan rate.

Syntax

Scan(Interval, Units, BufferOption, 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.

Parameter & Data Type	Enter										
Interval <i>Constant</i>	Enter the time interval at which the scan is to be executed. The interval may be in ms, s, or minutes, whichever is selected with the Units parameter. The minimum scan interval is 10 milliseconds. The maximum scan interval is 30 minutes.										
Units <i>Constant</i>	<table border="1"> <thead> <tr> <th colspan="2">The units for the time parameters.</th> </tr> <tr> <th>Alpha Code</th> <th>Units</th> </tr> </thead> <tbody> <tr> <td>MSEC</td> <td>milliseconds</td> </tr> <tr> <td>SEC</td> <td>seconds</td> </tr> <tr> <td>MIN</td> <td>minutes</td> </tr> </tbody> </table>	The units for the time parameters.		Alpha Code	Units	MSEC	milliseconds	SEC	seconds	MIN	minutes
The units for the time parameters.											
Alpha Code	Units										
MSEC	milliseconds										
SEC	seconds										
MIN	minutes										
BufferOption <i>Constant</i>	<p>Determines how data will be buffered during the Scan...NextScan process. The options are:</p> <table border="1"> <thead> <tr> <th>Option</th> <th>Result</th> </tr> </thead> <tbody> <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> </tbody> </table>	Option	Result	0, 1, or 2	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.	>3	The datalogger uses three or more buffers when processing measurements, based on the number of scans defined by this Constant.				
Option	Result										
0, 1, or 2	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.										
>3	The datalogger uses three or more buffers when processing measurements, based on the number of scans defined by this Constant.										
	<p>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.</p> <p>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</p> <p>The SlowSequence instruction does not allow for this buffering scheme even</p>										

	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.
Count <i>Integer</i>	The number of times to execute the Scan/NextScan loop. Enter 0 for infinite looping.

SelectCase ... EndSelect

Executes one of several statement blocks depending on the value of an expression.

Syntax

```

SelectCase testexpression
[Case expressionlist1
    [statementblock-1] ]
[Case expressionlist2
    [statementblock-2] ]
[CaseElse
    [statementblock-n] ]
EndSelect
    
```

The Select Case syntax has these parts:

Part	Description
SelectCase	Begins the SelectCase decision control structure. Must appear before any other part of the SelectCase structure.
<i>testexpression</i>	Any numeric or string expression. If <i>testexpression</i> matches the <i>expressionlist</i> associated with a Case clause, the <i>statementblock</i> following that Case clause is executed up to the next Case clause, or for the final one, up to the EndSelect . Control then passes to the statement following EndSelect . If <i>testexpression</i> 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 <i>expressionlist</i> matches <i>testexpression</i> .
<i>expressionlist</i>	The <i>expressionlist</i> consists of a comma-delimited list of one or more of the following forms. expression expression To expression Is compare-operator expression statementblock Elements <i>statementblock-1</i> to <i>statementblock-n</i> consist of any number of CRBasic statements on one or more lines.
CaseElse	Keyword indicating the <i>statementblock</i> to be executed if no match is found between the <i>testexpression</i> and an <i>expressionlist</i> in any of the other Case selections. When there is no CaseElse statement and no expression listed in

the **Case** clauses matches *testexpression*, program execution continues at the statement following **EndSelect**.

EndSelect Ends the **Select Case**. Must appear after all other statements in the **Select Case** control structure.

The argument expressionlist has these parts:

Part	Description
<i>expression</i>	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 **CaseElse** statement in your **SelectCase** 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

SelectCase statements can be nested. Each **SelectCase** statement must have a matching **EndSelect** statement.

SelectCase Example

The example uses **SelectCase** 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
        SelectCase 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.
        CaseElse                         'Must be something else.
            . . . .                       'Run some code.
        EndSelect
    EndIf
Else
    SelectCase 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.
    CaseElse                           'Out of range.
        . . . .                         'Run some code.
        . . . .                         'Run some code.
    EndSelect
EndIf
. . . .                                 'Run some code.
. . . .                                 'Run some code.
    
```

SetSecurity (security1, security2, security3)

Used to implement up to three levels of security in the datalogger.

The SetSecurity instruction is executed only at compile time. Up to three levels of security can be set in the datalogger. Level 1 must be set before Level 2 can be set, and Level 2 must be set before Level 3 can be set. If a level is set to 0, any level greater than it will also be set to 0 (e.g., if Level 2 is 0, Level 3 is 0). Valid security codes are 1 through 65535 (0 is no security). Each level must have a unique code.

Software access functions affected by each level of security are:

Parameter & Data Type	Enter
Security1 <i>Integer</i>	When this level is set, collecting data, setting the clock, and setting variables in the Public table are unrestricted, requiring no security code. If the user enters the Security1 code, the datalogger program can be changed or retrieved or variables can be set in the Status table.
Security2 <i>Integer</i>	When this level is set, data collection is unrestricted, requiring no security code. If the user enters the Security2 code, the datalogger clock can be changed and variables in the public table can be changed. If the user enters the Security1 code, non-read-only values in the status table can be changed and the datalogger program can be changed or retrieved.
Security3 <i>Integer</i>	When this level is set, all communication with the datalogger is prohibited if no security code is entered. If the user enters the Security3 code, data can be collected from the datalogger. If the user enters the Security2 code, data can be collected, public variables can be set, and the clock can be set. If the user enters the Security 1 code, all functions are unrestricted.

SetStatus ("FieldName", Value)

Used to change the value for a setting in the datalogger's Status table.

Syntax

SetStatus ("FieldName", Value)

Remarks

The FieldName parameter is the name of the setting to be changed; the name must be enclosed in quotes. The Value parameter is the value to which that field should be set. If the value being set is a string (such as in Messages or StationName), it must be enclosed in quotes. The following settings can be changed:

FieldName	Description
Low12VCount	An error counter indicating the number of times the 12V supply has dropped below the allowable level.
Low5VCount	An error counter indicating the number of times the 5V supply has dropped below the allowable level.

FieldName	Description
MaxProcTime	The maximum amount of time that it has taken to execute the program.
Messages	A field that can be used to hold a string value in the datalogger's Status table. The string must be enclosed in quotes.
SkippedScan	An error counter indicating the number of times a Scan has been missed because the datalogger was busy with another task (such as the previous scan).
SkippedSlowScan	An error counter indicating the number of times a SlowScan has been missed.
SkippedRecord	An error counter indicating the number of times a record was supposed to be stored but wasn't.
StationName	The name of the datalogger station.
VarOutOfBound	An indication that a variable is not dimensioned large enough to hold the values being returned.
WatchdogErrors	An error counter indicating the number of times the datalogger has had to reset its processor. Set to 0 to reset counter.

For all Status table settings except Messages and StationName, setting the value to 0 resets the error indicator. This can be useful for troubleshooting purposes.

Slow Sequence

The SlowSequence instruction is used to mark the beginning of a section of code that will run concurrently with the main program.

Syntax

SlowSequence

Remarks

The SlowSequence statement marks the end of the main program and begins a separate sequence of instructions. The instructions for the slow sequence program are executed when the main program is not running as time allows. It is possible to have up to four slow sequences executing at a rate different than that of the primary scan interval. Slow sequences can be declared with a Scan/NextScan structure, or they can be placed within a Do/Loop to execute whenever the datalogger is not busy with other tasks.

Because measurements in the main scan have priority over all other tasks, the measurement instructions in a slow sequence are performed during the times when the datalogger is not running the main scan. This can result in the measurements of a slow sequence being performed over a longer period of time than if they were placed in the main scan.

Priority of a slow sequence in the datalogger will vary, depending upon whether the datalogger is executing its program in pipeline mode or sequential mode. With the important exception of measurements, when running in pipeline mode all sequences in the program have the same priority. When running in sequential mode, the main scan has the highest priority for

measurements, followed by background calibration (which is automatically run in a slow sequence), then the first slow sequence, the second slow sequence, and so on. Refer to section OV 2.3 for additional information on the priority of sequences in the datalogger.

Slow sequences are typically run at a slower rate than the main program. They can be run at a faster rate if there are no measurement instructions in the slow sequence. There is, however, a risk of skipping scans in a slow sequence if the main scan interval is set too fast. A rule of thumb is that the main scan should be no faster than $N + 1$ * the fastest slow sequence, where N is the number of slow sequences in the program and the "1" is to account for background calibration. For example, if there are three slow sequences in the program, the main scan interval should be four times faster than the fastest slow sequence.

Subroutines and data tables called by a slow sequence should be declared after the SlowSequence instruction. Data written to data tables within a slow sequence will be time stamped with the start time of the slow sequence scan.

SlowSequence Example

```
'CR3000 Series Datalogger
'Slow Sequence Example

Public Temp107, PanelT, BattVolts

DataTable (T107,True,-1)
  DataInterval (0,1,Min,10)
  Average (1,Temp107,FP2,False)
EndTable

BeginProg
  Scan (1,Sec,10,0)
  Therm107 (Temp107,1,1,Vx1,0,250,1.0,0)
  CallTable T107
NextScan

'First Slow Sequence Scans once a minute and stores hourly average
SlowSequence
DataTable (TPanel,True,-1)
  DataInterval (0,1,Hr,10)
  Average (1,PanelT,FP2,False)
EndTable

Scan (1,Min,3,0)
  PanelTemp (PanelT,250)
  CallTable TPanel
NextScan
```

```
'Second Slow Sequence Scans every 30 minutes and stores daily average and min.
SlowSequence
DataTable (BattV,True,-1)
  DataInterval (0,1,Day,10)
  Average (1,BattVolts,FP2,False)
  Minimum (1,BattVolts,FP2,False,False)
EndTable

Scan (30,Min,3,0)
  Battery (BattVolts)
  CallTable BattV
NextScan
EndProg
```

SubScan (SubInterval, Units, Count) ... NextSubScan

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, Count)

Measurements and processing

NextSubScan

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 <i>Constant</i>	The time interval between subscans. Enter 0 for no delay between subscans.															
Units <i>Constant</i>	The unit of time for the SubInterval. <table border="1"> <thead> <tr> <th>Numeric Code</th> <th>Alpha Code</th> <th>Units</th> </tr> </thead> <tbody> <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> </tbody> </table>	Numeric Code	Alpha Code	Units	0	usec	microseconds	1	msec	milliseconds	2	sec	seconds	3	min	minutes
Numeric Code	Alpha Code	Units														
0	usec	microseconds														
1	msec	milliseconds														
2	sec	seconds														
3	min	minutes														
Count <i>Constant</i>	The number of times to loop through the subscan each time the scan runs. The maximum number is 65,535.															

Timer (TimNo, TUnits, TOption)

Used to return the value of a timer.

Remarks

Timer is a function that returns the value of a timer. **TOption** 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(TimNo, TUnits, TOption)

Parameter & Data Type	Enter		
TimNo <i>Constant, Variable, or Expression</i>	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.		
TUnits <i>Constant</i>	The units in which to return the timer value.		
	Numeric Code	Alpha Code	Units
	0	usec	microseconds
	1	msec	milliseconds
	2	sec	seconds
3	min	minutes	
4	days	days	
TOption <i>Constant</i>	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

This example uses Timer to measure the time required to execute the measurement instructions. The Timer is reset at the top of each scan. The time in microseconds is stored in the variable Elapsed.

```

SequentialMode

Public PTemp, TCTemp, Elapsed

DataTable (Test,True,-1)
  DataInterval (0,1,Min,10)
  Sample (1,TCTemp,FP2)
  Sample (1,Elapsed,IEEE4)
EndTable
    
```

```

BeginProg
  Scan (1,Sec,3,0)
    Timer (1,uSec,2)
    PanelTemp (Ptemp,250)
    TCDiff(TCTemp,1,mV20,1,TypeT,Ptemp,True,0,250,1.0,0)
    Elapsed = Timer (1,uSec,4)
    callable (Test)
  NextScan
EndProg

```

WaitDigTrig (Port, Edge)

The WaitDigTrig instruction can be executed within a Scan/NextScan sequence or it can be executed outside a scan (most commonly in a SlowSequence).

When used inside a scan, execution of the scan is paused until WaitDigTrig is true (that is, the specified control port rises or falls). When the scan starts, the time stamp will be incremented by the interval specified by the Scan instruction. This allows a clock source other than the datalogger's internal clock to control the timing of the measurements. This outside clock source could be a GPS clock or the clock of another "master" datalogger where a logic level line is connected from the clock source to one of the control ports on the datalogger.

When used outside a scan, WaitDigTrig behaves similarly to a Delay instruction, though the delay is based on the rising or falling edge of a control port instead of based on time. In this instance, the WaitDigTrig instruction can be placed inside an infinite Do/Loop, and the remaining instructions in that loop will be performed only when WaitDigTrig is true.

The priority for performing the tasks triggered by this instruction will differ depending upon whether the datalogger is executing its program in pipeline mode or sequential mode. Refer to section OV 2.3 for additional information.

Note: If the program is running in sequential mode and has a slow sequence that includes a WaitDigTrig instruction, once triggered, that sequence will not be able to perform any measurement tasks when the main scan is running. The slow sequence will pause before its first measurement instruction, until the main scan is completed, after which it will continue. If the slow sequence contains only processing tasks, these tasks can run in conjunction with the main scan.

While...Wend

The While...Wend instructions are used to execute 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 provides another 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
```


Section 10. Custom Keyboard Display Menus

CRBasic has the capability of creating a custom keyboard display menu for a datalogger program. The custom menu can either appear as submenu of the standard CR3000 menu or it can take the place of the standard menu and contain the standard menu as a submenu. An item in the custom menu may do one of four things: 1) display the value of a variable or a field in a data table. 2) display the value of a variable/flag and allow the user to change that value. 3) provide a link to another custom menu. 4) provide a link to the standard menu.

Syntax

```
DisplayMenu (MenuName, 0)  
  DisplayValue ("MenuItemName", tablename.fieldname )  
  MenuItem ("MenuItemName", Variable )  
    MenuPick (Item1, Item2, Item3...Item7 )  
  SubMenu (MenuName )  
    MenuItem ("MenuItemName", Variable )  
  EndSubMenu  
EndMenu
```

The DisplayMenu and EndMenu instructions mark the beginning and ending of a custom menu definition. Variables and stored data can be displayed as an item in a menu with the DisplayValue instruction. The MenuItem instruction creates an item that displays the value of a variable and allows the value to be edited. The MenuItem can be set up to be edited either by keying in a new numeric value or by selecting an option from a pick list. MenuPick is use to create a pick list for MenuItem. A link to another user menu can be created with the SubMenu and EndSubMenu functions.

Example:

```
'CR3000 Example for Custom Menu  
  
'Declare Variables for panel temperature, two thermocouples, a [down] counter  
'and a flag to determine if the count is active or not:  
Public Tpn1, Ttc(2)  
Public Counter, CountFlag  
  
'Declare constants for menu display:  
Const Yes = True  
Const No = False  
  
'Define DataTable Temp:  
DataTable (Temp,1,1000)  
  DataInterval (0,60,Sec,10)  
  Average (1,Tpn1,IIEEE4,0)  
  Average (2,Ttc(),IIEEE4,0)  
EndTable
```

```

'Define Custom Menu:
DisplayMenu ("Example Custom Menu",1)
  SubMenu("Current Temperatures")
    DisplayValue("Panel Temp",Tpn1)
    DisplayValue("TC 1",Ttc(1))
    DisplayValue("TC 2",Ttc(2))
  EndSubMenu
  SubMenu("Last 1 Min. Averages")
    DisplayValue("Panel Temp",Temp.Tpn1_Avg(1,1))
    DisplayValue("TC 1",Temp.Ttc_Avg(1,1))
    DisplayValue("TC 2",Temp.Ttc_Avg(2,1))
  EndSubMenu
  SubMenu ("Play with Down Count")
    MenuItem ("Enable",CountFlag)
      MenuPick (Yes,No)           'Create a pick list with constants
    MenuItem("Down Count",Counter)
      MenuPick(15,30,45,60)       'Create a pick list for Counter
    'While the counter can be reloaded with the above menu item,
    'using a sub menu allows slightly more descriptive text:
    SubMenu("Reload Down Counter")
      MenuItem("Pick Count",Counter)
        MenuPick(15,30,45,60)     'Create a pick list for Counter
      MenuItem("Enter No.",Counter) 'no pick list = user enters #
    EndSubMenu
  EndSubMenu
EndMenu

'Main Program
BeginProg
  Scan (1,Sec,3,0)
    PanelTemp (Tpn1,250)
    TCDiff (Ttc(),2,mV20C ,1,TypeT,Tpn1,True ,0,250,1.0,0)
    If CountFlag Then
      Counter=Counter-1
      If Counter <=0 Then Counter=0
    EndIf
    CallTable Temp
  NextScan
EndProg

```

DisplayMenu/EndMenu

Syntax:

```

DisplayMenu ("MenuName", AddtoSystem)
    menu definition (DisplayValue, MenuItem, and SubMenu)
EndMenu

```

The **DisplayMenu/EndMenu** instructions are used to mark the beginning and ending of a custom menu. **The DisplayValue, MenuItem, and SubMenu/EndSubMenu** instructions are used to define what will be displayed in the custom menu.

Parameter & Data Type	Enter								
MenuName <i>Text</i>	The text that will be shown as the heading for the custom menu. The string is limited to 20 characters, and it should be enclosed in quotation marks.								
AddtoSystem <i>Constant</i>	This constant determines if the custom menu is a sub menu or replaces the standard menu..								
	<table border="1"> <thead> <tr> <th>Value</th> <th>Result</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Upon power up, a starting window is displayed. When a key is pressed, the system menu will be displayed. The custom menu will appear as a submenu to the system menu</td> </tr> <tr> <td>-1</td> <td>Upon power up, a starting window is displayed. When a key is pressed, the custom menu will be displayed. The system menu will appear as a submenu to the custom menu.</td> </tr> <tr> <td>-2</td> <td>Upon power up, the custom menu is displayed. The system menu will appear as a submenu to the custom menu.</td> </tr> </tbody> </table>	Value	Result	0	Upon power up, a starting window is displayed. When a key is pressed, the system menu will be displayed. The custom menu will appear as a submenu to the system menu	-1	Upon power up, a starting window is displayed. When a key is pressed, the custom menu will be displayed. The system menu will appear as a submenu to the custom menu.	-2	Upon power up, the custom menu is displayed. The system menu will appear as a submenu to the custom menu.
	Value	Result							
	0	Upon power up, a starting window is displayed. When a key is pressed, the system menu will be displayed. The custom menu will appear as a submenu to the system menu							
-1	Upon power up, a starting window is displayed. When a key is pressed, the custom menu will be displayed. The system menu will appear as a submenu to the custom menu.								
-2	Upon power up, the custom menu is displayed. The system menu will appear as a submenu to the custom menu.								

DisplayValue ("MenuItemName", Expression)

Used to define the menu text and associated Variable or Data Table field to be displayed in the custom menu.

The MenuItemName parameter is the text that will appear on the left of the line in the custom menu. Up to 10 characters will be displayed along with the value of the source. The name should be enclosed in quotation marks. The expression must be a variable or a field from a data table. Values displayed using DisplayValue cannot be edited.

Note: DisplayValue does not allow the keyboard operator to change the value. Use MenuItem to display a variable and allow the operator to change the value.

Parameter & Data Type	Enter
MenuItemName <i>Text</i>	The text that will be shown as the heading for the custom menu. The string is limited to 20 characters, and it should be enclosed in quotation marks.
Expression <i>Variable or TableName.Field</i>	Defines the value from a data table (tablename.fieldname) or variable to be displayed, or any valid expression. When displaying a variable, the table name (public) is not included.

MenuItem ("MenuItemName", Variable)

Used to display the value of a variable and allow the user to change the value. Text can be displayed in place of a numeric value if MenuPick is used to create a pick list of constants. The constants must be defined in the program.

The MenuItemName parameter is the text that appear on the left of the line in the custom menu. The name is limited to 20 characters, but only 10 characters will be displayed when the variable value is shown (the entire 20 characters will be shown when the value is edited). MenuItemName should be enclosed in quotation marks.

The Variable parameter is the variable name of the value to be displayed. Values displayed using MenuItem can be edited, either by typing in a value directly or by creating a pick list of values using MenuPick.

Note: Use DisplayValue to display variable values without allowing them to be changed.

Parameter & Data Type	Enter
MenuItemName <i>Text</i>	The text that will be shown as the heading for the custom menu. The string is limited to 20 characters, and it should be enclosed in quotation marks.
Variable <i>Variable</i>	The source of the value to display to the right of the text "MenuItemName" The source must be a variable.

MenuPick (Item1, Item2, Item3, ..., Item7)

Used to create a pick list of values that the preceding MenuItem variable can be set to. When MenuPick is used, the pick list is the only way to set the variable from the custom menu.

The pick list can contain constants (see example). The constants must be defined in the program. When the list contains constants, the variable value shown in MenuItem will be displayed as the constant name (text) if the numeric value of the variable equals the constant.

The **MenuPick** instruction must immediately follow the **MenuItem** instruction for which a list of options is being generated. Each item in the list is separated from the next by a comma.

SubMenu/EndSubMenu

Syntax:

SubMenu ("MenuName")
menu definition (DisplayValue, MenuItem, and SubMenu)
EndSubMenu

The **SubMenu/EndSubMenu** instructions are used to define the beginning and end of a custom menu screen one level below the current menu. The MenuName parameter is the text that will be shown on the datalogger's display in the current menu and as the heading for the submenu. The string is limited to 20 characters, and it should be enclosed in quotation marks. **EndSubMenu** marks the end of the custom menu definition. The DisplayValue, MenuItem, and SubMenu instructions are used to define the submenu.

Parameter & Data Type	Enter
MenuName <i>Text</i>	The text that will be shown as the heading for the Sub menu. The string is limited to 20 characters, and it should be enclosed in quotation marks.

Section 11. String Functions

11.1 Expressions with Strings

11.1.1 Constant Strings

Fixed (constant) strings can be used in expressions using quotation marks “”. For example, `FirstName = “Mike”` causes the string variable `FirstName` to be assigned “Mike”.

11.1.2 Add Strings

Strings can be concatenated using the ‘+’ operator. For example, `FullName = FirstName + “ ” + MiddleName + “ ” + LastName` (The “ ” puts a space between the names.)

11.1.3 Subtraction of Strings

`String1-String2` results in an integer in the range of `-255..+255`. Starting with the first character in each string, the character in `string2` is subtracted from the character in `string1` until the difference is non-zero or until the end of each string is reached. This is mainly used to determine if the strings are the same or not.

11.1.4 String Conversion to/from Numeric

Conversion of Strings to Numeric and Numeric to Strings is done automatically when an assignment is made from a string to a numeric or a numeric to a string, if possible.

For example:

<code>Public Value</code>	<i>‘ default, a IEEE4 float</i>
<code>Public SensorString AS String * 8</code>	<i>‘an ASCII reading from a sensor</i>
<code>Value = SensorString * 1.8 + 32</code>	<i>‘Sensor string is converted to the IEEE4</i>
<code>Value and scaled from Celsius to Fahrenheit.</code>	

Example: Tag an ID onto the end of a list of names:

<code>Dim ID AS long</code>
<code>Public Names(10) AS STRING * 16</code>

<code>For ID = 1 to 10</code>
<code>Names(ID) = “ITEM”+ID</code>
<code>Next ID</code>

The array of `Names(10)` becomes “ITEM1”, “ITEM2”, ..., “ITEM10”

11.1.5 String Comparison Operators

The comparison operators =, >, <, <>, >= and <= operate on strings. The equality operators perform the string subtraction operation noted above and apply the appropriate rule to return either TRUE or FALSE.

Example

Find the name “Mike” in the array of Names

```
For ID = 1 to 10
  If Names(ID) = “Mike”
  ....
```

11.1.6 Sample() Type Conversions and other Output Processing Instructions

The Sample() instruction will do the necessary conversion if the source data type is different than the Sample() data type. The conversion of floats and longs to strings will allocate 12 bytes per field to hold the string.

For all other output processing instructions, except when using a reps of 1, the source data type must be the same as the data type specified in the instruction. (Only the first element of the source is converted from Long to Float if necessary. Therefore, this makes a rep of 1 legal.)

Strings are disallowed in all output processing instructions except Sample().

11.2 String Manipulation Functions

CHR(Code)

Used mainly to express non-printable ASCII characters.

Code ranges from 0..255. Note that 0 will terminate a string and therefore is useful only if this characters needs to be output.

CHR Example

Add a carriage return, line feed to a string at the end.

```
X = “Line”+Chr(13)+Chr(10)
```

FormatFloat (Float, FormatString)

Used to convert a floating point value into a string.

Syntax

String = FormatFloat (Float, FormatString)

Remarks

The string conversion of the floating point value is formatted based on the FormatString. See the CRBasic Editor help for parameter details.

Parameter & Data Type	Enter
Float <i>Variable or Constant</i>	The variable or constant that holds the floating point value to be converted.
FormatString <i>Constant</i>	Determines how the floating point value will be represented in the converted string. Note that the format string must be enclosed in quotes. The options are (m = mantissa; d = decimal; x = exponent):
%f	Decimal notation in the form of +mmm.dddddd; precision is 6 places to the right of the decimal
%e (or %E)	Decimal notation in the form of +m.dddddd e+xx; precision is 6 places to the right of the decimal
%g (or %G)	Mantissa and decimal are variable; trailing 0s and decimals are omitted
%Y.Zf	Decimal notation in the form of +m.d; precision is defined by Y places to the left of the decimal and Z places to the right of the decimal
%Ye (or %YE)	Decimal notation in the form of +m.d e+xx; precision is defined by Y characters to the right of the decimal
%Yg (or %YG)	Mantissa and decimal are variable; precision is defined by Y

FormatFloat Example

The following program shows the use of the FormatFloat instruction. Using the software's numeric monitor, enter a value into FloatVal that should be converted into a string.

```
Public FloatVal as Float
Public StringVal as String * 20

BeginProg
  Scan (1,Sec,3,0)
  StringVal = FormatFloat (FloatVal,"%f")
  NextScan
EndProg
```

InStr (Start, SearchString, SoughtString, SearchOption)

Used to find the location of a string within a string.

Syntax

Variable = InStr (Start, SearchString, SoughtString, SearchOption)

Remarks

This instruction returns the integer position of the SoughtString parameter. If the SoughtString is not found, the instruction returns 0.

This instruction returns the integer position of the FilterString parameter. If the FilterString is not found, the instruction returns 1.

Parameter & Data Type	Enter																
Start <i>Integer</i>	An integer that specifies where in the SearchString to start looking for the FilterString. A 1 indicates the first character in the string.																
SearchString <i>String or Variable</i>	The string to evaluate for the FilterString.																
FilterString <i>String or Variable</i>	The string to look for in the SearchString..																
SearchOption <i>Constant</i>	<p>A code used to help define the method of searching.</p> <table border="1"> <thead> <tr> <th>Numeric Code</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>NUMERIC - Numerics in the SearchString (FilterString is ignored)</td> </tr> <tr> <td>1</td> <td>NON-NUMERIC - Non-numerics (FilterString is ignored)</td> </tr> <tr> <td>2</td> <td>SEARCHSTRING - Each FilterString in SearchString</td> </tr> <tr> <td>3</td> <td>SEARCHCHARS - Each occurrence of any character that is in FilterString</td> </tr> <tr> <td>4</td> <td>HEADERFILTER - Strings succeeding FilterString</td> </tr> <tr> <td>6</td> <td>HEADERFILTERCHARS - Strings succeeding any character in the FilterString char list</td> </tr> <tr> <td>8</td> <td>NUMERICHEX - Hexadecimal numerics in the SearchString (FilterString is ignored)</td> </tr> </tbody> </table>	Numeric Code	Description	0	NUMERIC - Numerics in the SearchString (FilterString is ignored)	1	NON-NUMERIC - Non-numerics (FilterString is ignored)	2	SEARCHSTRING - Each FilterString in SearchString	3	SEARCHCHARS - Each occurrence of any character that is in FilterString	4	HEADERFILTER - Strings succeeding FilterString	6	HEADERFILTERCHARS - Strings succeeding any character in the FilterString char list	8	NUMERICHEX - Hexadecimal numerics in the SearchString (FilterString is ignored)
Numeric Code	Description																
0	NUMERIC - Numerics in the SearchString (FilterString is ignored)																
1	NON-NUMERIC - Non-numerics (FilterString is ignored)																
2	SEARCHSTRING - Each FilterString in SearchString																
3	SEARCHCHARS - Each occurrence of any character that is in FilterString																
4	HEADERFILTER - Strings succeeding FilterString																
6	HEADERFILTERCHARS - Strings succeeding any character in the FilterString char list																
8	NUMERICHEX - Hexadecimal numerics in the SearchString (FilterString is ignored)																

InStr Example

In the following program example, the Instr instruction is used to find the position of the text "direction:" in a string (SearchString). The result is stored in the variable Start1.

```

BeginProg
  SearchString = "maximum wind speed: 100 mph; direction: 263 degrees; Site 11 Cedar Mountain."
  FilterString1 = "direction:"
  Scan (1,Sec,0,0)
    Start1 = InStr (1,SearchString,FilterString1,2)
  NextScan
EndProg

```

LowerCase (SourceString)

Returns a lower case string of SourceString

Mid (SearchString, Start, Length)

The Mid instruction is used to return a substring that is within a string.

Syntax

String = Mid (SearchString, Start, Length)

Remarks

The Start and Length parameters are used to determine which part of the SearchString is returned. Regardless of the value of the Length parameter, the returned string will be no longer than the original string.

Parameter & Data Type	Enter
SearchString <i>String or Variable</i>	The string to evaluate for the FilterString.
Start <i>Integer</i>	An integer that specifies where in the SearchString to begin the operation. A 1 indicates the first character in the string...
Length <i>Integer</i>	An integer that specifies the maximum number of characters returned by the instruction..

SplitStr (ResultString, SearchString, FilterString, NumSplit, SplitOption)

The SplitStr instruction is used to return an array of strings or numerics from a search string.

Syntax

SplitStr (ResultString, SearchString, FilterString, NumSplit, SplitOption)

Remarks

The FilterString and SplitOption help to define the array returned by the SplitStr instruction.

Parameter & Data Type	Enter																		
ResultString <i>Variable Array</i>	An array in which the split string will be stored.																		
SearchString <i>String or Variable</i>	The string to evaluate.																		
FilterString <i>String or Variable</i>	Provides a filter for the string(s) to be returned.																		
NumSplit <i>Constant</i>	Defines the maximum number of strings or values returned by the instruction.																		
SplitOption <i>Constant</i>	A code used to specify the method of splitting the string. <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Numeric Code</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>NUMERIC - Numerics in the SearchString (FilterString is ignored)</td> </tr> <tr> <td>1</td> <td>NON-NUMERIC - Non-numerics (FilterString is ignored)</td> </tr> <tr> <td>2</td> <td>SEARCHSTRING - Each FilterString in SearchString</td> </tr> <tr> <td>3</td> <td>SEARCHCHARS - Each occurrence of any character that is in FilterString</td> </tr> <tr> <td>4</td> <td>HEADERFILTER - Strings succeeding FilterString</td> </tr> <tr> <td>6</td> <td>HEADERFILTERCHARS - Strings succeeding any character in the FilterString char list</td> </tr> <tr> <td>8</td> <td>NUMERICHEX - Hexadecimal numerics in the SearchString (FilterString is ignored)</td> </tr> <tr> <td>1x</td> <td>Right justify the resultant array, filling vacant elements with NAN (if numeric) or a NULL string if a string.</td> </tr> </tbody> </table>	Numeric Code	Description	0	NUMERIC - Numerics in the SearchString (FilterString is ignored)	1	NON-NUMERIC - Non-numerics (FilterString is ignored)	2	SEARCHSTRING - Each FilterString in SearchString	3	SEARCHCHARS - Each occurrence of any character that is in FilterString	4	HEADERFILTER - Strings succeeding FilterString	6	HEADERFILTERCHARS - Strings succeeding any character in the FilterString char list	8	NUMERICHEX - Hexadecimal numerics in the SearchString (FilterString is ignored)	1x	Right justify the resultant array, filling vacant elements with NAN (if numeric) or a NULL string if a string.
Numeric Code	Description																		
0	NUMERIC - Numerics in the SearchString (FilterString is ignored)																		
1	NON-NUMERIC - Non-numerics (FilterString is ignored)																		
2	SEARCHSTRING - Each FilterString in SearchString																		
3	SEARCHCHARS - Each occurrence of any character that is in FilterString																		
4	HEADERFILTER - Strings succeeding FilterString																		
6	HEADERFILTERCHARS - Strings succeeding any character in the FilterString char list																		
8	NUMERICHEX - Hexadecimal numerics in the SearchString (FilterString is ignored)																		
1x	Right justify the resultant array, filling vacant elements with NAN (if numeric) or a NULL string if a string.																		

SplitStr Example

In the following example, a string is split into 5 different arrays using the SplitStr instruction. The "," is used to split the search string.

```
Public SearchString As String * 50
Public ResultString(5) As String * 10

BeginProg
  Scan (1,Sec,3,0)
    SearchString="String1,String2,String3,String4,String5"
    SplitStr (ResultString(1),SearchString,",",5,5)
  NextScan
EndProg
```

StrComp (String1, String2)

The StrComp function is used to compare two strings by subtracting the characters in one string from the characters in another.

Syntax

Variable = StrComp (String1, String2)

Remarks

The StrComp instruction is typically used to determine if two strings are identical. Starting with the first character in each string, the characters in String2 are subtracted from the characters in String1 until the difference is non-zero or until the end of String2 is reached. The result of this instruction is an integer in the range of -255 to +255. If 0 is returned, the strings are identical.

UpperCase (SourceString)

Returns an upper case string of SourceString

Section 12. Serial Input and Output Functions

This set of instructions and functions are meant to be used with (non-PakBus) serial sensors and controllers and for purposes of dialing and paging through generic text based devices. They cover the functionality of the traditional Edlog dataloggers' P15 and P97 instructions with additional flexibility.

DialModem (ComPort, BaudRate, DialString, ResponseString)

Used to send a modem dial string out one of the datalogger's ports.

Syntax

DialModem (ComPort, BaudRate, DialString, ResponseString)

or

variable = DialModem (ComPort, BaudRate, DialString, ResponseString)

Remarks

The DialModem instruction performs a SerialOpen, multiple SerialOuts, and a SerialClose. If this instruction is set equal to a variable, a -1 will be returned if the ResponseString is successfully received or a 0 will be returned if it isn't.

DialModem can be used within the DialSequence/EndDialSequence commands to specify a communication route to be used for a PakBus datalogger, or it can be used within the BeginProg/EndProg statements to send the dial string any time the instruction is executed. When used within the DialSequence/EndDialSequence commands, set DialModem equal to a variable that will be used as the DialSuccess parameter for EndDialSequence. The variable will be monitored by the EndDialSequence instruction. If the call is unsuccessful, the link will be closed.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

Parameter & Data Type	Enter		
ComPort <i>Constant</i>	Specifies the communication port and mode that will be used when sending the dial string. If a negative code is entered for the ComPort (e.g., -1), the datalogger will not wait for successful communication before moving on to the next instruction. This will help to avoid skipped scans that would otherwise occur.		
	Numeric Code	Alpha Code	Description
	1	ComRS232	RS232 port of the datalogger
	2	ComME	Datalogger's CS I/O port; modem enabled
	3	Com310	Datalogger's CS I/O port; COM 310 modem
	4	ComSDC7	Datalogger's CS I/O port; SDC7
	5	ComSDC8	Datalogger's CS I/O port; SDC8
	6	Com1	Datalogger's control ports 1 (TX) & 2 (RX)
	7	Com2	Datalogger's control ports 3 (TX) & 4 (RX)
	8	Com3	Datalogger's control ports 5 (TX) & 6 (RX)
9	Com4	Datalogger's control ports 7 (TX) & 8 (RX)	

Parameter & Data Type	Enter
BaudRate	Used to set the rate, in bps, for communication. The options are 300, 1200, 4800, 9600, 19200, 38400, 57600, and 115200. Selecting one of these options fixes the baud rate at that rate of communication. If a negative baud rate is entered, the first communication attempt will be at the specified baud rate, but if communication fails at that rate, the datalogger will go into autobaud mode where it will try different rates until successful or until the instruction times out.
DialString <i>Telephone number and access codes</i>	The telephone number and any other codes used to dial the modem. A comma in the DialString inserts a 2 second pause.
ResponseString <i>String</i>	Used to specify the response code expected back from the modem when a connection is made. An entry for this parameter is required. If no response is expected, enter a null string. When a null string is entered (""), the default is "1"+CHR(13) (1 + Control M, or carriage return).

DialSequence (PakBusAddr) . . . EndDialSequence (DialSuccess)

Used to define the code necessary to route packets to a PakBus datalogger.

Syntax

DialSequence (PakBusAddr)

dialing instructions; e.g., DialSuccess = DialModem (ComPort, DialString, ResponseString)

EndDialSequence (DialSuccess)

Remarks

The DialSequence instruction indicates the beginning of the code; the EndDialSequence indicates the ending. The code is entered in the declarations section of the program, prior to the main program (defined by the BeginProg/EndProg instructions).

Any time an instruction in the main program requires that communication be made with the remote datalogger identified by the PakBusAddr parameter, the DialSequence code for that datalogger will be executed. The code will also be executed if the datalogger receives a message from another PakBus device that needs to be routed to the remote datalogger.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

Each instruction has one parameter:

Parameter & Data Type	Enter
PakBusAddr <i>Integer</i>	Identifies the PakBus address of the remote datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.
DialSuccess <i>Variable</i>	A variable that holds the results of the communications attempt. A -1 indicates communication was successful; 0 indicates a failure. If the communication attempt fails, the communication link will be closed. A variable holding the result of DialModem can be used for this parameter.

DialVoice (DialString)

Used to define the dialing string for a COM310 voice modem.

Syntax

DialVoice (DialString)

Remarks

If the DialVoice instruction is set equal to a variable, a -1 (True) will be returned if the communication attempt was successful or a 0 (False) will be returned if it failed. VoiceHangup is used after the communication attempt to hang up the voice modem.

DialString: The DialString is the telephone number and any other codes used to dial the modem. A comma in the DialString inserts a 2 second pause.

When creating code for voice modems, the VoiceKey instruction should be used to add a delay before the EndVoice instruction is executed. Otherwise, the datalogger will end the VoiceSpeak command before the spoken message is completed.

ModBusMaster (ResultCode, ComPort, BaudRate, ModBusAddr, Function, Variable, Start, Length, Tries, TimeOut)

The ModBusMaster instruction sets up a datalogger as a ModBus master device to send or retrieve data to/from a ModBus slave.

Syntax

ModBusMaster (ResultCode, ComPort, BaudRate, ModBusAddr, Function, Variable, Start, Length, Tries, TimeOut)

Remarks

The datalogger supports ModBus functions 01-05, 15, and 16 (see Function parameter below). The ModBusMaster instruction can be placed outside of the main program (defined by BeginProg/EndProg).

Parameter & Data Type	Enter		
ResultCode <i>Variable</i>	A variable that holds the results of the communication attempt. ResultCode is set to 0 if communication is successful; it increments by 1 with each failure.		
ComPort <i>Constant</i>	Specifies the communication port and mode that will be used when sending the dial string. If a negative code is entered for the ComPort (e.g., -1), the datalogger will not wait for successful communication before moving on to the next instruction. This will help to avoid skipped scans that would otherwise occur.		
	Numeric Code	Alpha Code	Description
	1	ComRS232	RS232 port of the datalogger
	2	ComME	Datalogger's CS I/O port; modem enabled
	3	Com310	Datalogger's CS I/O port; COM 310 modem
	4	ComSDC7	Datalogger's CS I/O port; SDC7
	5	ComSDC8	Datalogger's CS I/O port; SDC8
	6	Com1	Datalogger's control ports 1 (TX) & 2 (RX)
	7	Com2	Datalogger's control ports 3 (TX) & 4 (RX)
	8	Com3	Datalogger's control ports 5 (TX) & 6 (RX)
9	Com4	Datalogger's control ports 7 (TX) & 8 (RX)	
BaudRate	Used to set the rate, in bps, for communication. The options are 300, 1200, 4800, 9600, 19200, 38400, 57600, and 115200. Selecting one of these options fixes the baud rate at that rate of communication. If a negative baud rate is entered, the first communication attempt will be at the specified baud rate, but if communication fails at that rate, the datalogger will go into autobaud mode where it will try different rates until successful or until the instruction times out.		
ModBusAddr <i>Integer or Constant</i>	Specifies the address of the ModBusSlave with which you are trying to communicate. Valid ranges are 1 - 127 or 1 - 247 for ModBus/PakBus.		
ResponseString <i>String</i>	Used to specify the response code expected back from the modem when a connection is made. An entry for this parameter is required. If no response is expected, enter a null string. When a null string is entered ("") the default is "1"+CHR(13) (1 + Control M, or carriage return).		
Function <i>Integer or Constant</i>	Code	Name	Description
	1	Read Coil/Port Status	Reads the On/Off status of discrete output(s) in the ModBusSlave
	2	Read Input Status	Reads the On/Off status of discrete input(s) in the ModBusSlave
	3	Read Holding Registers	Reads the binary contents of holding register(s) in the ModBusSlave
	4	Read Input Registers	Reads the binary contents of input register(s) in the ModBusSlave
	5	Force Single Coil/Port	Forces a single Coil/Port in the ModBusSlave to either On or Off.
	15	Force Multiple Coils/Ports	Forces multiple Coils/Ports in the ModBusSlave to either On or Off.
16	Preset Multiple Registers	Presets values into a series of holding registers in the ModBusSlave.	

Parameter & Data Type	Enter
Variable Variable	The variable array that is used as the source of data to send to the ModBusSlave, or the variable array that is used as the destination for data received from the ModBusSlave.
Start Integer or constant	The first coil or register that will be acted upon with this instruction. Coil numbering starts at 1, and the coils are mapped to control ports 1 through 8. Registers are specified as the desired Start Register/2+1.
Length Integer or constant	The number of coils or registers to act upon with this instruction (e.g., if reading the status of control ports 1, 2, 3, and 4, you would specify a Start of 1 and a Length of 4).
Tries Integer or constant	The number of times the ModBusMaster datalogger should attempt to communicate with the SlaveModBus datalogger before moving on to the next instruction.
TimeOut Constant or variable	The amount of time, in 0.01 seconds, that the ModBusMaster datalogger should wait for a response from the ModBusSlave before considering the attempt a failure.

ModBusSlave (ComPort, BaudRate, ModBusAddr, ModBusVariable, BooleanVariable)

The ModBusSlave instruction sets up a datalogger as a ModBus slave device.

Syntax

ModBusSlave (ComPort, BaudRate, ModBusAddr, ModBusVariable, BooleanVariable)

Remarks

This instruction sets up a ModBus slave device to respond to the data request of a ModBus master. Supported ModBus functions are 01, 02, 03, 04, 05, 15, and 16. See the ModBusMaster for details on these functions.

Parameter & Data Type	Enter		
ComPort Constant	Specifies the communication port and mode that will be used when sending the dial string. If a negative code is entered for the ComPort (e.g., -1), the datalogger will not wait for successful communication before moving on to the next instruction.		
	Numeric Code	Alpha Code	Description
	0	ModBus /PakBus	ModBus packets routed via PakBus
	1	ComRS232	RS232 port of the datalogger
	2	ComME	Datalogger's CS I/O port; modem enabled
	3	Com310	Datalogger's CS I/O port; COM 310 modem
	4	ComSDC7	Datalogger's CS I/O port; SDC7
	5	ComSDC8	Datalogger's CS I/O port; SDC8
	6	Com1	Datalogger's control ports 1 (TX) & 2 (RX)
	7	Com2	Datalogger's control ports 3 (TX) & 4 (RX)
	8	Com3	Datalogger's control ports 5 (TX) & 6 (RX)
	9	Com4	Datalogger's control ports 7 (TX) & 8 (RX)

Parameter & Data Type	Enter
BaudRate	Used to set the rate, in bps, for communication. The options are 300, 1200, 4800, 9600, 19200, 38400, 57600, and 115200. Selecting one of these options fixes the baud rate at that rate of communication. If a negative baud rate is entered, the first communication attempt will be at the specified baud rate, but if communication fails at that rate, the datalogger will go into autobaud mode where it will try different rates until successful or until the instruction times out.
ModBusAddr <i>Integer or Constant</i>	Specifies the address of the ModBusSlave with which you are trying to communicate. Valid ranges are 1 - 127 or 1 - 247 for ModBus/PakBus.
ModBusVariable <i>Variable</i>	<p>The variable array that is used as the source of data to send to the ModBusMaster, or the variable array that is used as the destination for data received from the ModBusMaster. This variable can be formatted as either Floating Point or Integer. If Variable is declared as Long or Boolean, then this parameter is set as a ModBus integer; otherwise, floating point ModBus will be used.</p> <p>Floating point variables take two ModBus registers. The ModBus input registers are offset by 30000; ModBus holding registers are offset by 40000. Therefore, the first register corresponding to any array location X is holding register 40000+2X-1. For example, to retrieve array value number 3, you would ask for two registers starting with 40005.</p> <p>Most software uses an affixed "F" to denote that the register contains a floating point value. For example, in the Wonderware's Intouch MMI screen design you could request the value from Register "40005 F." Similarly, National Instrument's Lookout could use "Modbus1.F40005" to refer to the third input location.</p>
BooleanVariable <i>Constant or variable</i>	<p>A variable or variable array that is used to hold the result if the master sends one of the discrete on/off commands to the slave (i.e., 01 Read Coil/Port Status, 02 Read Input Status, 05 Force Single Coil/Port, 15 Force Multiple Coils/Ports). This parameter must be dimensioned as a Boolean or a compiler error is returned.</p> <p>If a 0 is entered for this parameter, then the discrete commands are mapped to control ports 1 through 8 instead.</p>

Notes:

The datalogger communicates in RTU mode (not ASCII mode) to other ModBus devices. The communications port, baud rate, data bits, stop bits, and parity are set in the ModBus driver for PC-based software or on the PLC.

The datalogger usually goes into sleep mode after 40 seconds of inactivity on the communications port. After going to sleep with some interface methods it sometimes takes a packet of incoming data to wake it up and then a retry packet to get the message through. For example, the first byte of the packet is spent waking up the SC32A/B or SC929, so a packet retry within 40 seconds is required to get a complete ModBus packet into the datalogger for processing. If packets continue arriving before the 40 second timeout, the datalogger should respond very quickly to the new packets. If necessary, you can tie pin 3 of the datalogger's CS I/O port to 5V to keep the datalogger awake. The drawback to this approach is that the average current draw will be higher than

if the datalogger is allowed to go into its low power sleep mode between infrequent ModBus queries.

Some ModBus devices (e.g., some RTUs made by Bailey Controls that use less common CPUs) require reverse word order (MSW/LSW) in the floating point format. The datalogger currently does not support this less common word order. (There are experimental versions of some datalogger operating systems that have used the reverse word order.) Some software packages have a set setting to work with this original ModBus format. For example, the “Modicon 32-bit floating point order (0123 vs. 3210)” advanced option must be enabled for the ModBus object in National Instruments’ Lookout.

ModemHangup (ComPort) ... EndModemHangup

The ModemHangup and EndModemHangup instructions are used to enclose code that should be run when a COM port hangs up communication.

Syntax

ModemHangup (ComPort)

instructions to be run upon hangup

EndModemHangup

Remarks

The ModemHangup instruction indicates the beginning of the code; the EndModemHangup indicates the ending. The code is entered in the declarations section of the program, prior to the main program (defined by the BeginProg/EndProg instructions). When the datalogger detects that a COM port is hanging up, the ModemHangup code will be run.

This instruction set is most often used with modems that must be sent a command sequence to disconnect and go into a low power state.

Note that each COM port operates independently; therefore, commands to hang up modems can be processed concurrently.

ComPort The ComPort parameter specifies the communication port and mode for this instruction.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

SerialClose (ComPort)

The SerialClose instruction is used to close a communications port that was previously opened by the SerialOpen instruction.

Syntax

SerialClose (ComPort)

Remarks

If this instruction is set equal to a variable, the result will be True (-1) if the port was opened or False (0) if it was already closed.

The ComPort parameter specifies the communication port that should be closed.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

SerialFlush (ComPort)

The SerialFlush instruction is used to clear any characters in the serial input buffer.

Syntax

SerialFlush (ComPort)

Remarks

This instruction clears the buffer and leaves the port open. If the input buffer should be cleared before each execution of SerialIn, place SerialFlush in the code before the SerialIn instruction.

The ComPort parameter specifies the communication port buffer that should be cleared.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

SerialIn (Dest, ComPort, TimeOut, TerminationChar, MaxNumChars)

The SerialIn instruction is used to set up a communications port for receiving incoming serial data.

Syntax

SerialIn (Dest, ComPort, TimeOut, TerminationChar, MaxNumChars)

Remarks

Incoming data is stored in the destination array until the TerminationChar is received, MaxNumChars value is met, or the TimeOut parameter is exceeded. Incoming characters are buffered in ring memory, the size of which is determined by the SerialOpen parameter. The buffer can be cleared using the SerialFlush instruction.

The SerialIn instruction has the following parameters:

See the CRBasic Editor help for parameter details.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

SerialInBlock (ComPort, Dest, MaxNumberBytes)

The SerialInBlock instruction stores incoming serial data. If set equal to a variable or used in place of an expression, it will return the number of bytes received.

Syntax

SerialInBlock (ComPort, Dest, MaxNumberBytes)

Remarks

Incoming serial data, up to the value defined in MaxNumberBytes, will be stored in the Dest parameter. SerialInBlock will not wait for the return of characters. If no new characters are received since the last execution of the instruction, 0 will be returned by the instruction. This instruction can be used as the expression for the NumberBytes parameter in the SerialOutBlock instruction.

ComPort The ComPort parameter specifies the communication port and mode that will be used when receiving the binary data.

See the CRBasic Editor help for parameter details.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

SerialOpen (ComPort, BaudRate, Format, TXDelay, BufferSize)

The SerialOpen instruction is used to set up one of the datalogger's ports for communication with a non-PakBus device.

Syntax

SerialOpen (ComPort, BaudRate, Format, TXDelay, BufferSize)

Remarks

When the SerialOpen instruction is executed, the serial port is "opened" and subsequent textual messages will flow in and out of the port in between PakBus packets. The data will be redirected away from the terminal mode input based on subsequent SerialIn and SerialOut instructions.

See the CRBasic Editor help for parameter details.

NOTE

SerialFlush is used to clear the buffer.

For PakBus communication, BufferSize can normally be left at 0. However, during communication with some devices it may be necessary to limit the packet size (BufferSize) and add a delay (TXDelay) for communication to be successful. For example, PakBus packets are 1000 bytes. The largest packet that an RF95 can accommodate is 248 bytes. Setting the buffer to 240 would limit the packet size and ensure that the RF95's buffer was not exceeded. A delay (e.g., 500,000 ms) would ensure that each packet has sufficient time to arrive at its destination before the next packet is transmitted.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

SerialOut (ComPort, OutString, WaitString, NumberTries, Timeout)

The SerialOut instruction is used to transmit a string over one of the datalogger's communication ports.

Syntax

SerialOut (ComPort, OutString, WaitString, NumberTries, TimeOut)

If this instruction is set equal to a variable, the number of characters output is returned. If a delay is needed before outputting the string, it should be entered in the TXDelay parameter of the SerialOpen instruction. If the OutString and WaitString variables are not formatted as a string, they will be converted to a string by the datalogger.

One of three conditions determines when the datalogger should proceed to the next instruction: when the WaitString is received, the NumberTries is exhausted, or the TimeOut is met.

See the CRBasic Editor help for parameter details.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

SerialOutBlock (ComPort, Expression, NumberBytes)

The SerialOutBlock instruction is used to send binary data out a serial port.

Syntax

SerialOutBlock (ComPort, Expression, NumberBytes)

Remarks

This instruction is needed when the data to be transmitted contains a null value. (The SerialOut instruction is terminated with a null value, thus, the transmission of binary data is required.) It can also be used when the number of bytes to be output is variable, or when the device receiving the transmitted data requires that data to be in a binary format.

See the CRBasic Editor help for parameter details.

This instruction runs sequentially from the processing task sequencer, regardless of whether the datalogger is in pipeline or sequential mode.

Section 13. PakBus Communication Instructions

This set of instructions is used to communicate with other PakBus devices. In general they specify a COM port and a PakBus address. If the route to the device is not yet known, it will try a direct route through the COM port. If the route is through a neighbor that must first be dialed, then it will first try to dial using the user specified DialSequence.

The PakBus Address parameter is a variable, so it can be used in a For/Next loop, for example.

The ComPort parameter is restricted to the following predefined constants:

1	ComRS232	CR3000 RS232 port
2	ComME	CR3000 CS I/O port; modem enabled
3	Com310	CR3000 CS I/O port; COM310 modem
4	ComSDC7	CR3000 CS I/O port; SDC7
5	ComSDC8	CR3000 CS I/O port; SDC8
6	Com1	CR3000 control ports C1 andC2
7	Com2	CR3000 control ports C3 andC4
8	Com3	CR3000 control ports C5 andC6
9	Com4	CR3000 control ports C7 andC8

This parameter sets a default com port in the case that the route to the remote node is not yet known.

The Baud Rate on asynchronous ports will default to 9600 baud unless set otherwise by a SerialOpen() function or if the port is opened by an incoming packet at some other baud rate.

The Baud Rate parameter on the asynchronous ports (ComRS232, ComME, Com1, Com2, Com3, and Com4) is restricted to 300,1200,4800,9600,19200,38400,57600,115200, with a default of 9600. (The bit rate on the synchronous ports is to be determined.)

The instructions in general include a ResultCode variable indicating whether it was successful or not. The ResultCode will be set to 0 if successful. If communication fails, it will increment. If the communication succeeds but there is some error indicated in the response code, then the ResultCode is set to the negative of the response code. The possible non-zero response codes (negated) are:

TABLE 13.1-2. ResultCode Error Codes	
1	Read Only or Permission denied
2	Out of Space in the remote
3	Syntax Error
16	Invalid Table Name or Field Name
17	Data type conversion not supported
18	Memory bounds violation
19	
20	Out of memory in the host
21	Cannot route to remote (communication not attempted)

The Timeout parameter in these instructions is in units of .01 seconds. If 0 is used, then the default timeout defined by the time of the best route is used. PakBus “Hop Metrics” are used to calculate this time.

Presently there is not a number of tries parameter. It is easy to retry within CRBasic: For I = 1 to 3: SetSettings(ResultCode,...): if ResultCode = 0 Exit For

(If needed, we can add a retry parameter or default to more than one try.)

These communication instructions will by default wait for a response or timeout before the program moves on to the next instruction. They of course can be used in a SlowSequence Scan. Optionally the ComPort parameter can be negated, which will cause the instruction not to wait for a response or timeout. This will make the instruction execute faster but any data that it retrieves and the ResultCode will be set asynchronously with the Scan, i.e., whenever the communication is complete.

ClockReport

The ClockReport instruction sends the datalogger's internal clock value to a remote datalogger in the PakBus network.

Syntax

ClockReport (ComPort, NeighborAddr, PakBusAddr)

Remarks

This instruction initiates a one-way transmission of the datalogger's clock value to a remote datalogger. No response is returned from the remote datalogger. If the remote datalogger has a PakBusClock instruction with this datalogger's address, the remote will set its clock according to the transmitted time value.

Parameter & Data Type	Enter
ComPort <i>Constant</i>	The communications port that will be used to communicate with the destination device. Enter a numeric or alphanumeric code listed in table 12.1-1.
NeighborAddr <i>Constant</i>	A static route to the destination datalogger. If 0 is entered, the destination device is assumed to be a neighbor
PakBusAddr <i>Integer(1-4094)</i>	The Pakbus address of the destination datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.

NOTE By default, LoggerNet uses PakBus address 4094 and PC400 uses 4093.

ClockReport Example

In the following example program, the host datalogger will send its clock value to a destination datalogger with PakBus address 10. If the datalogger with PakBus address 10 has a PakBusClock instruction containing this datalogger's PakBus ID, the destination datalogger's clock will be set.

Also see the PakBusClock example.

```

BeginProg
  Scan (1,Sec,3,0)
    ClockReport (ComSDC7,0,10)
  NextScan
EndProg
    
```

GetVariables (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, "TableName", "FieldName", Variable, Swath)

Used to retrieve values from a variable or variable array in a data table of a PakBus datalogger.

Syntax

GetVariables (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, "TableName", "FieldName", Variable, Swath)

Remarks

The TableName and FieldName parameters, along with the Swath parameter, are used to determine what values will be retrieved from the destination datalogger. The first value to be retrieved is defined with TableName and Fieldname, and the number of values is specified by Swath. The most recent variable(s) stored in the table are retrieved.

If security is enabled in the destination datalogger, it must be unlocked to level 3 for this instruction to be successful.

Parameter & Data Type	Enter
ResultCode <i>Variable</i>	The variable in which a response code for the transmission will be stored. A negative value indicates that communication was achieved but the command failed. See table 13.1-2 for a list of failure codes. A positive value indicates the number of communications failures. A zero indicates successful communication.
ComPort <i>Constant</i>	The communications port that will be used to communicate with the destination device. Enter a numeric or alphanumeric code listed in table 12.1-1.
NeighborAddr <i>Constant</i>	A static route to the destination datalogger. If 0 is entered, the destination device is assumed to be a neighbor
PakBusAddr <i>Integer(1-4094)</i>	The Pakbus address of the destination datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.
Security <i>Integer</i>	The security code of the datalogger to which variables will be sent. 0 is entered for this parameter if no security is set in the destination datalogger.
TimeOut <i>Constant or Variable</i>	The amount of time, in 0.01 seconds, that the datalogger should wait for a response from the destination device before considering the instruction to have failed. If 0 is entered for this parameter, then the datalogger will use a time based on its known route to the destination device.
“TableName” <i>String</i>	The data table in the destination datalogger from which the value(s) will be retrieved. TableName must be entered as a string (enclosed in quotes).
“FieldName” <i>String</i>	The name of the variable which holds the values to be retrieved from the destination datalogger. FieldName must be entered as a string (enclosed in quotes). If the variable in the destination datalogger has been assigned an Alias, the alias must be used for Fieldname.
Variable <i>Variable or Variable Array</i>	The array in which the values retrieved from the destination datalogger will be stored. The Variable parameter must be dimensioned equal to or greater than the Swath retrieved from the destination datalogger.
Swath <i>Constant or Expression</i>	The number of values that will be retrieved from the destination datalogger.

NOTE When retrieving input locations from an Edlog-programmed PakBus datalogger (CR10XPB, CR510PB, or CR23XPB), the TableName is "Inlocs" and the FieldName is the input location label.

GetVariables Example

In the following program, the datalogger will attempt to get 10 values from the destination datalogger each time Flag1 is high (non zero). The data values retrieved are from the Temp variable in the TempDat table.

```
Public RXResponse, CR10XData(10), Flag(1)

BeginProg
  Scan (1,sec,0,0)
  If Flag(1) Then
    GetVariables(RXResponse,ComSDC7,0,1,0000,0,"TempDat","Temp()",CR10XData(),10)
  EndIf
  NextScan
EndProg
```


Network (ResultCode, Reps, BeginAddr, TimeIntoInterval, Interval, Gap, GetSwath, GetVariable, SendSwath, SendVariable)

Used in conjunction with the SendGetVariables instruction to configure destination dataloggers in a PakBus network to send data to (and receive data from) the host.

Syntax

Network (ResultCode, Reps, BeginAddr, TimeIntoInterval, Interval, Gap, GetSwath, GetVariable, SendSwath, SendVariable)

Remarks

The Network instruction is used in the host datalogger; the SendGetVariables instruction is used in the destination dataloggers. The Network instruction assigns a time slot to all the destination devices during which they should respond with their data. The time slot assigned to each of the destination devices is based on the number of destination devices (Reps), the Interval/TimeIntoInterval, and the Gap, so that all destination devices respond by the time the Interval/TimeIntoInterval occurs.

Parameter & Data Type	Enter
ResultCode <i>Variable or Variable Array</i>	The variable in which a response code for the transmission will be stored. Each time the Network instruction is executed, the ResultCode for each destination device is incremented by 1. When a response is received back from a destination device, the ResultCode is set to -1. If the Reps parameter (the number of destination devices) is greater than one, then ResultCode should be dimensioned as an array large enough to accommodate a result from each of the destination devices.
Reps <i>Constant</i>	The number of destination dataloggers with which the host datalogger will be communicating.
BeginAddr <i>Integer(1-4094)</i>	The PakBus address of the first destination datalogger. Addresses for the destination devices must be in sequential order.
TimeIntoInterval <i>Constant</i>	Used along with the Interval parameter to provide a time to the destination dataloggers during which they will communicate. The TimeIntoInterval is essentially an offset for the interval parameter (e.g., if the interval is 60 seconds and the TimeIntoInterval is 15, communication will take place at 15 seconds after the top of the minute, each minute). This value is entered in seconds.
Interval <i>Constant</i>	How often the destination dataloggers should send data back to the host. This value is entered in seconds.
Gap <i>Constant</i>	Delay, in seconds, between communication from each of the destination devices. If 0 is entered, the default value, based on the host datalogger's routing table, will be used. If a negative number is entered, no delay will occur between the data transmission by each of the destination devices.
GetSwath <i>Constant or Variable</i>	The number of values that will be received from the destination datalogger. If values are being received from multiple destination devices, this is the number from each destination device.

Parameter & Data Type	Enter
GetVariables <i>Variable or Variable Array</i>	<p>The variable or variable array in which values retrieved from the destination datalogger will be stored. If multiple values are being returned by multiple destination devices, this variable must be a multi-dimensional array large enough to accommodate the values returned. For example, if 2 values are being returned by each of 4 destination devices, the variable should be dimensioned to (4, 2).</p> <p><u>Dest device #, Value = Store In</u> <u>Dest device #, Value = Store In</u></p> <p>Dest device 1, value 1 = (1, 1) Dest device 1, value 2 = (1, 2)</p> <p>Dest device 2, value 1 = (2, 1) Dest device 2, value 2 = (2, 2)</p> <p>Dest device 3, value 1 = (3, 1) Dest device 3, value 2 = (3, 2)</p> <p>Dest device 4, value 1 = (4, 1) Dest device 4, value 2 = (4, 2)</p>
SendSwath <i>Constant or Variable</i>	<p>The number of values that will be sent to the destination datalogger. If values are being sent to multiple destination devices, this is the number being sent to each destination device.</p>
SendVariables <i>Variable or Variable Array</i>	<p>The variable(s) that will be sent from this datalogger to the destination datalogger(s). If values are being sent to multiple destination dataloggers, the variable for this parameter must be a multi-dimensional array containing the values to be sent to each destination device. For instance, if you are sending 2 values to each of 3 destination devices, the variable should be dimensioned to (3, 2). The results would be:</p> <p><u>Dest device #, Value = Send From</u> <u>Dest device #, Value = Send From</u></p> <p>Dest device 1, value 1 = (1, 1) Dest device 1, value 2 = (1, 2)</p> <p>Dest device 2, value 1 = (2, 1) Dest device 2, value 2 = (2, 2)</p> <p>Dest device 3, value 1 = (3, 1) Dest device 3, value 2 = (3, 2)</p>

Network Example

This example program is written to set up two CR3000 dataloggers (addresses 109 and 110) to send and receive data via a SendGetVariables instruction in each of the destination devices. The Network instruction sends setup information to the destination dataloggers, which includes variable names and time values that are used by the TimeUntilTransmit instruction.

For the program that is run in the destination devices, see the SendGetVariables example.

```

Public PTemp, batt_volt,Result(2),Rx(2,3),Tx(2,1)

DataTable (Test,1,-1)
  DataInterval (0,15,Sec,10)
  Minimum (1,batt_volt,FP2,0,False)
  Sample (1,PTemp,FP2)
EndTable

BeginProg
  Scan (1,Sec,0,0)
  PanelTemp (PTemp,250)
  Battery (Batt_volt)
  Tx(1,1)=Tx(1,1)+1
  If Tx(1,1)=60 Then
    Tx(1,1)=0
  EndIf
  Tx(2,1)=Tx(1,1)
  Network (Result(),2,109,0,30,0,3,Rx(),1,Tx())
  CallTable Test
  NextScan
EndProg

```

NOTE

This instruction does not initiate communication with the destination dataloggers; communication is initiated by the destination devices using the SendGetVariables instruction.

PakBusClock (PakBusAddr)

Set the datalogger's clock to the clock value of a sending datalogger with the specified PakBus address.

Syntax

PakBusClock (PakBusAddr)

Remarks

The PakBusAddr parameter is the address of the remote datalogger from which this datalogger will accept a ClockReport. This must be an integer between 1 and 4094. The ClockReport instruction is used in the remote datalogger to send its clock value to this datalogger.

PakBusClock Program Example

In the following example program, if the datalogger receives a ClockReport from a PakBus datalogger with an address of 1, this datalogger's clock will be set.

Also see the ClockReport example.

```

BeginProg
  Scan (1,Sec,3,0)
  PakBusClock (1)
  NextScan
EndProg

```

Routes (Dest)

The Routes instruction returns a list of known dynamic routes for a PakBus datalogger.

Syntax

Routes (Dest)

Remarks

This instruction stores four values for each known route into the Dest parameter. The four values for each route are: ComPort used for communication, neighbor PakBus address, destination PakBus address, and expected response time (in milliseconds). The list of routes is terminated with a -1. Dest must be an array dimensioned large enough to accommodate the number of routes plus one for the termination character (4*routes + 1).

Routes Example Program

The following example shows the use of the Routes instruction. The destination variable for the PakBus routes is dimensioned to 21, which will accommodate up to 5 PakBus routes for this datalogger.

```
Public MyRoute(21)

BeginProg
  Scan (1,Sec,3,0)
    Routes (MyRoute())
  NextScan
EndProg
```

SendData (ComPort, RouterAddr, PakBusAddr, DataTable)

The SendData instruction is used to send the most recent record from a data table to a remote PakBus device.

Syntax

SendData (ComPort, RouterAddr, PakBusAddr, DataTable)

Remarks

This instruction can be used to send data to a PC running the LoggerNet server. When received, LoggerNet will store the data in a file under a name that follows its naming convention, as specified in the Setup window of the software.

Parameter & Data Type	Enter
ComPort <i>Constant</i>	The communications port that will be used to communicate with the destination device. Enter a numeric or alphanumeric code listed in table 12.1-1.
NeighborAddr <i>Constant</i>	A static route to the destination datalogger. If 0 is entered, the destination device is assumed to be a neighbor
PakBusAddr <i>Integer(1-4094)</i>	The Pakbus address of the destination datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.
DataTable <i>Name</i>	The name of the table from which the last record should be sent.

NOTE

By default, LoggerNet uses PakBus address 4094 and PC400 uses 4093.

SendData Example Program

The following example program uses the SendData instruction to send the most recent record from Table1 to PakBus ID 4094 once per minute. If the PakBus ID 4094 device is the LoggerNet server, the data will be stored in a data file on the PC.

```
Public Temp

DataTable (Table1,True,-1)
  DataInterval (0,1,Sec,10)
  Sample (1,Temp,FP2)
EndTable

BeginProg
  Scan (1,Sec,3,0)
  PanelTemp (Temp,250)
  If IfTime (0,1,Min) Then
    SendData (ComRS232,0,4094,Table1)
  EndIf
  CallTable (Table1)
NextScan
EndProg
```

SendGetVariables (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, SendVariable, SendSwath, GetVariable, GetSwath)

The SendGetVariables instruction is used in a remote datalogger to send an array of values to the host datalogger, and/or retrieve an array of data from the host datalogger.

Syntax

SendGetVariables (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, SendVariable, SendSwath, GetVariable, GetSwath)

Remarks

When the SendGetVariables instruction is used in a datalogger, data transmission times are controlled by a host datalogger. Most often, this instruction is preceded by the TimeUntilTransmit instruction to trigger the execution of the SendGetVariables instruction. The program in the host datalogger must contain the NetWork instruction, which sets the times that the remote dataloggers should respond.

If security is enabled in the host datalogger, it must be unlocked to level 2 for this instruction to be successful.

Parameter & Data Type	Enter
ResultCode <i>Variable</i>	The variable in which a response code for the transmission will be stored. A negative value indicates that communication was achieved but the command failed. See table 13.1-2 for a list of failure codes. A positive value indicates the number of communications failures. A zero indicates successful communication.
ComPort <i>Constant</i>	The communications port that will be used to communicate with the destination device. Enter a numeric or alphanumeric code listed in table 12.1-1.
NeighborAddr <i>Constant</i>	A static route to the destination datalogger. If 0 is entered, the destination device is assumed to be a neighbor
PakBusAddr <i>Integer(1-4094)</i>	The Pakbus address of the destination datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.
Security <i>Integer</i>	The security code of the datalogger to which variables will be sent. 0 is entered for this parameter if no security is set in the destination datalogger.
TimeOut <i>Constant or Variable</i>	The amount of time, in 0.01 seconds, that the datalogger should wait for a response from the destination device before considering the instruction to have failed. If 0 is entered for this parameter, then the datalogger will use a time based on its known route to the destination device.
SendVariables <i>Variable or Variable Array</i>	The variable(s) that will be sent from this datalogger to a destination datalogger.
SendSwath <i>Constant or Variable</i>	The number of variables that will be sent to the destination datalogger.
GetVariables <i>Variable or Variable Array</i>	The variable or variable array in which values retrieved from the destination datalogger will be stored. This variable must be dimensioned to the size of the GetSwath parameter.
GetSwath <i>Constant or Variable</i>	The number of variables that will be received from the destination datalogger.

SendGetVariables Program Example

In this example, a datalogger with Pakbus address 109 is programmed to send three variables to a datalogger with an address of 1 using the TimeUntilTransmit and SendGetVariables instructions. Three variables are sent to address 1 when the TimeUntilTransmit = 0. The variables are sent out the CS I/O port configured as SDC7 with MD-485 modems.

```

Public PTemp, batt_volt, RxData, TxData(3), RxResponse, counter, time

DataTable (Test,1,-1)
  DataInterval (0,15,Sec,10)
  Minimum (1,batt_volt,FP2,0,False)
  Sample (1,PTemp,FP2)
EndTable

BeginProg
  Scan (1,Sec,0,0)
    counter=counter+1
    If counter=100 Then
      counter=0
    EndIf
    PanelTemp (PTemp,250)
    Battery (Batt_volt)
    TxData(1)=PTemp
    TxData(2)=Batt_volt
    TxData(3)=counter
    time = TimeUntilTransmit
    If time = 0 Then
      SendGetVariables (RxResponse,ComSDC7,0,1,0000,0,TxData,3,RxData,1)
    EndIf
    CallTable Test
  NextScan
EndProg

```

SendTableDef (ComPort, NeighborAddr, PakBusAddr, DataTable)

The SendTableDef instruction is used to send the table definitions from a data table to a remote PakBus device.

Syntax

SendTableDef (ComPort, NeighborAddr, PakBusAddr, DataTable)

Remarks

This instruction can be used to send table definitions from a datalogger to a PC running the LoggerNet server.

Parameter & Data Type	Enter
ComPort <i>Constant</i>	The communications port that will be used to communicate with the destination device. Enter a numeric or alphanumeric code listed in table 12.1-1.
NeighborAddr <i>Constant</i>	A static route to the destination datalogger. If 0 is entered, the destination device is assumed to be a neighbor
PakBusAddr <i>Integer(1-4094)</i>	The Pakbus address of the destination datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.
DataTable <i>Name</i>	The name of the table from which the last record should be sent.

SendTableDef Example Program

The following example program sends the table definitions from Table1 to PakBus device ID 4094 every hour.

```

DataTable (Table1,True,-1)
  DataInterval (0,1,Sec,10)
  Sample (1,Temp,FP2)
EndTable

BeginProg
  Scan (1,Sec,3,0)
  PanelTemp (Temp,250)
  IF IfTime (0,1,Hr) Then
    SendTableDef (ComRS232,0,4094,Table1)
  EndIF
  CallTable (Table1)
NextScan
EndProg
    
```

SendVariables (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, "TableName", "FieldName", Variable, Swath)

The SendVariables instruction is used to send value(s) from a variable or variable array to a data table in a remote datalogger.

Syntax

SendVariables (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, "TableName", "FieldName", Variable, Swath)

Remarks

Values can only be sent to the remote datalogger's Public or Status table. The Dest and Swath parameters are used to determine what values will be sent to the remote datalogger. The first value to be sent is defined with Dest, and the number of values is specified by Swath. The most recent value(s) stored in the table are sent.

If security is enabled in the remote datalogger, it must be unlocked to level 2 for this instruction to be successful.

Parameter & Data Type	Enter
ResultCode <i>Variable</i>	The variable in which a response code for the transmission will be stored. A negative value indicates that communication was achieved but the command failed. See table 13.1-2 for a list of failure codes. A positive value indicates the number of communications failures. A zero indicates successful communication.
ComPort <i>Constant</i>	The communications port that will be used to communicate with the destination device. Enter a numeric or alphanumeric code listed in table 12.1-1.
NeighborAddr <i>Constant</i>	A static route to the destination datalogger. If 0 is entered, the destination device is assumed to be a neighbor
PakBusAddr <i>Integer(1-4094)</i>	The Pakbus address of the destination datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.

Parameter & Data Type	Enter
Security <i>Integer</i>	The security code of the datalogger to which variables will be sent. 0 is entered for this parameter if no security is set in the destination datalogger.
TimeOut <i>Constant or Variable</i>	The amount of time, in 0.01 seconds, that the datalogger should wait for a response from the destination device before considering the instruction to have failed. If 0 is entered for this parameter, then the datalogger will use a time based on its known route to the destination device.
TableName <i>Name</i>	The data table in the destination datalogger to which the value(s) will be sent. Values can be sent only to the Public (or Inlocs) or Status table. TableName must be entered as a string (enclosed in quotes).
FieldName <i>Constant or Variable</i>	The name of the variable or variable array in the destination datalogger to which data will be sent. FieldName must be entered as a string (enclosed in quotes). If the variable in the destination datalogger has been assigned an Alias, the alias must be used for Fieldname.
Variable <i>Variable or Variable Array</i>	The variable or variable array that holds the values to be sent to the destination datalogger. This variable must be dimensioned equal to or greater than the Swath of values that will be sent.
Swath <i>Constant or Variable</i>	The number of values that will be sent to the destination datalogger.

NOTES

When sending a value to an input location in an Edlog-programmed PakBus datalogger (CR10XPB, CR510PB, or CR23XPB), the TableName is "Inlocs" and the FieldName is the input location label.

If RF400 radios are being used for communication and retries are enabled, a negative value should not be used on the COMPort, and at least 2 seconds should be used for the TimeOut parameter.

SendVariables Example

In the following program, the datalogger will attempt to send 8 values to a destination datalogger with a PakBus ID of 1 each time Flag 1 is high (non zero). The data values to be sent are from the TCTemp variable.

```
Public RXResponse, TCTemp(8), RefTemp, Flag(1)

BeginProg
  Scan (1 ,sec,0,0)
  PanelTemp (RefTemp,250)
  TCDiff (TCTemp(),8,mV20C,1,TypeT,RefTemp,True ,0,250,1.0,0)
  If Flag(1) Then
    SendVariables (RXResponse,ComSDC7,0,1,0,0,"Public","Temp()",TCTemp(),8)
  EndIf
NextScan
EndProg
```

SetSettings (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, Settings)

The SetSettings instruction is used to set one or more settings in a remote datalogger.

Syntax

SetSettings (ResultCode, ComPort, NeighborAddr, PakBusAddr, Security, TimeOut, Settings)

Remarks

This instruction can be used to set one or more existing PakBus settings or user-created settings in the datalogger. If security is enabled in the remote datalogger, it must be unlocked to level 1 for this instruction to be successful.

Parameter & Data Type	Enter
ResultCode <i>Variable</i>	The variable in which a response code for the transmission will be stored. A negative value indicates that communication was achieved but the command failed. See table 13.1-2 for a list of failure codes. A positive value indicates the number of communications failures. A zero indicates successful communication.
ComPort <i>Constant</i>	The communications port that will be used to communicate with the destination device. Enter a numeric or alphanumeric code listed in table 12.1-1.
NeighborAddr <i>Constant</i>	A static route to the destination datalogger. If 0 is entered, the destination device is assumed to be a neighbor
PakBusAddr <i>Integer(1-4094)</i>	The Pakbus address of the destination datalogger with which the host datalogger is trying to communicate. Valid entries are 1 through 4094. Each PakBus device in the network must have a unique address.
Security <i>Integer</i>	The security code of the datalogger to which variables will be sent. 0 is entered for this parameter if no security is set in the destination datalogger.
TimeOut <i>Constant or Variable</i>	The amount of time, in 0.01 seconds, that the datalogger should wait for a response from the destination device before considering the instruction to have failed. If 0 is entered for this parameter, then the datalogger will use a time based on its known route to the destination device.
Settings <i>String</i>	The name of the setting to change, and the value to which it should be set. More than one setting can be defined in this parameter. Each setting is separated with a semicolon, and the entire string is enclosed in quotes. The syntax for this parameter is: "SettingName1=Value;SettingName2=Value;SettingName3=Value..."

NOTE

By default, LoggerNet uses PakBus address 4094 and PC400 uses 4093.

SetSettings Example

In the following program, when Flag(1) is set high in the datalogger, the PortStatus setting of a PakBus datalogger with ID 10 will be set True (or high).

```

Public Flag(1), SetResult

BeginProg
  Scan (1,Sec,3,0)
  If Flag(1) Then
    SetSettings (SetResult,ComSDC7,0,10,0000,0,"PortStatus(1)=True")
  EndIf
  NextScan
EndProg

```

TimeUntilTransmit

The TimeUntilTransmit instruction returns the time remaining, in seconds, before communication with the host datalogger.

Syntax

TimeUntilTransmit

Remarks

The TimeUntilTransmit value is derived from the time slot information that is sent by the host datalogger. If the host datalogger has not yet sent time slot information, this instruction will use a random time interval between 0 and 60 seconds until communication with the host is made.

A typical use of this instruction is to trigger the execution of the SendGetVariables instruction when the datalogger's communication time slot occurs (e.g., If TimeUntilTransmit = 0 Then SendGetVariables).

TimeUntilTransmit Example

In this example, a datalogger with Pakbus address 109 is programmed to send three variables to a datalogger with an address of 1 using the TimeUntilTransmit and SendGetVariables instructions. Three variables are sent to address 1 when the TimeUntilTransmit = 0. The variables are sent out the CS I/O port configured as SDC7 with MD-485 modems.

Note that the scan rate in this program must be 1 second. For the program that is run in the host datalogger (address 1), see the Network example.

```
Public PTemp, batt_volt, RxData, TxData(3), RxResponse, counter, time

DataTable (Test,1,-1)
  DataInterval (0,15,Sec,10)
  Minimum (1,batt_volt,FP2,0,False)
  Sample (1,PTemp,FP2)
EndTable

BeginProg
  Scan (1,Sec,0,0)
    counter=counter+1
    If counter=100 Then
      counter=0
    EndIf
    PanelTemp (PTemp,250)
    Battery (Batt_volt)
    TxData(1)=PTemp
    TxData(2)=Batt_volt
    TxData(3)=counter
    time = TimeUntilTransmit
    If time = 0 Then
      SendGetVariables (RxResponse,ComSDC7,0,1,0000,0,TxData,3,RxData,1)
    EndIf
    CallTable Test
  NextScan
EndProg
```

Appendix A. CR3000 Status Table

The CR3000 status table contains current system operating status information that can be accessed from the running CR3000 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.

Status Fieldname	Description	Variable Type	Default	Normal Range	User can change?	Info Type
FileMark	A value created by the PC software					
RecNum	The record number for this set of data					
TimeStamp	The time the record was generated	Time				
OSVersion	The version of the Operating System	String				Status
OSDate	Date the OS was released.	String				Status
OSSignature	The Operating System Signature	Integer				Status
SerialNumber	Machine specific serial number. Stored in FLASH memory.	Integer				Status
RevBoard	Hardware revision number. Stored in FLASH memory.	Integer				Status
StationName ¹	Name of the machine. Stored in FLASH memory.	String			Yes	Config
PakBusAddress ²	Logger PakBus address.	Integer	1	1 to 3999	Yes	Config PB
ProgName	Running program name.	String				Status
StartTime	Time that the program began running.	Time				Status
RunSignature	Signature of the current running program file.	Integer				Status
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	Integer				Status
Battery	Current value of the battery voltage. This measurement is made in the background calibration.	Float		9.6-16 Volts		Measure
PanelTemp	Current panel temperature measurement. This measurement is made in the background calibration.	Float				Measure
WatchdogErrors ³	The number of Watchdog errors that have occurred while running this program.	Integer	0	0	Can Reset = 0	Error
LithiumBattery ⁴	Current value of the Lithium battery voltage. This measurement is updated in the background calibration.	Float		2.7-3.6 Volts		Measure

Status Fieldname	Description	Variable Type	Default	Normal Range	User can change?	Info Type
Low12VCount ⁵	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.	Integer	0	0	Can Reset = 0	Error
Low5VCount	Keeps a running count of the number of occurrences of the 5VExtLow signal being asserted.	Integer	0	0	Can Reset = 0	Error
CompileResults	Contains any error messages that were generated by compilation or during run time.	String	_	0	_	Error
StartUpCode ⁶	A code variable that allows the user to know how the system woke up from poweroff.	String	0	0	_	Status / Error
ProgErrors	The number of compile (or runtime) errors for the current program.	Integer	_	0	_	Error
VarOutOfBound ⁷	Number of times an array was accessed out of bounds.	Integer	0	0	Can Reset = 0	Error
SkippedScan	Number of skipped scans that have occurred while running the current scan.	Integer	0	_	Can Reset = 0	Error
SkippedSlowScan ⁸	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.	Integer array	0	_	Can Reset = 0	Error
SkippedRecord ⁹	Variable array that tells how many records have been skipped for a given table. Each table has its own entry in this array.	Integer array	0	0	Can Reset = 0	Error
ErrorCalib ⁸	A counter that is incremented each time a bad calibration value is measured. The value is discarded (not included in the filter update) and ErrorCalib is incremented.	Integer	0	0	_	Error
MemorySize	Total amount of SRAM (bytes) in this device.	_	_	2097152 (2M) 4194304 (4M)	_	Status
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.	Integer	_	_	_	Status

Status Fieldname	Description	Variable Type	Default	Normal Range	User can change?	Info Type
ProgMemFree	Amount of free space in the CPU ramdisk that is used to store program files.	Integer		0 - 95232	—	Status
USRDriveFree	Number of bytes available in the USR Drive	Integer	—	—	—	Status
CommsMemFree	Array of two values. First value is bytes free, the second value is the number of small blocks available.	Integer array of 2	—	(1) 2000-15000	—	Status
FullMemReset	A value of 98765 written to this location will do a full memory reset. Full Memory Reset will reinitialize RAM Disk, FinalStorage, PakBus memory, and return DevConfig parameters back to defaults.	Integer	0	—	Enter 98765 to Reset	Config
DataRecordSize	Number of records in a 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.	—	—	—	—	
DataFillDays	Time in days to fill a given table. Each table has its own entry in this array.	—	—	—	—	
CardStatus	Contains a string with the most recent card status info.	String	—	—	—	Status
CardBytesFree ¹⁰	Gives the number of bytes free on the Card.	Integer	—	—	—	Status
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.	Integer	—	—	—	Status
MeasureTime	The time in microseconds 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.	Integer	—	—	—	Status
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.	Integer	—	—	—	Status
MaxProcTime	The maximum time in microseconds required to run through processing for the current scan. This value is reset when the scan exits.	Integer	—	—	Can Reset = 0	Status

Status Fieldname	Description	Variable Type	Default	Normal Range	User can change?	Info Type
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.	Integer array	—	—	—	Status
SlowProcTime ¹¹	The time in microseconds 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.	Integer array	—	—	—	Status
MaxSlowProcTime ¹²	The maximum time in microseconds 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.	Integer array	—	—	—	Status
PortStatus	Array of Boolean values giving the state of the control ports. The values are updated every 500mS.	Boolean array of 8	False	True or False	Yes	Status
PortConfig	Array of strings explaining the use of the associated control port. Valid entries are: Input, Output, SDM, SDI-12, Tx, and Rx.	String array of 8	Input	Input or Output	—	Status
Security ¹³	An array of the (3) Security settings (will not be shown if security is enabled).	Integer array of 3	0, 0, 0	0 - 65535 (0 is no security)	Yes	Status
CommsActive ¹⁴	Array of Boolean values telling if communications is currently active on the corresponding port. Aliased to CommActiveRS232, CommActiveME, CommActiveCOM310, CommActiveSDC7, CommActiveSDC8, CommActiveCOM1, CommActiveCOM2, CommActiveCOM3, CommActiveCOM4	Boolean array of 9	False, except for the active COM	True or False	—	Status
CommsConfig	Array of values telling the configuration of comm ports. Aliased to CommConfigRS232, CommConfigME, CommConfigCOM310, CommConfigSDC7, CommConfigSDC8, CommConfigCOM1, CommConfigCOM2, CommConfigCOM3, CommConfigCOM4	Integer array of 9	RS232-SDC8 = 4 COM1-4 = 0	0 or 4	—	Config

Status Fieldname	Description	Variable Type	Default	Normal Range	User can change?	Info Type
Baudrate ¹⁵	Array of baudrates for comms. Aliased to: BaudrateRS232, BaudrateME, BaudrateCOM310, BaudrateSDC7, BaudrateSDC8, BaudrateCOM1, BaudrateCOM2, BaudrateCOM3, BaudrateCOM4	Integer array of 9	RS232=1200, ME-115200, SDC8=115200, COM1-4=0	1200, 2400, 4800, 9600, 19.2k, 38.4k, 57.6k, 115.2k	Yes, can also use SerialOut instruction to setup.	Config
IsRouter	Is the CR3000 configured to act as router	Boolean	False	0 or 1	Yes	Config PB
PakBusNodes	Number of nodes (approximately) that will exist in the PakBus network. This value is used to determine how much memory to allocate for networking.	Integer	50	>=50	Yes	Config PB
CentralRouters(1) - (8) ¹⁶	Array of (8) PakBus addresses for central routers.	Integer array of 8	0	—	Yes	Config PB
Beacon (Beacon Interval)	Array of Beacon intervals (in seconds) for comms ports. Aliased to BeaconRS232, BeaconME, BeaconCOM310, BeaconSDC7, BeaconSDC8, BeaconCOM1, BeaconCOM2, BeaconCOM3, BeaconCOM4	Integer array of 9	0	0 - approx. 65,500	Yes	Config PB
Verify	Array of verify intervals (in seconds) for com ports. Aliased to VerifyRS232, VerifyME, VerifyCOM310, VerifySDC7, VerifySDC8, VerifyCOM1, VerifyCOM2, VerifyCOM3, VerifyCOM4	Integer array of 9	0	0 – approx. 65,500	—	Status
MaxPacketSize	Maximum number of bytes per data collection packet	—	1000	—	—	—
USRDriveSize	Used to set up the size of the USR Drive (a user created drive) on the CR3000.	Integer	0	8192 + increments of 512 bytes	Yes	Status
Messages	Contains a string of messages that can be entered by the user.	String	—	—	Yes	—
CalVolts	Calibration table of factory correction factors for the measurement range calibration	Float array of 15	—	—	—	Calib
CalGain ¹⁷	Calibration table of Gain values. Each integration / range combination has a gain associated with it. These numbers are updated by the background slow sequence if needed in the program.	Float array of 15	—	—	—	Calib

Status Fieldname	Description	Variable Type	Default	Normal Range	User can change?	Info Type
CalSeOffset ¹⁷	Calibration table of 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 needed in the program.	Integer array of 15	—	close to 0	—	Calib
CalDiffOffset ¹⁷	Calibration table of differential offset values. Each integration / range combination has a differential offset associated with it. These numbers are updated by the background slow sequence if needed in the program.	Integer array of 15	—	close to 0	—	Calib
IxResistor	Factory calibrated correction factor applied to resistance measurements	Float	—	Close to 1000	—	Calib
CAOffset	At compile time the offset of each CAO channel, if used, is measured and used when setting the CAO to the users programmed value.	Float	—	Close to 0	—	Calib

- 1 The StationName instruction can also be used in a program to write to this field.
- 2 Pak Bus Addresses 1 to 4094 are valid. Addresses ≥ 4000 are generally used for a PC by PC200, PC400, or LoggerNet.
- 3 Watchdog errors are automatically reset upon compiling a new program.
- 4 Replace the lithium battery if $< 2.7V$. See section 1.10.2 for replacement directions.
- 5 The 12V low comparator has some variation, but typically triggers at about 9.0 volts. The minimum specified input voltage of 9.6 V will not cause a 12 V low, but a 12 V low condition will stop the program execution before the CR3000 will give bad measurements due to low of supply voltage.
- 6 Currently not being used (12/1/2004)
- 7 The Variable out of Bounds error occurs when a program tries to write to an array variable outside of its declared size. It is a programming error that causes this, and should not be ignored. When the datalogger detects that a write outside of an array is being attempted it does not perform the write and increments the VOOB in the status table. The compiler and pre-compiler can only catch things like reps too large for an array etc. If an array is used in a loop or expression the pre-compiler does not (in most cases cannot) check to see if an array will be accessed out of bounds (i.e.

- accessing an array with a variable index such as $arr(index) = arr(index-1)$, where index is a variable).
- 8 The self calibration is run automatically in a slow scan (Section 3.8.)
 - 9 The order of tables is the order in which they are declared.
 - 10 Card bytes free is shown = -1 when no card is present.
 - 11 displays large number until a SlowScan runs.
 - 12 MaxSlowProcTime displays 0 until a SlowScan runs.
 - 13 Security can be changed via DeviceConfig, CR3000KD, PBGraph, StatusTable, and SetSecurity instruction. Shows -1 if security code has not been given/deactivated.
 - 14 When the SerialOpen instruction is used CommsConfig is loaded with the format parameter of that instruction. Currently (11/2004), the only formatting option available is 0 = No error checking. PakBus communication can occur concurrently on the same port. If the port was previously opened (in the case of the CP UARTS) for PakBus, or if the port is always opened (CS-9pin, and RS232) for PakBus the code will be 4.
 - 15 The value show is the initial baud rate the CR3000 will use. A negative value will allow the CR3000 to auto baud but will dictate at which baud rate to begin
 - 16 A list of up to 8 PB addresses for routers that can act as Central Routers. See DeviceConfig for more information.
 - 17 (1) 5000 mV range 250 uS integration,
 (2) 2500 mV range 250 uS integration,
 (3) 250 mV range 250 uS integration,
 (4) 25 mV range 250 uS integration,
 (5) 7.5 mV range 250 uS integration,
 (6) 2.5 mV range 250 uS integration,
 (7) 5000 mV range 1/60 Hz integration,
 (8) 2500 mV range 1/60 Hz integration,
 (9) 250 mV range 1/60 Hz integration,
 (10) 25 mV range 1/60 Hz integration,
 (11) 7.5 mV range 1/60 Hz integration,
 (12) 2.5 mV range 1/60 Hz integration,
 (13) 5000 mV range 1/50 Hz integration,
 (14) 2500 mV range 1/50 Hz integration,
 (15) 250 mV range 1/50 Hz integration,
 (16) 25 mV range 1/50 Hz integration,
 (17) 7.5 mV range 1/50 Hz integration,
 (18) 2.5 mV range 1/50 Hz integration

Campbell Scientific Companies

Campbell Scientific, Inc. (CSI)

815 West 1800 North
Logan, Utah 84321
UNITED STATES
www.campbellsci.com
info@campbellsci.com

Campbell Scientific Africa Pty. Ltd. (CSAf)

PO Box 2450
Somerset West 7129
SOUTH AFRICA
www.csafrica.co.za
cleroux@csafrica.co.za

Campbell Scientific Australia Pty. Ltd. (CSA)

PO Box 444
Thuringowa Central
QLD 4812 AUSTRALIA
www.campbellsci.com.au
info@campbellsci.com.au

Campbell Scientific do Brazil Ltda. (CSB)

Rua Luisa Crapsi Orsi, 15 Butantã
CEP: 005543-000 São Paulo SP BRAZIL
www.campbellsci.com.br
suporte@campbellsci.com.br

Campbell Scientific Canada Corp. (CSC)

11564 - 149th Street NW
Edmonton, Alberta T5M 1W7
CANADA
www.campbellsci.ca
dataloggers@campbellsci.ca

Campbell Scientific Ltd. (CSL)

Campbell Park
80 Hathern Road
Shepshed, Loughborough LE12 9GX
UNITED KINGDOM
www.campbellsci.co.uk
sales@campbellsci.co.uk

Campbell Scientific Ltd. (France)

Miniparc du Verger - Bat. H
1, rue de Terre Neuve - Les Ulis
91967 COURTABOEUF CEDEX
FRANCE
www.campbellsci.fr
info@campbellsci.fr

Campbell Scientific Spain, S. L.

Psg. Font 14, local 8
08013 Barcelona
SPAIN
www.campbellsci.es
info@campbellsci.es