INSTRUCTION MANUAL

CR9000 Measurement and Control System

Revision: 5/05

Warranty and Assistance

The **CR9000 MEASUREMENT AND CONTROL SYSTEM** 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.

CR9000 Table of Contents

Overview

The CR9000 is a modular, multi-processor system that provides precision measurement capabilities in a rugged, battery-operated package. The system makes measurements at a rate of up to 100 K samples/second with 16-bit resolution. Higher sample rates and resolutions can be attained using some of our higher end modules (CR9052E filter module, or the CR9058E isolation module). The CR9000 Base System includes CPU, power supply, and A/D modules. Up to nine I/O modules are inserted to configure a system for specific applications. The on-board, BASIC-like programming language includes data processing and analysis routines. PC9000 Windows™ Software provides program generation and editing, data retrieval, and realtime monitoring.

FIGURE OV1-1. CR9000 Measurement and Control System

OV1. Physical Description

OV1.1 Basic System

CR9031 CPU Module

The CR9031 CPU Module provides system control, processing, and communication to a PC via Transputer Link (TLINK) and fiber optic. The main processor is a 32-bit Inmos T805 Transputer. The module has 2MB static RAM and 2MB Flash EEPROM.

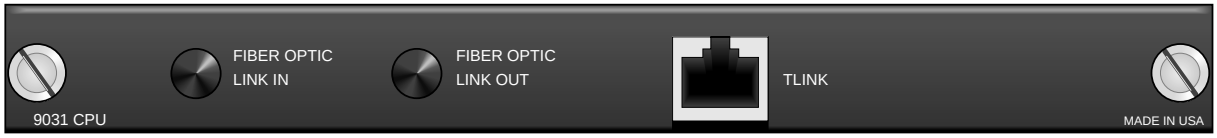


FIGURE OV1-2. CR9031

CR9041 A/D and Amplifier Module

The CR9041 A/D and Amplifier Module provides signal conditioning and 16 bit, 100 kHz A/D conversions.

FIGURE OV1-3. CR9041

CR9011 Power Supply Module and AC Adapter

The CR9011 Power Supply Module provides regulated power to the CR9000 from the internal battery modules. It also regulates battery charging from power supplied by the AC adapter, a DC input, or other external sources. The AC adapter may be used where AC power is available (100 - 240 volts) to provide power to the CR9000 and charge its batteries.

The CR9011 has a relay that allows shutting off power under program control. The Power Up inputs allow an external signal to awaken the CR9000 from a powered down state (PowerOff, Section 9). When the CR9000 is in this power off state the ON Off switch is in the on position but the internal relay is open. The power LED is not lit. If the "<0.5 " input is switched to ground or if the ">2" input has a voltage greater than 2 volts applied, the CR9000 will awake, load the program in memory and run. If the "< 0.5" input continues to be held at ground while the CR9000 is powered on and goes through its 2–5 second initialization sequence, the CR9000 will not run the program in memory.

MEASUREMENTS:

Battery (voltage and current)

CONTROL:

PowerOff

FIGURE OV1-4. CR9011

OV1.2 Measurement Modules

CR9050(E) Analog Input Module

The CR9050(E) Analog Input Module has 14 differential or 28 single-ended inputs for measuring voltages up to ±5 V. Voltages exceeding ±9 V may cause errors on other channels. An on-board PRT provides the reference temperature for thermocouple measurements, while 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 1.6 μV.

MEASUREMENTS:

Voltage
- Differential Voltage (VoltDiff)
- Single-Ended Voltage (VoltSE)

Thermocouple, Differential Voltage (TCDiff) Thermocouple, Single-Ended Voltage (TCSE)

Bridge measurements (also require CR9060 Excitation Module)
- Full Bridge (BrFull)
- 6 Wire Full Bridge (BrFull6W)
- Half Bridge (BrHalf)
- 3 Wire Half Bridge (BrHalf3W)
- 4 Wire Half Bridge (BrHalf4W)

Module Temperature (ModuleTemp)

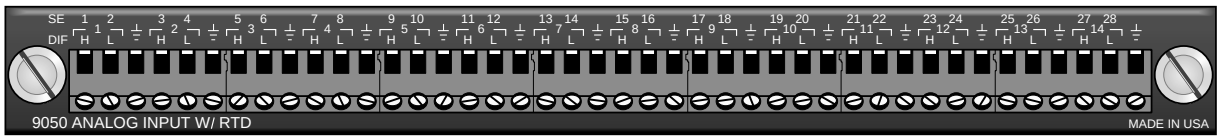


FIGURE OV1-5. CR9050

CR9051E Fault Protected 5V Analog Input Module

The number of channels and measurements are the same as for the CR9050 Analog Input Module. Each input channel is fault-protected so as to permit over-voltages between +50 V and –40 V without corruption of measurements on other input channels. All the CR9051E input channels become open switches when the CR9000 is powered off. The CR9051E is recommended

over the CR9050E for applications where fault voltages beyond ±9 V could come in contact with the inputs, or when the CR9000 could be powered off while still connected to sensors that have power applied to them.

FIGURE OV1-6. CR9051E

CR9052DC Anti-Alias Filter Module with DC Excitation

The CR9052DC is a high-performance anti-alias filter module that extends the capability of the CR9000 Measurement and Control System. The module includes six anti-aliased analog measurement channels with differential input ranges from ±20 mV to ±5 V. Each input channel has current and voltage excitation options. Measurement rates up to 50 kHz per channel are possible.

MEASUREMENTS:

VoltFilt
FFTFilt

FIGURE OV1-7. CR9052DC

CR9052IEPE Anti-Alias Filter Module with DC Excitation

The The CR9052IEPE module allows direct connection of Internal Electronics Piezo-Electric (IEPE) accelerometers and microphones to CR9000- or CR9000X-series dataloggers. Each CR9052IEPE includes six channels. Each channel has a BNC connector, an open circuit indicator LED, and a short circuit indicator LED which can indicate if the channel is over-or under-driven. Each channel has a built-in constant current source which is software programmable to 0, 2, 4, or 6 mA.

MEASUREMENTS:

VoltFilt
FFTFilt

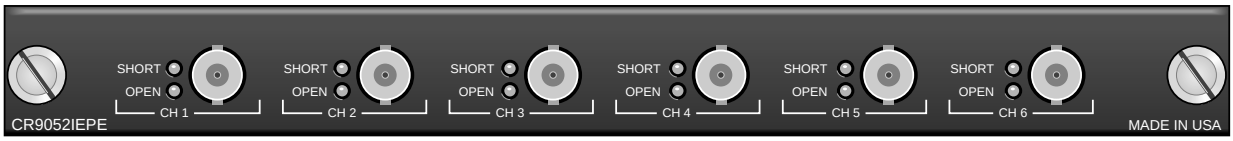


FIGURE OV1-8. CR9052IEPE

CR9058E Isolation Module

The CR9058E is a 10 channel, differential input isolation module. Each channel has a 24 bit A/D converter which supplies input isolation for up to ±60 V continuous common mode voltage conditions. The full scale ranges available are ±60 V, ±20 V, and ±2 V with a resolution to 2 μVolts. Due to its superb signal to noise ratio, and good resolution, an accurate thermocouple measurement can be made on the 2 Volt range code. An on-board programmable DSP provides digital filtering.

MEASUREMENTS:

VoltDiff
TCDiff

FIGURE OV1-9. CR9058

CR9055 50-Volt Analog Input Module

The CR9055 50-Volt Analog Input Module has 14 differential or 28 single-ended inputs for measuring voltages up to ± 50 V. Resolution on the most sensitive range is 16 μV. The CR9055 has a common mode range of ± 50 V.

MEASUREMENTS:

Voltage
  Differential Voltage (VoltDiff)
  Single-Ended Voltage (VoltSE)

Normally thermocouple measurements would be made on the CR9050 Analog Input Module (±5 Volt) because of its greater resolution, however they can be made on the CR9055 if the ±50 V common mode range is necessary.

Thermocouple, Differential Voltage (TCDiff)
Thermocouple, Single-Ended Voltage (TCDiff)

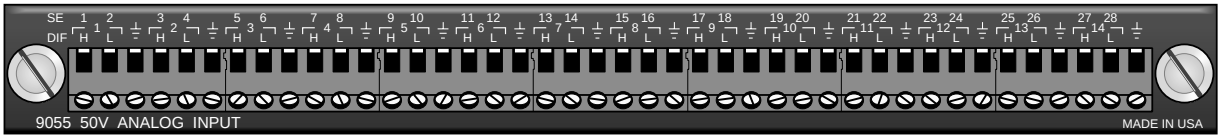


FIGURE OV1-10. CR9055

CR9060 Excitation Module

The CR9060 Excitation Module has six continuous analog outputs with individual digital-to-analog converters for PID Algorithm, waveform generation, and excitation for bridge measurements. Ten switched excitation channels provide precision voltages for bridge measurements. Each analog output will provide up to 50 mA between ±5 V. Also includes eight digital control outputs (0 V low, 5 V high).

MEASUREMENTS:

Excite
PortSet

Full Bridge (BrFull)
6 Wire Full Bridge (BrFull6w)
Half Bridge (BrHalf)
3 Wire Half Bridge (BrHalf3W)
4 Wire Half Bridge (BrHalf4W)

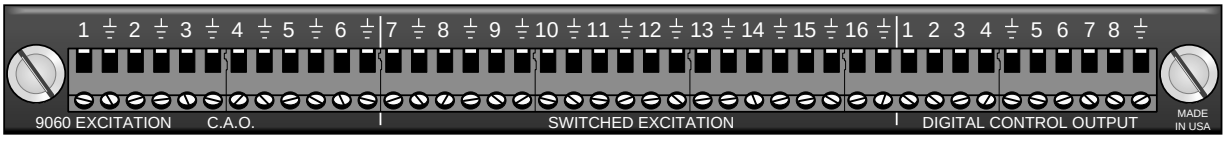


FIGURE OV1-11. CR9060

CR9070 Counter - Timer / Digital I/O Module — Obsolete

Features 12 channels capable of high-level (5 V square wave) pulse counting at frequencies up to 5 MHz. Four channels can also count switch closures; the other eight can count low-level A/C signals. In addition, there are 16 independent digital I/O channels for digital control, communications, and triggering.

MEASUREMENTS:

Count Pulses or frequency (PulseCount)
Read state of I/O Channels (ReadI/O)
Write to I/O Channels (WriteI/O)

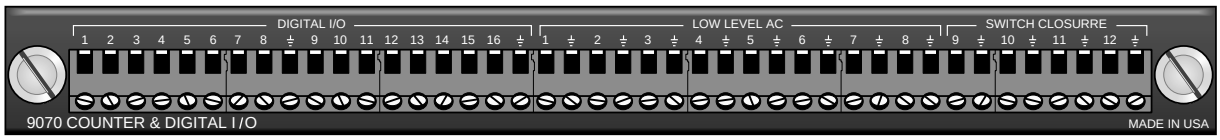


FIGURE OV1-12. CR9070

CR9071E Counter and Digital I/O Module

Features 12 channels capable of high-level (5 V square wave) pulse counting at frequencies up to 1 MHz. The pulse channels can also do interval timing measurements with 40 ηs resolution. Four channels can also count switch closures; the other eight can count low-level A/C signals. In addition, there are 16 independent digital I/O channels for digital control, communications, and triggering.

MEASUREMENTS:

Count Pulses or frequency (PulseCount)
Read state of I/O Channels (ReadI/O)
Write to I/O Channels (WriteI/O)
Interval and Timing Measurements (TimerI/O)

FIGURE OV1-13. CR9071E

Data Storage Peripheral and Memory Module

Contains slots for two type I/II PCMCIA cards or one type III PCMCIA card. A 9-pin serial I/O port supports CSI peripherals. The LEDs indicate the status of the cards in slots A and B. **Not lit:** no card detected, **green:** present and correctly formatted, **red:** present but corrupt or unrecognized, **orange:** accessing the card. Press the button next to the status LED to power down a card before removing it. The LED will blink green several times then go out for ten seconds. Remove the card while the LED is not lit. The card will be reactivated if not removed.

CAUTION Removing a card while it is active can cause garbled data and can actually damage the card. Do not switch off the power (CR9011 Module) while the cards are present and active.

MEASUREMENTS:

Output data to PAM (PAMOut)
DSP4 Display (DSP4)
CSAT3 Sonic Anemometer (CSAT)

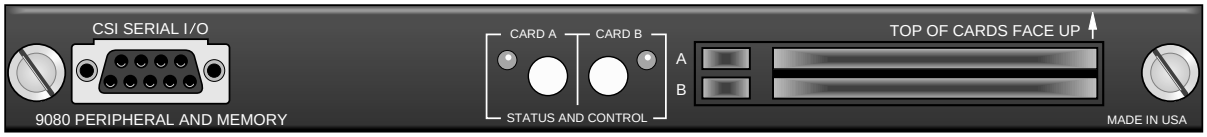


FIGURE OV1-14. CR9080

OV1.3 Communication Interfaces

TL925 RS232-TLINK Interface

The TL925 CR9000 to Computer Interface converts RS232 signals from the computer into a transputer link for the CR9000. The TLINK cable can be up to 30 meters long.

FIGURE OV1-15. TL925

BLC100 Bus Link Card and Fiber Optic Link Interface — Obsolete

The BLC100 is an interface board that plugs into a half-length card slot (AT bus) in the user's computer. It can be used for either TLINK (8 wire, up to 30 meters) or for fiber optic (separate transmit and receive) communications. Communication rates up to 10 MBPS can be attained.

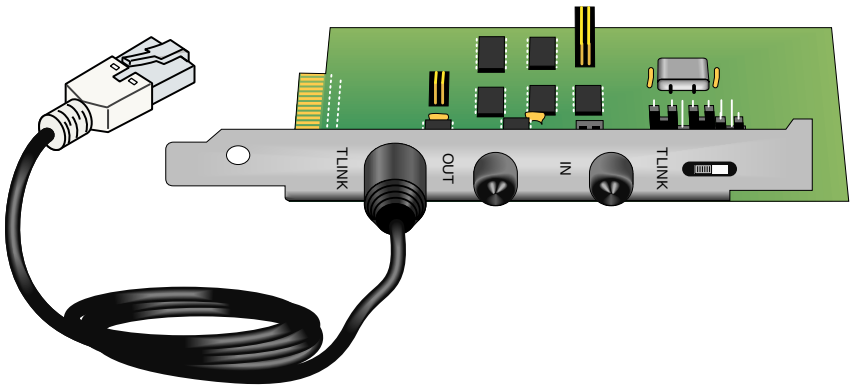


FIGURE OV1-16. BLC100 Bus Link Card

PLA100-L Parallel Link Interface

The PLA100-L converts a parallel port on a computer to a TLINK for communication with the CR9000.

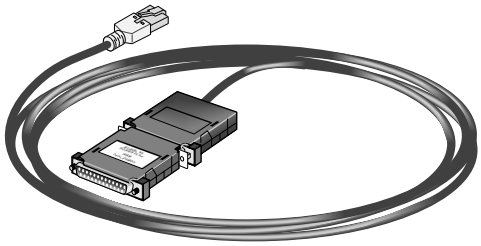


FIGURE 0V1-17. PLA100-L Parallel Link Interface

OV2. Memory and Programming Concepts

OV2.1 Memory

The CR9031 CPU Module in the CR9000 base system has 2MB static RAM and 2MB Flash EEPROM. The static RAM allows fast read write cycles (150 ηs). The Flash EEPROM is much slower to write to (15 μs *minimum*) but it retains its information when power is shut off. The operating system and user program listing(s) are stored in the flash EEPROM. When the CR9000 is powered up, the operating system and the compiled program are loaded into RAM. The memory that is not used by the operating system and program is available for data storage. The size of available memory may be seen in the status file. Additional data storage is available with the 9080 PAM Module.

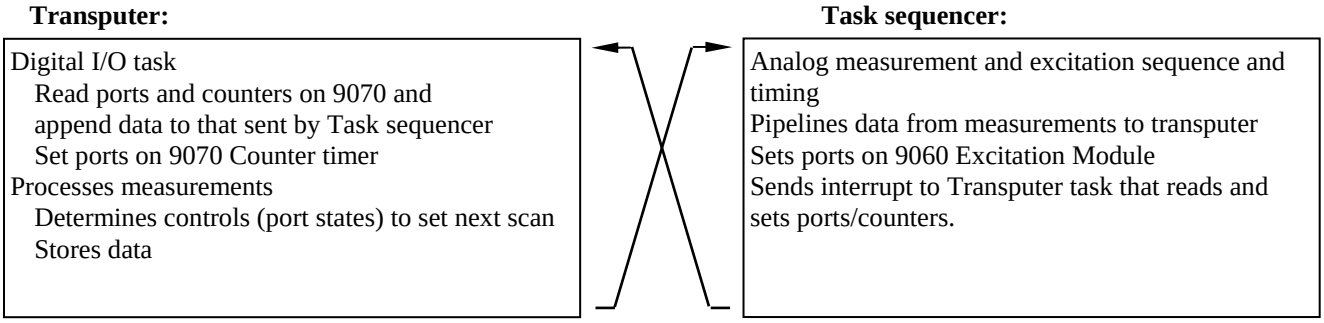
OV2.2 Measurements, Processing, Data Storage

The CR9000 divides datalogging and control between two entities. The **task sequencer** manipulates the measurement and control hardware on a rigidly timed sequence. The main processor, an Inmos T805 **Transputer,** processes and stores the resulting measurements and makes the decisions to actuate controls.

The **Transputer** is a 32-bit processor that has parallel processing capabilities. Four communication links allow rapid transfer of data with little processor time. One link is used to transfer data to and from intelligent modules in the CR9000 (e.g., the PAM module). One link is used for TLINK communications and another for the fiber optic link. The forth link gives the task sequencer direct memory access (DMA) to store raw Analog to Digital Converter (ADC) data directly into transputer memory. As soon as the data from a scan is in memory, the transputer starts processing it. There are two buffers allocated for this raw ADC data, thus the transputer can be busy full time processing one scan of data while the task sequencer is filling the other.

The transputer directly controls the 9070/CR9071E Counter and Digital I/O Module.

The **task sequencer** is a combination of components that include memory, a Xylinx Programmable Gate Array (i.e., a CSI customized chip), and the digital bus. When a program is compiled by the transputer, it loads the task sequencer memory with a series of instructions that define the sequence and timing of the measurements. This control includes channel and gain switching and ADC control that is done in our other dataloggers by the CPU. When the program runs, the task sequencer steps through the instructions at a precise rate, ensuring that the measurement timing is exact and invariant.



OV2.3 Data Tables

The CR9000 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 the program generator or when writing a datalogger program directly.

Table OV2-1. Typical Data Table

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

OV3. PC9000 Application Software

PC9000 is a Windows™ application for use with the CR9000. The software supports CR9000 program generation, real-time display of datalogger measurements, graphing, and retrieval of data files.

OV3.1 Hardware and Software Requirements

The following computer resources are recommended:

- IBM PC, Portable or Desktop
- 64 Meg of Ram
- VGA Monitor
- Windows 2000, Windows XP, Windows NT, or Windows 4.0
- 60 Meg of Hard Drive Space for software
- 400 Meg of Hard Drive Space for data
- Parallel port and a PLA100-L, RS232 Serial Port and TL925, or BLC100 Bus Link Card

The following computer resources are recommended:

- 128 Meg of Ram
- 233 MHz 486 or faster
- Mouse

OV3.2 PC9000 Installation

To install the PC9000 Software:

- Start Microsoft Windows 2000, NT, or XP
- Insert diskette 1 (marked 1 of 2) in a disk drive.
- From the Program Manager, select **F**ile menu and choose **R**un
- Type (disk drive):\setup and press Enter e.g. a:\setup<Enter>
- The setup routine will prompt for disk 2.

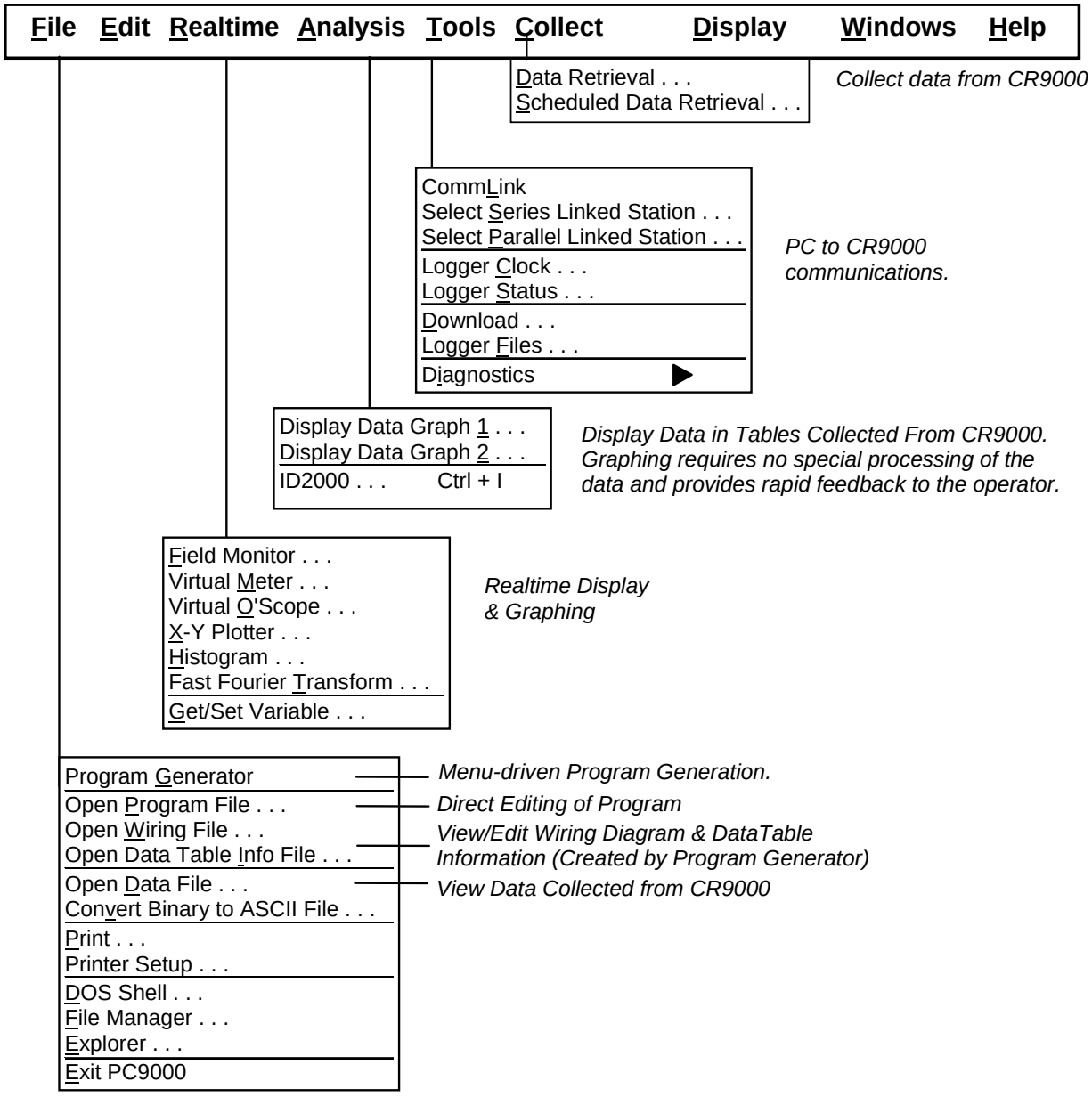
You may use the default directory of PC9000 or install the software in a different directory. The directory will be created for you.

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

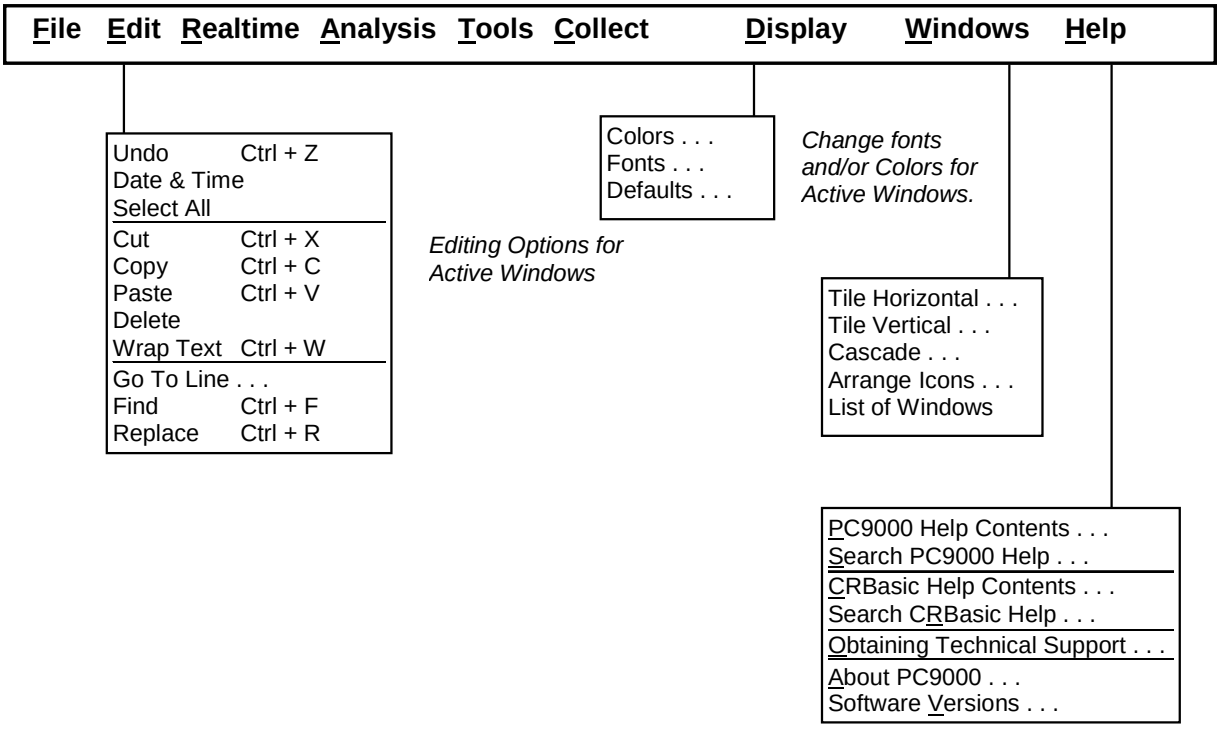
OV3.3 PC9000 Software Overview

This overview points out the main PC9000 functions and where to find them. PC9000 has extensive on-line help to guide the user in its operation. Install PC9000 to get the details. A CR9000 is not necessary to try out the programming and real time display options; the demo uses canned data for viewing. Without a CR9000, there are no communications with the datalogger; operations such as downloading programs and retrieving data will not function.

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



OV3-1. PC9000 Primary Functions



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

File

Program Generator
Guides the user through a series of menus to configure the measurement types: thermocouple, voltage, bridge, pulse counting, frequency, and others. Creates a CR9000 program, wiring diagram, output table, description, and configuration file.

Program Editor
Create programs directly or edit those created by the program generator or retrieved from the CR9000. Provides context-sensitive help for the CR9000's BASIC-like language.

Edit

REALTIME

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

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

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

Analysis

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

TOOLS

Control and Communications
Supports PC to CR9000 communications: clock read/set, status read, program download, and program retrieval.

COLLECT

Collect data from CR9000 data tables

DISPLAY

Configure the font and color scheme in an active window.

WINDOWS

Size and arrange windows.

HELP

On-line help for PC9000 software.

OV4. Specifications

General CR9000 & CR9000C Specifications

Electrical specifications are valid over a -25° to +50°C range unless otherwise specified; testing over -40° to +70°C available as an option, excluding batteries. Non-condensing environment is required. To maintain specifications, Campbell Scientific recommends recalibrating dataloggers every two years.

9031 CPU MODULE

PROCESSORS: Main CPU is 32-bit with on-chip floating point unit. Measurements, timing, and setup done by hardware task sequencer with DMA type transfer to CPU memory.

MEMORY: 2 MB Flash EEPROM, 2 MB Static RAM

9011 POWER SUPPLY MODULE

VOLTAGE: 9.6 to 18 Vdc

TYPICAL CURRENT DRAIN: Base system with no modules is 500 mA active; 300 mA standby. Current drain of individual I/O modules varies. Refer to specifications for each I/O module for specific values. Power supply module can place the system in standby mode by shutting off power to the rest of the modules.

DC CHARGING: 9.6 to 18 Vdc input charges internal batteries at up to 2 A rate. Charging circuit includes temperature compensation.

INTERNAL BATTERIES: Sealed rechargeable with 14 Ahr (7 Ahr for the CR9000C) capacity per charge.

EXTERNAL BATTERIES: External 12 V batteries can be connected.

9041 A/D and AMPLIFIER MODULE

A/D Conversions: 16-bit, 100 kHz

PC9000(C) INTERFACES

PLA100

TYPICAL CURRENT DRAIN: 50 mA, supplied by the CR9000(C)

SIZE (excluding cable): 2.25" x 0.5" x 4.0" (5.7 x 1.3 x 10.2 cm)

CABLE LENGTH: Specified, in feet, by the user, 50 ft maximum length

WEIGHT: 2.5 lb (0.11 kg)

TL925

TYPICAL CURRENT DRAIN: 50 mA, supplied by the CR9000(C)

BAUD RATE: 300 bps to 115.2 kbps with auto baud detection.

SIZE: 2.1" x 1.0" x 6.8" (5.3 x 2.5 x 17.3 cm)

WEIGHT: 2.5 lb (0.11 kg)

TRANSIENT PROTECTION

All analog and digital inputs and outputs use gas discharge tubes and transient filters to protect against high-voltage transients. Digital I/Os also have over-voltage protection clamping.

PHYSICAL SPECIFICATIONS

Size

Lab Enclosure:	15.75"L x 9.75"W x 8"D (40 x 24.8 x 20.3 cm)
Fiberglass Enclosure:	18"L x 13.5"W x 9"D (45.7 x 34.3 x 22.9 cm)
CR9000C:	10"L x 11"W X 9"D (25.4 x 27.9 x 22.9 cm)

Weight

Lab Enclosure: 30 lbs including modules (13.6 kg)

Fiberglass Enclosure: 42 lbs including modules (19.1 kg)

CR9000C: 27 lbs including modules (12.3 kg)

Replacement Batteries: 6.4 lbs (2.9 kg)

Additional Modules: 1 lb each (0.5 kg)

WARRANTY

Three years against defects in materials and workmanship.

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

CR9000 & CR9000C I/O Module Specifications

CR9050(E) and CR9051E ANALOG INPUT MODULE with RTD

INPUT CHANNELS PER MODULE: 14 differential or 28 single-ended.

RANGE AND RESOLUTION:

Input Range (mV)	Resolution (1 A/D count) (µV)	Input Noise (µV RMS)	Max Sample Rates (kHz)
±5000	158.0	105	100
±1000	32.0	35	100
±200	6.3	7	50
±50	1.6	4	50

Input Range (mV)	Input Noise (µV RMS) CR9050(E)	CR9051E
±5000	105	130
±1000	35	35
±200	7	7
±50	4	4

Note: Measurement averaging provides lower noise and better resolution.

ACCURACY OF VOLTAGE MEASUREMENTS:

Single-Ended & Differential:
±(0.07% of reading + 4 A/D counts) -25° to +50°C
±(0.14% of reading + 4 A/D counts) -40° to +70°C

Dual Differential:
(two measurements with input polarity reversed)
±(0.07% of reading + 1 A/D count) -25° to +50°C
±(0.14% of reading + 1 A/D count) -40° to +70°C

COMMON MODE RANGE: ±5 V

DC COMMON MODE REJECTION: >120 dB

INPUT RESISTANCE: 2.5 gigaohms typical

MAXIMUM INPUT VOLTAGE WITHOUT DAMAGE: ±20 V CR9050(E), -40 to +50 V CR9051E

TYPICAL CURRENT DRAIN: 25 mA active

Resistance & Conductivity Measurements

(Also requires 9060 Excitation Module)

ACCURACY: ± (0.04% of reading + 2 A/D counts) limited by accuracy of external bridge resistors.

MEASUREMENT TYPES: 6-wire and 4-wire full bridge, 4-wire, 3-wire, and 2-wire half bridge. Uses excitation reversal to remove thermal EMF errors.

CR9052 ANTI-ALIAS FILTER MODULE

INPUT CHANNELS PER MODULE: six differential

CONTINUOUS EXCITATION CHANNELS PER MODULE: 12 (6 current, 6 voltage)

TYPICAL CURRENT DRAIN: 400 mA + 1.5*[I_{ex}] where I_{ex} is the sum of excitation currents provided by all channels.

Refer to the CR9052 product literature for a complete listing of specifications.

CR9055(E) 50 V-ANALOG INPUT MODULE

INPUT CHANNELS PER MODULE: 14 differential or 28 single-ended.

RANGE AND RESOLUTION:

Input Range (V)	Resolution (1 A/D count) (µV)	Input Noise (µV RMS)	Max Sample Rates (kHz)
±50	1580	1050	100
±10	320	350	100
±2	63	85	50
±0.5	16	60	50

Note: Measurement averaging provides lower noise and better resolution.

ACCURACY OF VOLTAGE MEASUREMENTS:

Single-Ended & Differential:
±(0.1% of reading + 4 A/D counts) -25° to +50°C
±(0.2% of reading + 4 A/D counts) -40° to +70°C

Dual Differential:
(two measurements with input polarity reversed)
±(0.1% of reading + 1 A/D count) -25° to +50°C
±(0.2% of reading + 1 A/D counts) -40° to +70°C

COMMON MODE RANGE: ±50 V

DC COMMON MODE REJECTION: >62 dB

INPUT RESISTANCE: 100 Kohms typical

MAXIMUM INPUT VOLTAGE WITHOUT DAMAGE: ±150 V

TYPICAL CURRENT DRAIN: 15 mA active

CR9058E ISOLATION MODULE

INPUT CHANNELS PER MODULE: 10 isolated, differential; each channel has its own isolation ground for shielded cable connection.

RANGE, RESOLUTION, AND INPUT RESISTANCE:

Input Range (Vdc)	Resolution w/o Averaging (µV)	Resolution w/ Averaging (µV)	Input Resistance (Kohms)
±2	±10	±2	10,000
±20	±100	±20	88.9
±60	±300	±60	269

ACCURACY: ±0.02% of Full Scale Range over -40° to +70°C

MINIMUM SCAN TIME PER MODULE:
VoltDiff: 1285 µs (778 samples per second) + integration time for no input reversal (RevDiff=0); or 2990 µs (334 samples per second) + integration time with input reversal (RevDiff=1)

TCDiff (range parameter set to V2C): 2570 µs (389 samples per second) + integration time for no input reversal (RevDiff=0); or 4275 µs (233 samples per second) + integration time with input reversal (RevDiff=1).

MAXIMUM CONTINUOUS VOLTAGE W/O DAMAGE:

Input Range (Vdc)	H to L (Vdc)	H or L to ISO Ground (Vdc)	ISO Ground to Systm Ground (Vdc)	H or L to Systm Ground (Vdc)
±2	±208	±109	±360	±469
±20	±223	±121	±360	±481
±60	±448	±233	±360	±593

MAXIMUM ESD VOLTAGE ON INPUTS: ±5000V

CR9060 EXCITATION MODULE

TYPICAL CURRENT DRAIN:
108 mA quiescent, 125 mA active

Analog Outputs

ANALOG OUTPUTS PER MODULE:
10 switched, 6 continuous

SWITCHED: Provides excitation for resistance measurements. Only one output can be active at a time.

CONTINUOUS: All outputs can be active simultaneously.

RANGE: ±5 V

ACCURACY: ± (0.2% of output ±4 mV)

RESOLUTION: 12-bit A/D (2.4 mV)

OUTPUT CURRENT: ±50 mA

Digital Control Outputs

CONTROL CHANNELS PER MODULE: 8

OUTPUT VOLTAGES (no load):
High: 5.0 V ±0.2 V
Low: < 0.2 V

OUTPUT RESISTANCE: 100 ohms

CR9071E COUNTER & DIGITAL I/O MODULE

Counter Channels

COUNTER CHANNELS PER MODULE: 12

MAXIMUM COUNTS PER INTERVAL: 2^{32} Maximum counts per interval should never be reached because with a maximum input frequency of 1 MHz, the 32-bit counter will go 71.58 minutes before it rolls over. The maximum CR9000 scan rate is 1 minute.

SWITCH CLOSURE MODE (4 channels)
Minimum switch closed time: 5 ms
Minimum switch open time: 6 ms
Maximum bounce time: 1 ms open without being counted

HIGH FREQUENCY MODE (all channels)
Minimum pulse width: 500 ns
Maximum input frequency: 1 MHz
Thresholds: Pulse counted on transition from below 1.5 V to above 3.5 V
Maximum input voltage: ±20 V

LOW LEVEL AC MODE (8 channels)
Input hysteresis: 10 mV
Minimum ac voltage: 25 mV RMS
Maximum input voltage: ±20 V
Frequency range:

(mV RMS)	RANGE (Hz)
25 mV	1 to 10,000
≥50 mV	0.5 to 20,000

Digital Inputs/Outputs

I/O CHANNELS PER MODULE: 16

OUTPUT VOLTAGES (no load)
High: 5.0 V ±0.2 V
Low: < 0.2 V

OUTPUT RESISTANCE: 320 ohms

Input State
High: 3.5 to 5 V
Low: -0.5 to 1.2 V

Input Resistance: 100 KOhms

Interval Measurement

I/O CHANNELS:

Resolution is the scan rate

PULSE CHANNELS

Maximum interval: 1 minute
Resolution: ±40 ns

CR9080 PCMCIA and MEMORY MODULE

PCMCIA CARD INTERFACE: Accepts two Type I/II, or one Type III SRAM or ATA Flash Memory Cards.

SERIAL I/O: Allows serial communications with CSI peripherals at up to 115,200 bps.

TYPICAL CURRENT DRAIN: 300 mA active

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

CAMPBELL SCIENTIFIC, INC.
815 W. 1800 N. • Logan, Utah 84321-1784 • (435) 753-2342 • FAX (435) 750-9540
Offices also located in: Australia • Brazil • Canada • England • France • South Africa • Spain

Section 1. Installation

1.1 Enclosure

The CR9000 is equipped with either the –L option laboratory case or the –F option fiberglass case. The laboratory case can be used in a clean, dry, indoor environment or mounted in an enclosure. The fiberglass case provides a self-contained field enclosure. Campbell Scientific does not punch holes in the fiberglass case because it is our experience that most users like to customize the wire entry locations for their applications.

During the manufacturing of the fiberglass case, the base and lid are formed together to insure a perfectly matched fit. A six-digit serial number is stamped into the extruded aluminum rims on both the base and lid. When more than one CR9000 is owned, care should be taken to avoid a mismatch which could prevent a gas-tight seal. (Note that there is a pressure release valve on the enclosure. If you have difficulty removing the lid, try pressing the release valve to equalize the pressure differential between the case and atmosphere.)

1.1.1 Connecting Sensors

The CR9000 input modules use screw terminals for connecting sensor wires (Figure 1.1-1). Terminals for individual wires provide the most flexibility for connection to the wide range of sensors the CR9000 is used to measure as well as allowing the simplest field repair of the wire termination (strip and twist or tin).

1.1.2 Quick Connectors

Some customers who use CR9000s for numerous tests requiring the same or similar sets of sensors have found it useful to pre-wire the CR9000 to a set of plug-in quick connectors that mate with those installed on their sensors. Bulkhead type connectors can be installed either in the aluminum wiring panel cover or in the fiberglass case (Figure 1.1-2).

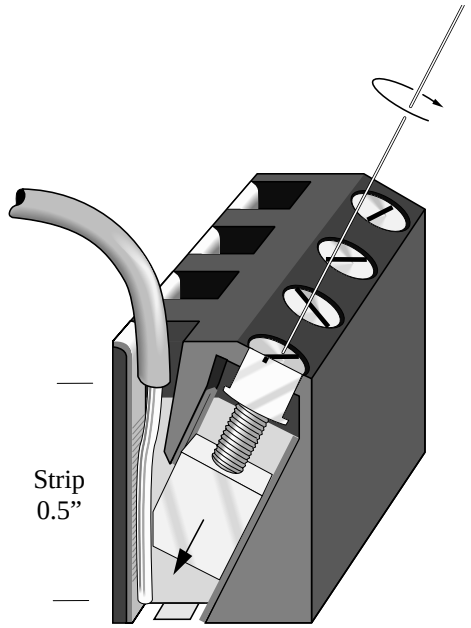


FIGURE 1.1-1 CR9000 Input Terminals

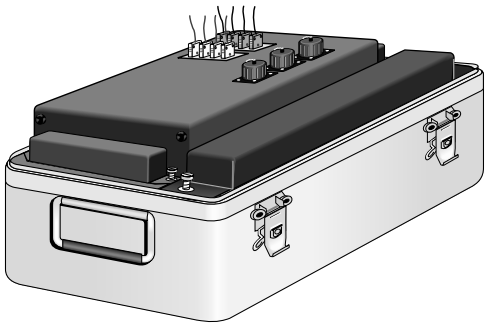


FIGURE 1.1-2 Quick Connectors Installed in CR9000 Cover

1.1.3 Junction Boxes

Individual sensor leads (and multiconductor cables) may be routed directly from the sensor locations to the CR9000 or routed to a junction box and then to the CR9000. When sensors are spread out over a large area, a junction box provides a convenient method for changing sensors in one location quickly. Junction boxes can also provide more localized protection against instrumentation damage as a result of lightning induced high voltages. Junction boxes should be sealed adequately to limit air exchange and stocked with fresh desiccant (Section 1.3). When used for thermocouple lead wires junction boxes need to be insulated to reduce thermal gradients (Section 3.4).

1.2 System Power Requirements and Options

The standard CR9000 is equipped with sealed lead acid battery packs and charging circuitry for charging the batteries from a 9-18 volt DC input. The input can come from 120/240 VAC line power via the universal AC power adapter (included with CR9000), vehicular 12V power sources, solar panels, et cetera. When fully charged, the internal batteries of the CR9000 are capable of providing 13-14 Amp-Hours, between 4 and 13 hours of operation in a typical application where the CR9000 is active continuously (not powering itself down).

1.2.1 Power Supply and Charging Circuitry

The 9011 Power Supply Module has two CHARGE inputs for connecting a DC Power source: either the plug connector used with the AC adapter or the screw terminals. A DC source with voltage in the range of 9 to 18 VDC will charge the internal lead acid batteries and power CR9000 provided sufficient current is available and the system is setup to use 3 amps or less (see Table 1.2-2). If the CR9000 system configuration requires greater than 3 amps, consult a Campbell Scientific applications engineer for information on the CR9011 Power Supply High-Current modification. The voltage is automatically stepped up to an adequate voltage for charging. A temperature compensated charging regulator circuit regulates the charging voltage supplied to the lead acid batteries and the CR9000. The charging circuitry operates with the ON/OFF switch in either position. The charging circuitry is NOT designed to charge a large external 12V battery.

Power for running the CR9000 and charging the internal batteries from AC line power is provided via the CR9000's universal AC adapter through the power input connector located on the 9011 Power Supply Module. The universal adapter converts 100–240 VAC 50–60 Hz to 17.5 VDC.

On the left end of the Power Supply Module there are two LEDs, Power and Charge. The charge LED is lit when there is sufficient power connected to charge the batteries. Power to the CR9000 is controlled by the ON/OFF toggle switch. The power LED is lit when the CR9000 is on. It goes off when the switch is in the off position or when the CR9000 is powered off under program control (PowerOff instruction).

The sealed lead acid battery packs are located at each end of the CR9000 (Figure 1.1-3).

FIGURE 1.1-3 CR9000 Battery Pack

TABLE 1.2-1. CR9000 Battery and Charging Circuitry Specifications

CR9000 WITH STANDARD BATTERIES (4):	
Battery life, no supplemental charge	13 hours to 10.5 V (assuming 1A current)
Voltage at full discharge	10.5 volts
Recharge time (AC Adapter input)	9 hours from 100% discharge 5 hours from 50% discharge.
Individual Batteries	
Type	Yuasa NP7-6
Nominal Voltage	6 Volts
Nominal Capacity	20 hr rate of 350 mA to 5.25 V, 7 Ahr 10 hr rate of 650 mA to 5.25 V, 6.5 Ahr
Operating Temperature range:	
Charge	–15 to 50 ºC
Discharge	–20 to 60 ºC
Shelf Life @ 20 ºC:	
1 month	97%
3 months	91%
6 months	85%
Life Expectancy:	
Standby	3 to 5 years
Cycle use	
100% depth of discharge	250 cycles
50% depth of discharge	550 cycles
30% depth of discharge	1200 cycles
Number of batteries	4
CHARGING CIRCUIT	
Type	Controlled voltage with temperature compensated voltage regulation
Charging Current	limited to 2 Amps max
POWER SUPPLY TRANSFORMER	
Input Voltage	100-240 VAC, 50-60 Hz
Input Current	1.4 A maximum
Output Voltage	17.5 VDC
Output Current	3.5 A maximum

NOTE

At typical CR9000 current demand, the batteries are 100% discharged at a system battery voltage of 10.5 V. Discharging the batteries below this voltage damages the cells. As can be seen from the above table, battery life expectancy decreases with depth of discharge. CSI's warranty does NOT cover battery or cell damage resulting from deep discharge.

Avoid deep discharge states by storing the battery voltage as part of the collected data and periodically checking the voltage record to be sure the batteries and charging system are working correctly.

All external charging devices must be disconnected from the CR9000 in order to measure the true voltage level of the internal batteries.

This CR9000 current drain depends on the number and type of modules installed, the sensors excited, and the scan interval and measurements made. The current drain of a specific CR9000 can be approximated from the information provided in Table 1.2-2.

TABLE 1.2-2. Current required by CR9000 modules

Model No.	Module	Quiescent Current	Current During Measurement
9031 9041 9011	CPU Module A/D Module Power Supply Module	410 mA	485 mA
9050(E) 9051E	Analog Input Module	0 mA	15 mA
9052DC	Filtered Analog Input Module	5 mA if not programmed	500 mA + 1.5 (sum of excitation currents on channels)
9055	50–Volt Analog Input Module	0 mA	15 mA
9058E	60 V Isolation Module	5 mA	360 mA
9060	Excitation Module	108 mA	125 mA
9070	Counter–Timer Module	0 mA	80 mA
9071E	Counter–Timer Module	25 mA	35 mA
9080	Peripheral Adapter Module		

As an example, the current drain of a CR9000 System containing the base system (CPU Module, A/D Module, and Power Supply Module: 410 mA / 485 mA) 1 9060 Excitation Module (108 mA / 125 mA, this does not include the current required for exciting the sensors), 2 9070 Counter/Timer Modules (0 mA / 30 mA), and 4 Analog Input Modules (0 mA / 60 mA) is about 518 mA between measurement scans and 700 mA during measurement. If it was active measuring close to 100 percent of the time, fully charged internal batteries (1 Ahr) would be depleted to a full SAFE discharge level (10.5 V) in about 20 hours. If the CR9000 system configuration requires greater than 3 amps, consult a Campbell Scientific applications engineer for information on the CR9011 Power Supply High-Current modification.

1.2.2 Connecting to Vehicle Power Supply

A vehicle 12 Volt electrical system can be connected directly to the charge input on the Power Supply Module. The Power Supply Module will step the voltage from the vehicle up or down to the proper voltage for charging the CR9000 batteries. The input is diode protected so the CR9000 batteries will not leak power to the vehicle if the vehicle's battery is low.

Because the charge input supplies power to charge the CR9000 batteries (up to two amps when discharged) as well as power the CR9000, the current drawn from the vehicle could be in excess of three amps.

1.2.3 Solar Panels

In a remote installation, solar panels, in conjunction with a large external battery, may be used to power the CR9000. The solar panels that Campbell Scientific carries on it's price list are sized for lower power requirements and will only be adequate if the CR9000 is programmed to periodically power itself off so that it is active less than 25 percent of the time. Other panels are available for continuous operation. Contact a Campbell Scientific application engineer for help in configuring a solar powered CR9000 installation.

1.2.4 External Battery Connection

An external battery may be used in place of the internal lead acid batteries of the CR9000. The external battery is connected using a special cable that is plugged into the CR9000 in place of a standard battery pack (Figure 1.2-1).

CAUTION Reverse polarity protection is NOT provided on these terminals and CR9000 damage will occur if external power is connected with reverse polarity.

CSI recommends using 16 AWG lead wires or larger when connecting an external battery to the CR9000.

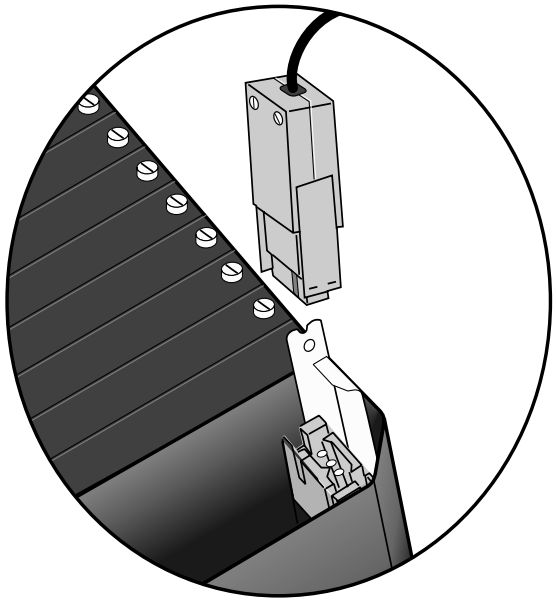


FIGURE 1.2-1 Connector for External Battery

1.2.5 Safety Precautions

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. Because the potential for excessive hydrogen build up does exist, CSI makes the following recommendations:

1. A CR9000 equipped with standard lead acid batteries should NEVER be used in environments requiring INTRINSICALLY SAFE EQUIPMENT.
2. When attaching an external battery to the CR9000, insulate the bare lead ends to protect against accidental shorting while routing the power leads.
3. When the CR9000 is to be located in a gas-tight enclosure or used in a gas-tight mode with the standard ENVIRONMENTALLY SEALED FIBERGLASS CASE, the internal lead acid batteries SHOULD BE REMOVED and an external battery substituted.

1.3 Humidity Effects and Control

The CR9000 system is designed to operate reliably under environmental conditions where the relative humidity inside its enclosure does not exceed 90% (noncondensing). Condensing humidity may result in damage to IC chips, microprocessor failure and/or measurement inaccuracies due to condensation on the various PC board runners. Effective humidity control is the responsibility of the user and is particularly important in environments where the CR9000 is exposed to salty air.

Two humidity control methods are:

1. the use of desiccant
2. nitrogen purging

1.3.1 Desiccant

As a minimal precaution, the packets of HUMI-SORB desiccant shipped with the CR9000 should be placed inside the case. These packets should be routinely replaced. Obviously, the desiccant requires more frequent attention in environments where the relative humidity is high.

1.3.2 Nitrogen Purging

Several CSI customers have had success in preventing humidity-related equipment malfunctions in harsh environments by allowing nitrogen gas to slowly bleed into the datalogger enclosure. The sensor leads, power cables, etc., are routed to the terminal blocks of the datalogger through simple, inexpensive conduit elbows which are left unplugged. A nitrogen bottle is then left at the field site with its regulator valve slightly open so that nitrogen is

allowed to escape slowly through a rubber tube which is routed along with the sensor leads through the conduit elbows into the CR9000 enclosure.

Equipment required for this method of humidity control generally can be obtained from any local welding supply shop and includes a nitrogen bottle, regulator with tube adapter (content gauge, optional), hose clamp and a suitable length of small diameter rubber tubing. Nitrogen bottles are available in various sizes and capacities. The size of the nitrogen bottle used depends on the transport facilities available to and from the field site and on the time interval between visiting the site. Where practical, larger nitrogen bottles should be used to reduce cost and refilling frequency.

1.4 Recommended Grounding Practices

1.4.1 Protection from Lightning

Primary lightning strikes are those where the lightning hits the datalogger or sensors. Secondary strikes occur when the lightning strikes somewhere near the lead in wires and induces a voltage in the wires. All input and output connections in the I/O Module are protected using spark gaps. This transient protection is useless if there is not a good connection between the CR9000 and earth ground.

All dataloggers in use in the field should be grounded. A 12 AWG or larger wire should be run from the grounding terminal on the right side of the I/O Module case to a grounding rod driven far enough into the soil to provide a good earth ground.

A modem/phone line connection to the CR9000 provides another pathway for transients to enter and damage the datalogger. The phone lines should have proper spark gap protection at or just before the modem at the CR9000. The phone line spark gaps should also have a solid connection to earth ground.

1.4.2 Effect on Measurements: Common Mode Range

A difference in ground potential between a sensor or signal conditioner and the CR9000 can offset the measurement. A differential voltage measurement gets rid of offset caused by a difference in ground potential. However, in order to make a differential measurement, the inputs must be within the CR9000 common mode range of ±5V (+15/-5 for the CR9052E module, ±50V for the 9055 module, or ±60V for the CR9058E module).

The common mode range is the voltage range, relative to CR9000 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 4V and the low side is at 3V relative to CR9000 ground, there is no problem, a measurement made on the ±1.5V range would indicate a signal of 1V. However, if the high input is at 5.8V and the low input is at 4.8V, the measurement cannot be made because the high input is outside of the CR9000 common mode range.

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

Problems with exceeding common mode range may be encountered when the CR9000 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 CR9000. 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 CR9000 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 CR9000 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.

1.5 Use of Digital Control Ports for Switching Relays

The digital control outputs on the 9060 Excitation Module and the I/O channels on the CR9070/CR9071E Counter Timer Module may be used to actuate controls but, because of current supply limitations, the output ports are not used directly to drive a relay coil. Relay driver circuitry is used to switch current from another source to actually power the relay. These relays may be used for activating an external power source to run a fan motor or for altering an external circuit as a means of multiplexing signal lines, etc. CSI's Model A21REL-12 and A6 REL12 are Relay Controllers using a 12 VDC source for switching the relays. Solid state relays that may be controlled with a 0-5 V logic signal are also available for switching AC or DC power.

Figure 1.5-1 is a schematic representation of a typical external coil driven relay configuration which may be used in conjunction with one of the CR9000s digital control output ports. The example shows a DC fan motor and 12V battery in the circuit. This particular configuration has a coil current limitation of 75mA because of the NPN Medium Power Transistors used (Part No. 2N2222).

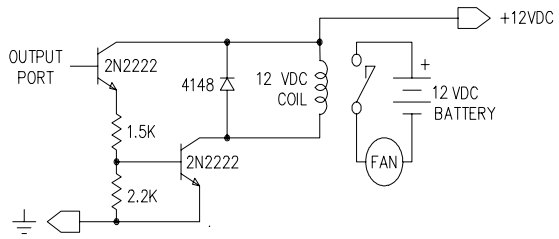


FIGURE 1.5-1. Typical Connection for Activating/Powering External Devices, Using a Digital Control Output Port and Relay Driver

Section 2. Data Storage and Retrieval

The CR9000 can store individual measurements or it may use its extensive processing capabilities to calculate averages, maxima, minima, histograms, FFTs, etc., on periodic or conditional intervals. Data are stored in tables. For simplicity, the PC9000 program generator allows a maximum of six data tables (in the native language the limit on the number of data tables depends on where the tables are stored). The number of tables and the values to output in each table are selected when running the program generator (Overview) or when writing a datalogger program directly (Sections 4 – 9).

2.1 Data Storage in CR9000

There are three possible areas for data storage on the CR9000:

Internal Static Ram–*avaliable storage the only limit on the number of tables*

Internal Flash Memory–*6 data tables maximum*

PCMCIA PC Card–*30 data tables maximum*

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

When PC9000 requests data from a table that is stored in Flash memory or a PC card, the CR9000 only looks for the data in flash memory or in the PC card when the oldest data are requested or if the data are not available in CPU memory.

In the CRBASIC program, the DataTable instruction sets the size of the data table or buffer area. A data table can be stored in a PC card by including the PAMOut instruction within the data table declaration. A data table can be stored in internal flash memory by including the FlashOut instruction within the data table declaration. A data table cannot be sent to both a PC card and to internal flash memory; the CR9000 will flag an error if both PAMOut and FlashOut are in a data table declaration.

2.1.1 Internal Static Ram

Internal Ram is used as either the sole storage area for a data table or as a buffer area when data are sent to PC card or to internal flash memory. The only limit on the number of tables is the available memory. Data in RAM are lost if the CR9000 is powered down either by switching off the power switch or with the PowerOff instruction.

2.1.2 Internal Flash Memory

There are 1.5 M bytes of flash memory available for data storage. No more than six data tables can be stored in flash memory. Flash Memory is always fill and stop, that is, once the space allocated for the data table is full, no more data are stored until the table is reset.

When the CR9000 compiles and runs a program that uses flash memory, it checks to see if the program is different from the last program it ran. If the program has changed (including any changes to comments), flash memory is erased and reset. If the program is the same, the CR9000 leaves the flash memory as it was, appending data to tables that are not yet full.

The 1.5 M bytes of flash memory available for data storage is in six 256 K erasable segments. A segment can only store data from 1 table (otherwise the tables could not be individually reset). This is the reason a maximum of six tables are possible (none larger than 256 K). Automatic allocation (negative number for FlashOut size) will divide tables on the 256 K boundaries.

2.1.3 9080 PAM Module – PCMCIA PC Card

The CR9000 9080 PAM Module allows expanding the CR9000's storage capacity with Type I, II, or III PCMCIA Cards. SRAM, ATA Flash, and ATA hard disk cards are supported. ATA hard disks cards cannot withstand the environmental temperature range of the CR9000's specifications. A program can send a maximum of 30 data tables to PC cards.

Data stored on cards can be retrieved through one of the communication links to the CR9000 or by removing the card and inserting it in a PC card slot in a computer. Converting the data using the computer's PC card slot is much faster than retrieving it through the CR9000 using one of the communication links.

The CR9000 uses an MS DOS format for the PC cards. Cards can be formatted in a PC or in the CR9000.

TABLE 2.2-1. CR9000 DATA TYPES

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

2.2 Internal Data Format

Data are stored internally in a binary format. Variables and calculations are performed internally in IEEE 4 byte floating point with some operations calculated in double precision. There are two data types used to store data: IEEE4 four byte floating point and Campbell Scientific two byte floating point (FP2). The data format is selected in the instruction that outputs the data. A third data type, the four byte integer format (LONG) is used by the CR9000 for storing time and record number. Within the CR9000, time is stored as integer seconds and nanoseconds into the second since midnight, the start of 1990. While IEEE 4 byte floating point is used for variables and internal calculations, FP2 is adequate for most stored data. Campbell Scientific 2 byte floating point provides 3 or 4 significant digits of resolution, and requires half the memory space as IEEE 4 byte floating point (2 bytes per value vs 4).

TABLE 2.2-1. Resolution and Range Limits of FP2 Data		
Zero	Minimum Magnitude	Maximum Magnitude
0.000	±0.001	±7999.

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

TABLE 2.2-2. FP2 Decimal Location	
Absolute Value	**Decimal Location**
0 - 7.999	X.XXX
8 - 79.99	XX.XX
80 - 799.9	XXX.X
800 - 7999.	XXXX.

2.3 Data Collection

Data can be transferred into a computer using PC9000 via a communications link or by transferring a PC card from the PC9000 to the computer. There are three ways to collect data via a link to the CR9000 using the PC9000 software:.

1. The **collect** menu is used to collect any or all stored data Tables and is used for most archival purposes.

2. In PC9000's Field Monitor RealTime window there is a **"Disc file"** check box. Data stored to the table while the box is checked are also stored to a file on the PC. If communications cannot keep up with the measurement rate, there will be holes (missing data) in the data files.

3. Logger Files under the Tools menu has the option of retrieving a file from a PC card. This can be used to retrieve a data file in the raw binary format.

When the CR9000 is used without a computer in the field, or large data files are collected on a PC card, the **PC card** can be transported to the computer with the data on it.

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

2.3.1 The Collect Menu

When Retrieve Data is selected in the Collect menu, PC9000 displays the Collect Data dialog box (Figure 2.3.1). The station name (may be entered by the user when a program is downloaded) is retrieved from the connected CR9000 and shown at the top.

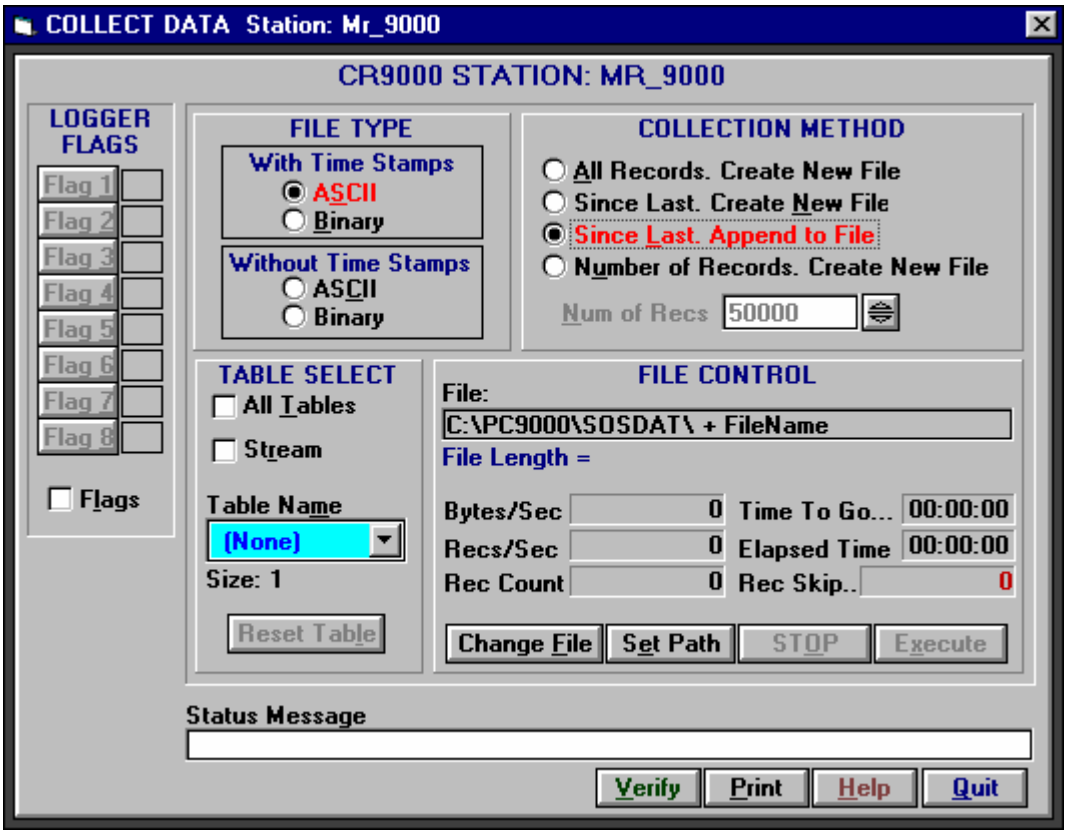


FIGURE 2.3-1. Collect Data Dialog Box

2.3.1.1 File Type

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

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

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

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

2.3.1.2 Collection Method

All Records, Create New File – Collects the entire table stored in the CR9000. PC9000 gets the current record number from the table in the CR9000 and then retrieves the oldest record in the table up to the current record number. The number in the file name is incremented to create the file name in which the data are stored.

Since Last, Create New File – Click here to save new data in a new file. PC9000 searches for the last file with the Root name, gets the last record number from that file, then the current record from the table in the CR9000, and requests all records in between those numbers from the CR9000. The number in the file name is incremented to create the file name in which the data are stored.

Since Last, Append To File – Click here to append retrieved data to the end of the named file. PC9000 searches for the last file with the Root name, gets the last record number from that file, then the current record from the table in the CR9000, and requests all records in between those numbers from the CR9000. The data are appended to the existing file.

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

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

2.3.1.3 Table Selection

All Tables – When the All Tables box is checked, all data tables except the Public and Status tables are collected when collection is executed. The data from each table are stored in a file with the table name and increment number (see Table naming). This is a convenient method of collecting all data from the CR9000. The first time data are collected, “all data” is checked and the file type and collection method are selected. PC9000 remembers the settings, and on subsequent collections the operator only needs to click on execute.

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

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

Reset Table – Resetting a data table erases all data in the table and sets the record number back to 0. Unless the table is configured as fill and stop by the CR9000 program, it is not necessary to reset the table because the "Since Last" collection option can be used to get only the new data. If the table is configured as fill and stop, it stops collecting data once full and must be reset before more data can be collected. Use with caution.

2.3.1.4 File Control

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

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

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

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

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

EXECUTE – Press here to begin collecting data.

2.3.1.5 Status Messages

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

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

LOGGER MESSAGE – Displays messages from the CR9000.

2.3.2 RealTime Write File

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

This collection method requires that the PC is connected to the CR9000 while the data are collected. Because the beginning and end points of the data file are roughly determined by when the box is checked, this is best suited to collecting data when the user rather than the measurements determine when data should be collected.

The bottom line of the screen will periodically display the current record being written. The name of the file written will be the first six characters of the table name plus 2 digits and an extension of .DAT. If the table name is MAIN then the first file created will be named MAIN00.DAT. The next

file will be named MAIN01.DAT and so on. It takes a little time to open the file so be sure it is opened in advance of the event you want to store.

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

2.3.3 Logger Files Retrieve

Logger Files under the PC9000 tools menu allows the user to check the programs stored in CPU Flash memory and the files stored on the PCMCIA cards. Any of the files shown in logger files can be copied to the computer by highlighting the file and pressing the retrieve button. Data files in the CR9000 CPU and Flash memory are not shown.

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

FIGURE 2.3-2. Logger Files Dialog Box

2.3.4 Via PCMCIA PC Card

When the CR9000 is used without a computer in the field, or large data files are collected on a PC card, the **PC card** can be transported to the computer with the data on it.

2.3.4.1 Removing Card from CR9000

The 9080 PCMCIA Adapter Module contains slots for two Type I/II PCMCIA cards or one type III PCMCIA card. The LEDs indicate the status of the cards in slots A and B.

- **Not lit:** no card detected.
- **green:** present and correctly formatted.
- **red:** present but corrupt.
- **orange:** accessing the card.

To remove a card, press the button next to the status LED to power down the card. The LED will blink green several times then go out for 10 seconds. Remove the card while the LED is not lit. The card will be reactivated if not removed.

Caution: Removing a card while it is active can cause garbled data and can actually damage the card. Do not switch off the power (9011 Module) while the cards are present and active.

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

2.3.4.2 Converting File Format

The CR9000 stores data on PC cards in TOB2 Format. TOB2 is a binary format that incorporates features to improve reliability of the PC Cards. TOB2 allows the accurate determination of each record's time without the space required for individual time stamps.

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

PC9000's file converter will convert TOB1 files to ASCII or TOB2 files to TOB1, ASCII, DaDisp, or ID-2000W. The Convert Data Files option is in the File Menu. The options for TOB2 appear after the name of the file to convert has been selected (Figure 2.3-3.)

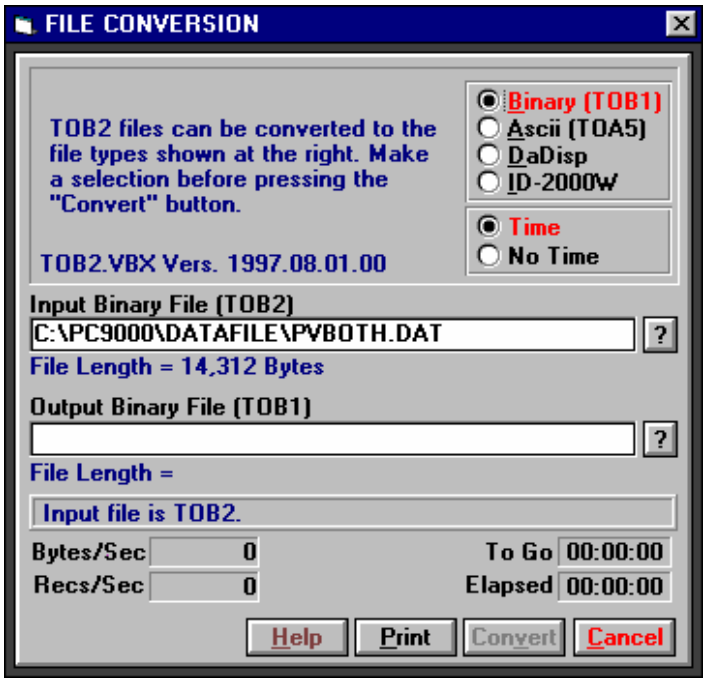


FIGURE 2.3-3. File Conversion Dialog Box

2.4 Data Format on Computer

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

2.4.1. Header Information

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

"File Format","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. This information is used by the historical graphing and file conversion functions of PC9000.

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

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

Logger Serial Number
The serial number of the logger that the data was collected from. This is the serial number of the CR9000 CPU.

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

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

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

Table Name
The data table name.

Field Name
The name of the field in the data table. This name is created by the CR9000 by appending underscore (_) and a three character mnemonic for the output processing.

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

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

Smp = Sample

Max = Maximum

Min = Minimum

Avg = Average

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

UINT4 = Unsigned 4 byte integer

IEEE4 = 4 byte floating point

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

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

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

2.4.2 TOA5 ASCII File Format

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

```
"TOA5","Bob's9K","CR9000","1048575","1.00","EXPLDAT.DLD","4339","Temp"
"TIMESTAMP","RECORD","RefTemp_Avg","TC_Avg(1)","TC_Avg(2)","TC_Avg(3)","TC_Avg(4)
"
"TS","RN","degC","degC","degC","degC","degC"
"","","Avg","Avg","Avg","Avg","Avg"
"1995-09-19 14:31:43.84",458,29.94,25.6,25.36,25.48,25.4
"1995-09-19 14:31:43.85",459,29.93,25.6,25.36,25.41,25.35
```

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

TOA5	Bob's9K	CR9000	1048575	1.00	EXPLDAT.DLD	4339	Temp
TIMESTAMP	RECORD	RefTemp_Avg	TC_Avg(1)	TC_Avg(2)	TC_Avg(3)	TC_Avg(4)	
TS	RN	degC	degC	degC	degC	degC	
		Avg	Avg	Avg	Avg	Avg	
1995-09-19 14:31:43.84	458	29.94	25.6	25.36	25.48	25.4	
1995-09-19 14:31:43.85	459	29.93	25.6	25.36	25.41	25.35	

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

```
"TOA5","Bob's9K","CR9000","1048575","1.00","EXPLDAT.DLD","4339","Temp"
"RefTemp_Avg","TC_Avg(1)","TC_Avg(2)","TC_Avg(3)","TC_Avg(4)"
"degC","degC","degC","degC","degC"
"Avg","Avg","Avg","Avg","Avg"
29.94,25.6,25.36,25.48,25.4
29.93,25.6,25.36,25.41,25.35
```

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

TOA5	Bob's9K	CR9000	1048575	1.00	EXPLDAT.DLD	4339	Temp
RefTemp_Avg	TC_Avg(1)	TC_Avg(2)	TC_Avg(3)	TC_Avg(4)			
degC	degC	degC	degC	degC			
Avg	Avg	Avg	Avg	Avg			
29.94	25.6	25.36	25.48	25.4			
29.93	25.6	25.36	25.41	25.35			

2.4.3 TOB1 Binary File Format

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

TOB1,Bob's9K,CR9000,1048575,1.00,EXPLDAT.DLD,4339,Temp
SECONDS,NANOSECONDS,RECORD,RefTemp_Avg,TC_Avg(1),TC_Avg(2),TC_Avg(3),TC_Avg(4)
SECONDS,NANOSECONDS,RN,degC,degC,degC,degC,degC
,,,Avg,Avg,Avg,Avg,Avg
UINT4,UINT4,UINT4,IEEE4,IEEE4,IEEE4,IEEE4,IEEE4
(data lines are binary and not directly readable)

This is an example of binary without time stamps.

TOB1,Bob's9K,CR9000,1048575,1.00,EXPLDAT.DLD,4339,Temp
RefTemp_Avg,TC_Avg(1),TC_Avg(2),TC_Avg(3),TC_Avg(4)
degC,degC,degC,degC,degC
Avg,Avg,Avg,Avg,Avg
IEEE4,IEEE4,IEEE4,IEEE4,IEEE4
(data lines are binary and not directly readable)

2.4.4 TOB2 Binary File Format

The TOB2 binary format has a header similar to the other formats. TOB2 data is stored in fixed size “frames” that generally contain a number of records. The size of the frames is a function of the record size. The frames are time stamped, allowing the calculation of time stamps for their records. If there is a lapse in periodic interval records that does not occur on a frame boundary, an additional time stamp is written within the frame and its occurrence noted in the frame boundary. This additional time stamp takes up space that would otherwise hold data.

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

Section 3. CR9000 Measurement Details

3.1 Measurements using the CR9041 A/D

The CR9050(E), CR9051E, and the CR9055(E) modules all use the A/D module to digitize their analog measurements. Section 3.1 documents measurement details for the measurements made using these modules. The Filter module (CR9052DC) and the Isolation Module (CR9058E) both have an A/D converter for each channel. The analog inputs are digitized by the modules (the CR9041 A/D module is not used) and the digital data is sent directly to the CR9000's CPU module. The differences in measurement details for these modules are covered in Sections 3.2 and 3.3. The measurement details for the CR9070 and CR9071 Pulse modules are covered in Section 3.4.

3.1.1 Analog Voltage Measurement Sequence

The CR9000 measures analog voltages with a sample and hold analog to digital (A/D) conversion. The signal at a precise instant is sampled and this voltage is held or "frozen" while the digitization takes place. The A/D conversion is made with a 16 bit successive approximation technique which resolves the signal voltage to approximately one part in 62,500 of the full scale range (e.g., for the ±5000 mV range, 10 V/62,500 = 160 µV). The analog measurements are multiplexed through a single A/D converter with a maximum conversion rate of 100,000 per second or one every 10 µs.

The timing of the CR9000 measurements is precisely controlled by the task sequencer, a combination of components that switches the measurement circuitry on a rigid schedule that is determined at compile time and loaded into the task sequencer's memory. The basic tick of the task sequencer measurement clock may be thought of as 10 µs. The minimum time between measurements is 10 µs. When voltage signals are measured at a 10 µs/measurement rate, every 10 µs the task sequencer holds the signal from one channel and then switches to the next channel. When the signal is held, the A/D converter goes to work and ships the result off to the transputer memory.

The instructions executed by the task sequencer (e.g., hold, turn on the excitation, switch to the next channel, etc.) take 400 ηs each. When measuring every 10 µs, after holding for one measurement, the task sequencer switches to the next channel (400 ηs), waits 9200 ηs, then holds for the next measurement (400 ηs).

Changing voltage ranges requires one 10 µs tick; the task sequencer sets up the new voltage range then delays until the next 10 µs boundary before switching to the first channel. This only occurs before the first measurement within a scan or when the voltage range actually changes. **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 CR10, 21X 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 reverse the polarity of the excitation voltage (**RevEx**), reverse the high and low differential inputs (**RevDiff**), to set the time to wait between switching to a channel and making a measurement (**Delay**), and the length of time to integrate a measurement (**Integ**).

3.1.1.1 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, 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 CR9000 measurement circuitry. One measurement is made with the high input referenced to the low input and a second with the low referenced to the high.

3.1.1.2 Delay

When the CR9000 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 CR9000 delays, 10 μs for the 5000 and 1000 mV ranges and 20 μs for the 200 and 50 mV ranges, are the minimum required for the CR9000 to settle to within its accuracy specifications. Additional delay is necessary when working with high sensor resistances or long lead lengths (higher capacitance). It is also possible to shorten the delay on the 200 and 50 mV ranges to 10 μs when speed and resolution is more important than high accuracy. Using a delay increases the time required for each measurement.

When the CR9000 Reverses the differential input or the excitation polarity It delays the same 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 CR9000 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.

3.1.1.3 Integration

With the 9050 and 9055 analog input modules, there is no analog integration of the signal and minimal filtering from the 422 ohm series resistor and 0.001 μF capacitor to ground that protect the input. The signal is sampled when the task sequencer issues a hold command and any noise that may be on the signal becomes part of the measured voltage. The rapid sample is a necessity for high speed measurements. Integrating the signal will reduce noise. When lower noise measurements are needed or speed is not an issue, integration can be specified as part of the measurement.

The CR9000 uses digital integration. An integration time in microseconds (10 μs resolution) is specified as part of the measurement instruction. The CR9000 will repeat measurements every 10 μs throughout the integration interval and store the average as the result of the measurement.

The random noise level is decreased by the square root of the number of measurements made. For example, the input noise on the ±5000 mV range with no integration (one measurement) is 90 μV RMS; integrating for 40 μs (four measurements) will cut this noise in half (90/(√4)=45).

One of the most common sources of noise is not random but is 60 Hz from AC power lines. An integration time of 16,670 μs is equal to one 60 Hz cycle. Integrating for one cycle will integrate the AC noise to 0.

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

3.1.2 Single Ended and Differential Voltage Measurements

A single-ended measurement is made on a single input which is measured relative to ground. A differential measurement measures the difference in voltage between two inputs. Twice as many single ended measurements can be made per Analog Input Module.

NOTE There are two sets of channel numbers on the Analog Input Modules. 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 CR9000 ground, any difference in ground potential between the sensor and the CR9000 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 CR9000 is grounded, the measured voltage would be 1 mV greater than the thermocouple output, or approximately 25 ºC high. Another instance where a ground potential difference creates a problem is in a where external signal conditioning circuitry is powered from the same source as the CR9000. 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.

3.1.2.1 Single Ended Voltage Range

The voltage range for single ended measurements is the range in which the input voltage must be, relative to CR9000 ground, for the measurement to be made.

The resolution (the smallest difference that can be detected) for the A/D conversion is a fixed percentage of the full scale range. To obtain the best resolution, select the smallest range that will cover the voltage output by the sensor. For example, the resolution of an A/D conversion made on the ± 50 mV range is 1.6 µV; the resolution on the ±5000 mV range is 160 µV. A copper-constantan thermocouple outputs a voltage of about 40 µV / °C (difference in temperature between the measurement and reference junction). The temperature resolution on the ± 50 mV range is 0.04 degrees (1.6 µV / 40 µV / 1°C); the resolution on the ±5000 mV range is 4 degrees (160 µV / 40 µV / °C). Because the smallest ± 50 mV range will allow a 1250 degree difference (0.05 V / 0.00006 V), which is greater than the sensor capability (-200 to 400 degrees C) there is no reason to use a larger range.

3.1.2.2 Differential Voltage Range

When a differential voltage measurement is made, the high (H) input is referenced to the low (L) input. To obtain the best resolution, select the smallest range that will cover the voltage output by the sensor as described for single ended voltage measurements above.

Common mode range

In order to make a differential measurement, the inputs must be within the CR9000 common mode range of ±5 V for the 9050 Analog Input Module and the 9051E Fault-Tolerant Analog Input Module or ±50 V for the 9055 Analog input Module. The common mode range is the voltage range, relative to CR9000 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 CR9000 ground, there is no problem. A measurement made on the 9050 module with 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 should not be trusted because the high input is outside of the 9050 Module's ±5 V common mode range. Differential made on signals outside the common mode range

may return the over range value Not-A-Number (NAN) or a valid, but incorrect, number. To avoid misleading data, either be sure the signal is referenced to CR9000 ground or use the voltage range R option to check common mode range as described below.

Sensors that have a floating output (the output is not referenced to ground through a separate connection) may float out of common-mode range, causing measurement problems. The voltage range C option described below can be used to keep floating differential inputs within common-mode range. Another solution is to connect one side of the differential input to ground to ensure the signal remains within the common mode range.

There are several measurement options for differential voltage measurements and differential voltage thermocouple measurements (VoltDiff and TCDiff), specified in the range code, that are related to common mode:

Range Code C option : Before making the differential measurement, the H and L inputs are briefly connected to internal voltages within the common mode range allowing floating inputs to remain within common mode for the differential measurement.

The C option has the added benefit of being able to detect an open input (e.g., broken thermocouple). The H input is connected to a voltage approximately 2.8 V above the L input so that an open input will result in an over range on the ±200 mV and ±50 mV input ranges. With an open input the high and low inputs are floating independently and remain close to the values they reached while connected to the excitation, over ranging voltage ranges up to ±200 mV and causing Not a Number (NAN) to be returned for the result.

Check common mode range, R, option (e.g., mV1000R): After making the differential measurement, appropriate single-ended measurements are made on the H and L inputs to determine if the differential measurement was within common-mode range. The result of the differential measurement is set to the over range value (NAN) if the measurement was determined to be out of common-mode range.

The options to pull into common mode before the differential measurement and to check that the input remained in common mode with a single ended measurement after the differential measurement can be combined (e.g., mV1000CR).

Problems with exceeding common mode range may be encountered when the CR9000 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 CR9000. 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 CR9000 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 CR9000 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. The maximum offset when the inputs are reversed on a differential measurement offset is about one quarter what it is on a single ended or one way differential.

NOTE Sustained voltages in excess of ±20 V on the 9050 Module inputs or ±150 V on the 9055 Module inputs will damage the CR9000 circuitry.

3.1.3 Signal Settling Time

Whenever an analog input is switched into the CR9000 measurement circuitry prior to making a measurement, a finite amount of time is required for the signal to stabilize at it's correct value. The rate at which the signal settles is determined by the input settling time constant which is a function of both the source resistance and input capacitance. The CR9000 delays after switching to a channel to allow the input to settle before initiating the measurement. The default delays used by the CR9000 are 10 µs on the ±5000 and ±1000 mV ranges and 20 µs on the ±200 and ±50 mV range. This settling time is the minimum required to allow the input to settle to the resolution specification. The 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.

MINIMIZING SETTLING ERRORS

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

1. When measurement speed is not a prime consideration, additional delay time can be used to ensure ample settling time.

2 When making fast bridge measurements, use the continuous excitation channels (1-6) to excite the bridges so the excitation doesn't have to settle before each measurement.

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

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

5. Use the CR9000 to measure the input settling error associated with a given configuration. Stabilize the sensor so that its output is not changing. Program the CR9000 to make the measurement with the delay you would like to use and a second time with a much longer delay that ensures adequate settling time. The difference between the two measurements is the error due to inadequate settling time.

3.1.4 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. 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 CR9000 determines thermocouple temperatures using the following sequence. First the temperature of the reference junction is measured. If the reference junction is the CR9000 Analog Input Module, the temperature is measured with the PRT in the 9050 Analog Input Module (ModuleTemp instruction). The reference junction temperature in ^{o}C is stored and then referenced by the thermocouple measurement instruction (TCDiff or TCSE). The CR9000 calculates the voltage that a thermocouple of the type specified would output at the reference junction temperature if its reference junction were at 0 ^{o}C, and adds this voltage to the measured thermocouple voltage. The temperature of the measuring junction is then calculated from a polynomial approximation of the NIST TC calibrations

3.1.4.1 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 CR9000 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.

Reference Junction Temperature with 9050

The PRT in the CR9000 is mounted on the circuit board near the center of the 9050 terminal strip. This resistance temperature device (RTD) is accurate to ±0.1 ^{o}C over the CR9000 operating range. The I/O Module was designed to minimize thermal gradients. It is encased in an aluminum box which is thermally isolated from the CR9000 fiberglass enclosure. Measurement modules have aluminum mounting plates extending beyond the edges of the circuit cards that provide thermal conduction for rapid equilibration of thermal gradients. Sources of heat within the CR9000 enclosure exist due to power dissipation by the electronic components or charging batteries. In a situation where the CR9000 is at an ambient temperature of approximately 20^{o}C and no external temperature gradients exist, the temperature gradient between one end of an Analog Input card to the other is likely to be less than 0.1°C. The gradient from one end of the I/O Module to the other, is likely to be about

4°C. The end of the enclosure with the CPU Module will be warmer due to heat dissipated by the processor.

For the best accuracy, use the temperature of each 9050 module as the reference temperature for any thermocouples attached to it. Given the above conditions, this would keep the reference junctions within 0.05°C of the temperature of the RTD. When making more thermocouple measurements than can be accomplished on a single 9050 module, it is faster to measure the temperature of one 9050 module and use it for all thermocouples. If speed is more important than the reduced accuracy, the temperature of a single 9050 module can be used for thermocouples connected to other modules.

A foam block that fits under the terminal cover is sent with the CR9000. When installed, this block insulates and limits air circulation around the terminals. This helps to limit temperature gradients on the analog input modules, particularly when the CR9000 is subjected to rapid temperature changes. Figure 3.4-1 shows the thermocouple temperature errors experienced on different channels of the analog module when the CR9000 was subjected to an abrupt change in temperature (-40ºC to +60ºC in approximately 12 minutes).

Thermocouple Limits of Error

The standard reference which lists thermocouple output voltage as a function of temperature (reference junction at 0ºC) is the National Institute of Standards and Technology Monograph 175 (1993). The American National Standards Institute has established limits of error on thermocouple wire which is accepted as an industry standard (ANSI MC 96.1, 1975). Table 3.4-1 gives the ANSI limits of error for standard and special grade thermocouple wire of the types accommodated by the CR9000.

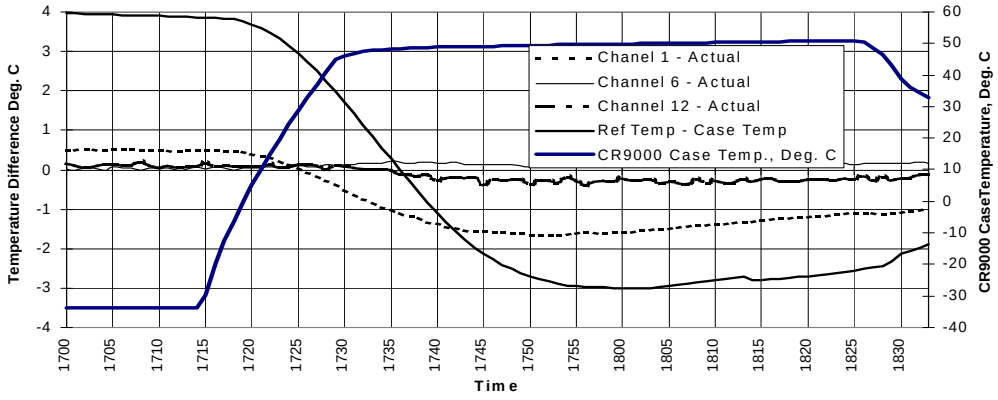


FIGURE 3.4-1. Thermocouple Temperature Errors During Rapid Temperature Change

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

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

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

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

Accuracy of the Thermocouple Voltage Measurement

The accuracy of a CR9000 voltage measurement is specified as 0.07% the measured voltage plus 4 A/D counts of the range being used to make the measurement. The input offset error reduces to 1 A/D count if a differential measurement is made utilizing the option to reverse the differential input.

For optimum resolution, the ±50 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 40 to 60 °C between the measurement and reference junctions is required for a thermocouple to output 2.285 mV, the voltage at which 0.07% of the reading is equal to 1 A/D count (1.6 mV). For example, assume that a type T thermocouple is used to measure a temperature of 45 °C and that the reference temperature is 25 °C. The voltage output by the thermocouple is 830.7 µV. At 45 degrees a type T thermocouple outputs 42.4 µV per ºC. The possible slope error in the voltage measurement is 0.0007x830.7 µV = 0.58 µV or 0.014 ºC (0.58/42.4). An A/D count on the ±50 mV range is worth 1.6 µV or 0.038 ºC. Thus, the possible error due to the voltage measurement is 0.166 ºC on a single-ended or non-reversing differential, or 0.052 ºC with a reversing differential measurement. The value of using a differential measurement with reversing input to improve accuracy is readily apparent.

The error in the temperature due to inaccuracy in the measurement of the thermocouple voltage is worst at temperature extremes, particularly when the temperature and thermocouple type require using the 200 mV range. For example, assume type K (chromel-alumel) thermocouples are used to measure temperatures around 1300 ºC. The TC output is on the order of 52 mV, requiring the ±200 mV input range. At 1300 ºC, a K thermocouple outputs 34.9 µV per ºC. The possible slope error in the voltage measurement is 0.0007x52 mV = 36.4 µV or 1.04 ºC (36.4/34.9). An A/D count on the 200 mV range is worth 6.3 µV or 0.18 ºC. Thus, the possible error due to the voltage measurement is 1.77 ºC on a single-ended or non-reversing differential, or 1.22 ºC with a reversing differential measurement.

TABLE 3.4-2. Voltage Range for maximum Thermocouple resolution

Thermocouple Type and temperature range ºC	Temperature range for ±50 mV range	Temperature range for ±200 mV range
T -270 to 400	-270 to 400	not used
E -270 to 1000	-270 to 660	>660
K --270 to 1372	-270 to 1230	>1230
J -210 to 1200	-210 to 870	> 870
B 0 to 1820	0 to 1820	not used
R -50 to 1768	-50 to 1768	not used
S -50 to 1768	-50 to 1768	not used
N -270 to 1300	-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. If the voltage potential exceeds the common mode range of the 9050 module (e.g., the +12 V terminal of an automotive battery) it is possible to use the 9055 ±50 V Analog Input Module to make the Thermocouple measurement. The resolution and noise level are much worse than with the 9050 Module. The ±500 mV range offers the best resolution, 1 A/D count is 16 µV, about 0.4 ºC for most thermocouples.

Noise on Voltage Measurement

The input noise on the ±50 mV range for a measurement with no integration is 4 µV RMS. On a type T thermocouple (approximately 40 µV/ºC) this is 0.1 ºC. Note that this is an RMS value, some individual readings will vary by greater than this. By integrating for 500 µs (50 samples) the noise level is reduced to 0.6 µV RMS ($4/\sqrt{50}=0.6$). If a 500 µs integration is combined with reversing the differential input, there are 100 samples in the measurement and the noise level is reduced to 0.4 µV RMS.

Thermocouple Polynomial: Voltage to Temperature

NIST Monograph 175 gives high order polynomials for computing the output voltage of a given thermocouple type over a broad range of temperatures. In order to speed processing and accommodate the CR9000's math and storage capabilities, 4 separate 6th order polynomials are used to convert from volts to temperature over the range covered by each thermocouple type. Table 3.4-3 gives error limits for the thermocouple polynomials.

TABLE 3.4-3. Limits of Error on CR9000 Thermocouple Polynomials (Relative to NIST Standards)

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

Reference Junction Compensation: Temperature to Voltage

The polynomials used for reference junction compensation (converting reference temperature to equivalent TC output voltage) do not cover the entire thermocouple range. Substantial errors will result if the reference junction temperature is outside of the linearization range. The ranges covered by these linearizations include the CR9000 environmental operating range, so there is no problem when the CR9000 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 ºC. The compensation voltage calculated by the CR9000 corresponds to a temperature of 272.6 ºC, a -27.4 ºC error. The type T thermocouple with the measuring junction at 290 ºC and reference at 300 ºC would output -578.7 µV; using the reference temperature of 272.6 ºC, the CR9000 calculates a temperature difference of -10.2 ºC, a -0.2 ºC error. The temperature calculated by the CR9000 would be 262.4 ºC, 27.6 ºC low.

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

Type	Range ºC	Limits of Error ºC
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 CR9000 are likely to be due to the limits of error on the thermocouple wire and in the reference temperature determined with the 9050 RTD. 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 ºC is measured with a type T (copper-constantan) thermocouple, using the ±50 mV range. The nominal accuracy on this range is 1 µV (0.01% of 10 mV) which at 45 ºC changes the temperature by 0.012 ºC. The RTD is 25 ºC but is indicating 25.1 ºC, and the terminal that the thermocouple is connected to is 0.05 ºC cooler than the RTD.

TABLE 3.4-5. Example of Errors in Thermocouple Temperature

Source	Error: ºC : % of Total Error			
	Single-Ended or single Differential		Reversing Differential w:500 µs Integration	
	ANSI TC Error (1ºC)	TC Error 1% Slope	ANSI TC Error (1ºC)	TC Error 1% Slope
Reference Temp.	0.15º:10.6%	0.15º:24.3%	0.15º:12.3%	0.15º:36.2%
TC Output	1.0º:70.5%	0.2º:32.3%	1.0º:82.4%	0.2º:48.3%
Voltage Measurement	0.166º:11.7%	0.166º:26.8%	0.052º:4.3%	0.052º:12.6%
Noise	0.1º:7%	0.1º:16.2%	0.01º:0.8%	0.01º:2.4%
Reference Linearization	0.001º:0.1%	0.001º:0.2%	0.001º:0.1%	0.001º:0.25%
Output Linearization	0.001º:0.1%	0.001º:0.2%	0.001º:0.1%	0.001º:0.25%
Total Error	1.418º:100%	0.618º:100%	1.214º:100%	0.414º:100%

3.1.4.2 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 CR9000. 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 CR9000. Alternatively, the junction box can be used to couple extension grade thermocouple wire to the thermocouples being used for measurement, and the CR9000 I/O Module used as the reference junction. 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 CR9000. This is only a factor when using type K thermocouples, where the upper limit of the reference compensation linearization is 100 ºC and the upper limit of the extension grade wire is 200 ºC. With the other types of thermocouples the reference compensation range equals or is greater than the extension wire range. In any case, errors can arise if temperature gradients exist within the junction box.

Figure 3.4-1 illustrates a typical junction box. Terminal strips will be a different metal than the thermocouple wire. Thus, if a temperature gradient exists between A and A' or B and B', the junction box will act as another thermocouple in series, creating an error in the voltage measured by the CR9000. This thermoelectric offset voltage is a factor whether or not the junction box is used for the reference. This offset can be minimized by making the thermal conduction between the two points large and the distance small. The best solution in the case where extension grade wire is being connected to thermocouple wire would be to use connectors which clamped the two wires in contact with each other.

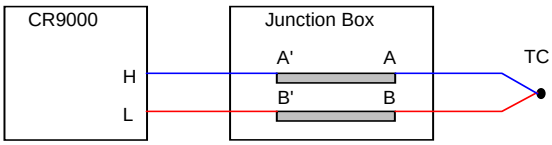


FIGURE 3.4-1. Diagram of Junction Box

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

Radiation shielding must be provided when a junction box is installed in the field. Care must also be taken that a thermal gradient is not induced by conduction through the incoming wires. The CR9000 can be used to measure the temperature gradients within the junction box.

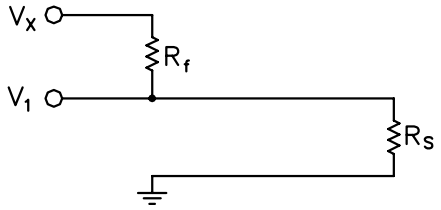
3.1.5 Bridge Resistance Measurements

There are four bridge measurement instructions included in the standard CR9000 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 until the hold for the analog to digital conversion is completed. When more than one measurement per sensor is necessary (four wire half bridge, three wire half bridge, six wire full bridge), excitation is applied separately for each measurement. For example, in the four wire half bridge when the excitation is reversed, the differential measurement of the voltage drop across the sensor is made with the excitation at both polarities and then excitation is again applied and reversed for the measurement of the voltage drop across the fixed resistor.

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

BrHalf



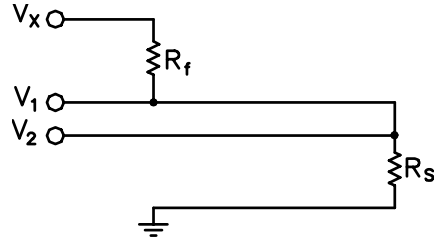
X = result w/mult = 1, offset = 0

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

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

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

BrHalf3W



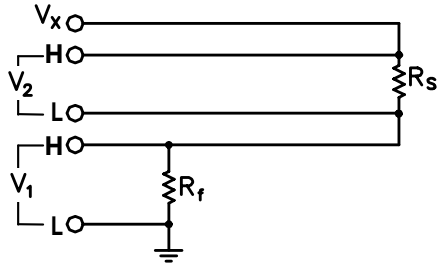
X = result w/mult = 1, offset = 0

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

$$R_s = R_f X$$

$$R_f = R_s / X$$

BrHalf4W



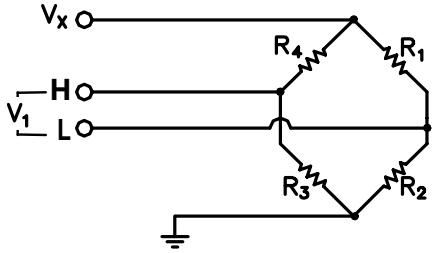
X = result w/mult = 1, offset = 0

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

$$R_s = R_f X$$

$$R_f = R_s / X$$

BrFull



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

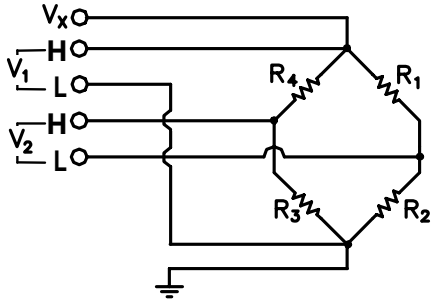
$$X = 1000 \frac{V_1}{V_x} = 1000 \left(\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2} \right)$$

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

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

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

BrFull6W



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

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

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

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

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

FIGURE 3.5-1. Circuits Used with Bridge Measurement Instructions

3.1.6 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 50 kHz, the excitation is switched on and then reversed 10 µs later when the first measurement is held and then is switched off after another 10 µs when the second measurement is held (i.e., reverse the excitation, 10 µs delay, no integration). A switched excitation channel (7-16 on the 9060 Module) should be used when AC excitation is required because it will be switched out as soon as the measurement is completed. The continuous excitation channels (1-6 on the 9060 Module) should not be used because they retain the last voltage programmed (i.e., after reversing the excitation, the channel would be left at the reversed polarity voltage until the next instruction that acted on the excitation channel).

3.1.7 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 CR9000 ground, and can be represented by the model diagrammed in Figure 3.6-1.

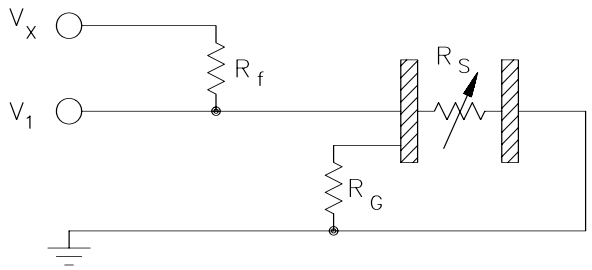


FIGURE 3.6-1. Model of Resistive Sensor with Ground Loop

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 CR9000 earth ground. With R_G in the network, the measured signal is:

$$V_1 = V_X \frac{R_S}{(R_S + R_f) + R_S R_f / R_G} \quad [3.6\text{-}1]$$

$R_S R_f / R_G$ is the source of error due to the ground loop. When R_G is large the equation reduces to the ideal. The geometry of the electrodes has a great effect on the magnitude of this error. The Delmhorst gypsum block used in the 227 probe has two concentric cylindrical electrodes. The center electrode is used for excitation; because it is encircled by the ground electrode, the path for a ground loop through the soil is greatly reduced. Moisture blocks which consist of two parallel plate electrodes are particularly susceptible to ground loop problems. Similar considerations apply to the geometry of the electrodes in water conductivity sensors.

The ground electrode of the conductivity or soil moisture probe and the CR9000 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.

3.2 CR9058E Isolation Module Measurements

Each CR9058E input channel has its own 24 bit sigma delta analog to digital converter taking 10,000 measurements per second, or one measurement sample per 100 microseconds. The effective resolution at this sample rate is 18.7 bits, or +/- 9.4 microvolts when using the +/- 2 Volt range, because of the inherent noise of the A/D converter and noise from other sources. The effective resolution can be dramatically improved through filtering, and/or integrating, multiple measurements. Thus, noise reduction and measurement speed can be traded off using the Integration parameter. Noise is reduced by approximately the square root of the number of samples within the integration time. Thus, if the integration time is set to 10000 versus 100 microseconds, noise should be reduced approximately by a factor of ten . This approximation assumes that the noise is white noise, which is not entirely true because some of the noise is due to interference from sources at fixed frequencies. Noise reduction by filtering can go just so far, and the best the CR9058E can achieve is approximately 21 bits of resolution (+/- 1.9 microvolts on the 2 Volt range).

The CR9058E isolated input module is similar in operation to the CR9050 analog input module except for:

- The CR9058E has ten differential input channels instead of 16 differential / 32 single-ended inputs.
- The CR9058E has different voltage ranges of +/- 60 Volts DC, +/- 20 Volts DC, and +/- 2 Volts DC.
- The CR9058E has a slower maximum scan rate than the CR9050, but this is somewhat balanced by the fact that the CR9058E measures all of its channels simultaneously, as each channel has its own 24 bit sigma delta analog to digital converter. Conversely, the measurements from the

CR9050(E) are multiplexed sequentially through a single A to D converter.

3.2.1 CR9058E Supported Instructions

The CR9058E currently supports three CR9000X measurement instructions:

1. VoltDiff (Dest, Reps, Range, ASlot, DiffChan, RevDiff, Settle, Integ, Mult, Offset)
2. TCDiff (Dest, Reps, Range, ASlot, DiffChan, TCType, TRef, RevDiff, Settle, Integ, Mult, Offset)
3. ModuleTemp (Dest, Reps, ASlot, Integ)

These instructions operate the same as with the CR9050 with these differences:

- DiffChan must be within 1..10.
- VoltDiff supports these voltage ranges: V2 (+/- 2 Volts DC), V2C (+/- 2 Volts with open thermocouple checking), V20 (+/- 20 Volts DC), and V60 (+/- 60 Volts DC).
- TCDiff will work with the same range settings as the VoltDiff instruction, but only V2 (no open thermocouple checking) or V2C (+/-2 volt range with open thermocouple checking) should be used with TCDiff. When the range is set = V2C, an open circuit will report an over-range condition to the CR9000X.
- The Settle time parameter is unused.
- The minimum scan time when using the VoltDiff instruction, without input reversal, for the CR9058E is 1290 microseconds for integration times under 200 microseconds. If the integration time is greater than 200 microseconds, then the minimum scan interval is 1090 + integration time (microseconds). When using the VoltDiff instruction with input reversal and integration under 200 microseconds, the minimum scan interval is 2990 microseconds. With input reversal and integration times greater than 200 microseconds, the minimum scan time is (2790 + (2 x integration time)).
- The minimum scan time when using the TCDiff instruction, without input reversal, for the CR9058E is 2570 microseconds for integration times under 200 microseconds. If the integration time is greater than 200 microseconds, then the minimum scan interval is 1370 + integration time (microseconds). When using the VoltDiff instruction with input reversal and integration under 200 microseconds, the minimum scan interval is 4280 microseconds. When using input reversal and integration times greater than 200 microseconds, the minimum scan time is (4080 + (2 x integration time)).

- The Integ parameter in VoltDiff and in TCDiff (not in ModuleTemp) can be set to –1, -2, -3, -4, or -5 and the CR9058E will set the corresponding synch filter of 1, 2, 3, 4, or 5. The integration time will be maximized for the given synch filter and scan interval. The integration and synch filter order that a given CR9058E is using can be seen through PC9000's terminal mode window (Tools/Diagnostics/Terminal Mode). In the I/O Port area, click on "Open Port". Next, click in the Low Level I/O section and hit Enter several times until the CR9000> prompt is returned. Type in "4" and enter. The CR9058Es' slot numbers, integration times, and sync filters will be returned.

- **Input reversal (for offset cancellation) isn't individually selectable within the ten channels of a CR9058E module. If any one channel of a CR9058E's ten input channels has input reversal selected, by setting the Rev parameter of the VoltDiff or TCDiff instruction to true, input reversal will be applied to all ten channels. If other VoltDiff or TCDiff instructions tied to this module within the same scan don't have the Rev parameter set True, then, without generating an error, input reversal will be applied to them all anyway.**

- A CR9058E can only have one integration time per scan interval that applies to all ten of its channels. If multiple measurement instructions within a scan are tied to a single CR9058E module, and they don't all have the same Integ time parameter, then the Integration time for all of that module's channels will be set by the value of the Integ time parameter of the last measurement instruction, tied to that module, within the scan.

- The Integ parameter in the VoltDiff and TCDiff instructions, within the constraints listed above, can be used to adjust the measurement frequency response. For example, for both 60 Hz and 50 Hz rejection the Integ parameter could be set = 100,000 microseconds.

- The CR9058E ModuleTemp measurement is independent of the isolated input measurements. The CR9058E ModuleTemp measurement method is identical to that of the CR9050, using a platinum resistance thermometer to obtain the thermocouple reference junction temperature at the EZ-connect terminal module.

- Because heat is generated within the CR9058E, a thermal gradient can develop across the EZ-connect terminal block which can produce errors in thermocouple measurements. To minimize this error, keep the CR9058EC covers in place. Also, type E or K thermocouples are better than type T because type T thermocouples have a copper conductor which is an excellent conductor of heat increasing the thermal gradient across the terminal block.

- Each channel has an H (high) input terminal, a L (low) terminal, and a G (isolated ground) terminal. The isolated ground terminals are not connected to the CR9000X system ground. The isolated ground terminal can be used to connect the shield of a shielded cable. Also, when un-shielded thermocouples are used, the G terminal can be tied to the H or L terminal to reduce noise in the readings.

- The CR9058E does not directly support Bridge measurements, but Bridge type measurements can be performed through using the CR9060s CAOs or external excitation and adjusting the multiplier according to the excitation level.

3.2.2 CR9058E Sampling, Noise and Filtering

The ten analog to digital converters are re-synchronized at the beginning of each scan. There are 1290 microseconds of over-head associated with this process and other tasks. Therefore the minimum scan period for the CR9058E is 1290 microseconds when setting the integration parameter to 200 or less (two measurement samples 100 microseconds apart per channel is attained at this rate). The integration time (microseconds) divided by 100 determines the number of measurements taken during a scan. Setting the integration parameter to a value greater than 200 microseconds requires a minimum scan interval of 1090 + integration time. If reverse measurement is set true, then the over-head is 2990 microseconds. With reversal on, and setting the integration parameter to a value of 200 or less, a minimum scan interval of 2990 microseconds will be required.

The CR9058E has a digital signal processor that performs "sync-n" filtering of the analog to digital converter results to reduce noise. At compile time the CR9058E computes the order of the sync-n filter based on the integration time and Scan interval. The more samples available, the higher the order of sync-n filter is implemented up to an order of five. The equation used to calculate the filter is:

$$filterorder = \frac{(AvailTime - SampleTime)}{(IntegTime - SampleTime)}$$

where:

AvailTime = Scan Interval with the following adjustments:

Subtract off 1290 microseconds if range code v2C is used.

Divide by 2 and subtract off 420 microseconds if input reversal is true.

Finally, subtract another 1090 microseconds

IntegTime = user entered Integration time in microseconds.

SampleTime = 100 (microseconds) or
SampleTime = N X 100 (microseconds). See below.

Where N is the number of integrated values that are averaged together before the sync-n filter is applied. Initially N is set to 1 (no averaging of the integrated values). If the CR9058E finds that it doesn't have enough memory to implement the filter, it increments N above and repeats the calculation until the filter will fit in its memory.

If the Integration time parameter is set = -1, the IntegTime above is set = AvailTime. This gives a filter order of 1, which amounts to simple averaging of all the samples measured in a scan interval.

CHART 3.1
FREQUENCY RESPONSE OF SYNC FILTER ORDERS 1 - 5

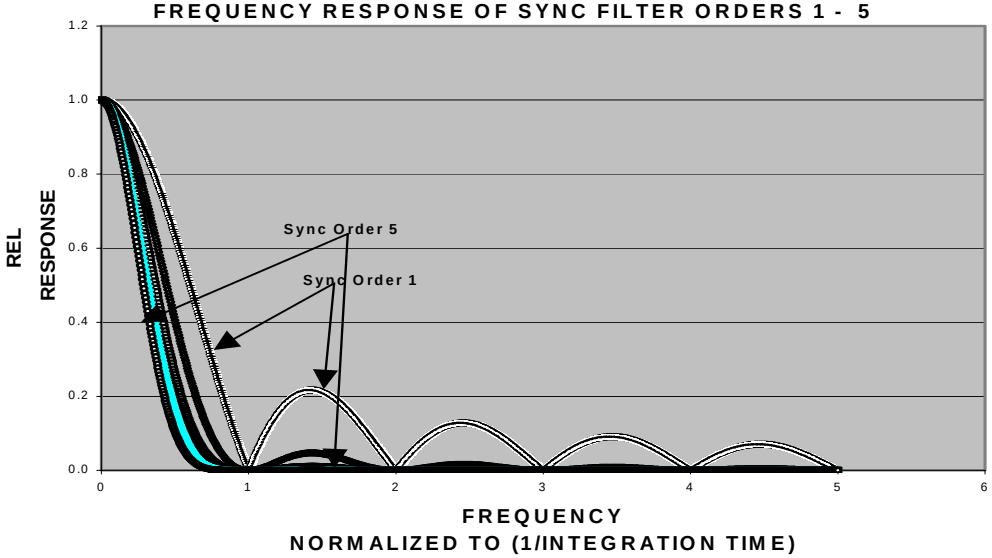
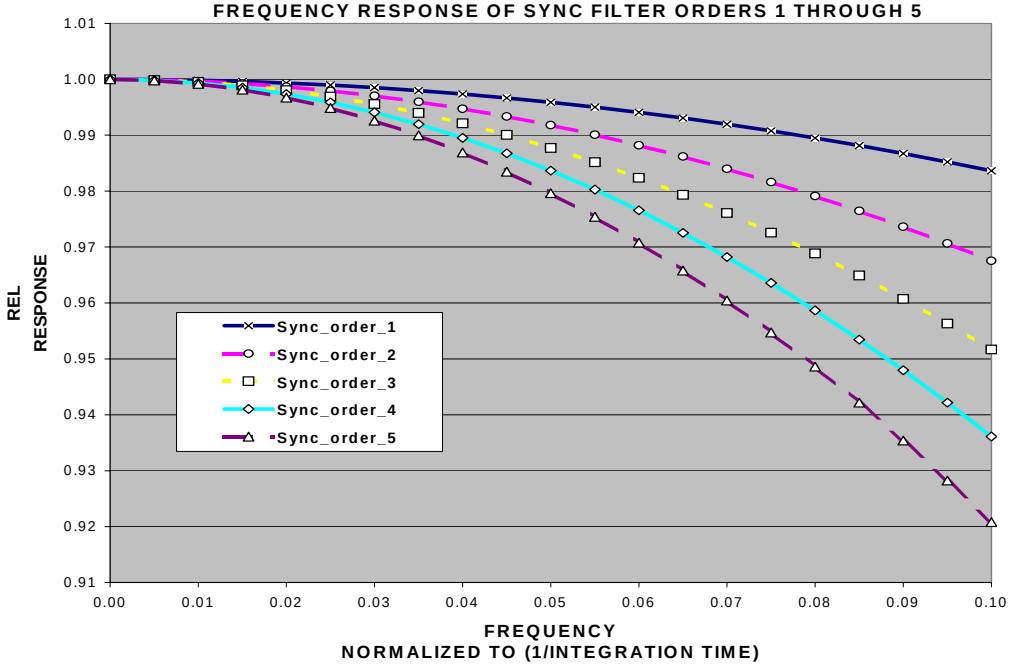


Chart 3.1 shows the response times for the synch filters available for the CR9058E. As can be seen, the 1rst order sync filter does not filter out the higher frequency components of the input signal. This could result in higher frequency signals being aliased back to lower frequencies. While the 5th order sync filter does a fairly good job filtering out higher order frequencies, the trade off is that it also attenuates the signal at lower frequencies as can be seen in Chart 3.2.

CHART 3.2
FREQUENCY RESPONSE OF SYNC FILTER ORDERS 1 THROUGH 5



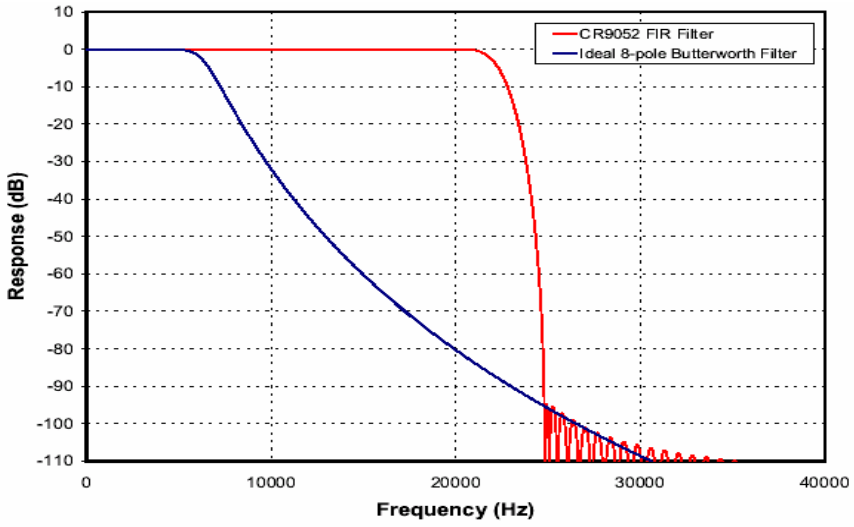
3.3 CR9052 Filter Module Measurements

Each CR9052 module has six differential analog measurement channels with programmable input ranges from ±20 mV to ±5 V. Each channel has its own programmable-gain instrumentation amplifier, pre-sampling analog filter, and sigma-delta analog-to-digital converter. All CR9052 channels in a single CR9000X chassis are sampled simultaneously.

The CR9052 implements anti-aliasing with programmable, real-time, low-pass, finite impulse response (FIR) filters. An on-board digital signal processor (DSP) collects alias-free, 50-kHz samples from each of the module's sigma-delta converters, and then applies real-time, programmable low-pass filtering and decimation to anti-alias and down-sample the data to the selected measurement rate, selectable from 5 Hz to 50 kHz. The CR9052 can also accumulate snapshots of anti-aliased time-series, Fourier transform them into frequency spectra, and send the resulting real-time spectra to the CR9000X's main processor.

The CR9052 can burst measurements to its on-board, 8-million sample buffer at 50,000 measurements per second per channel. Using the FFT spectrum analyzer mode, the module's DSP can provide real-time spectra from "seamless", anti-aliased, 50-kHz, 2048-point time-series snapshots for each of its six analog input channels. The decimated data can be downloaded to an appropriate PC card at an aggregate rate of 100,000 measurements per second.

The CR9052 filter's pass-band ripple is less than ±0.01 dB (0.1 percent), and the stop-band attenuation exceeds 90 dB (1/32,000). The FIR filter's transition band has a steep roll-off, with the stop-band frequency starting a factor of 1.24 above the pass-band frequency. In comparison, the stop-band frequency of an ideal eight-pole Butterworth filter with the same ripple and attenuation starts a factor of 5.81 above its pass-band frequency.



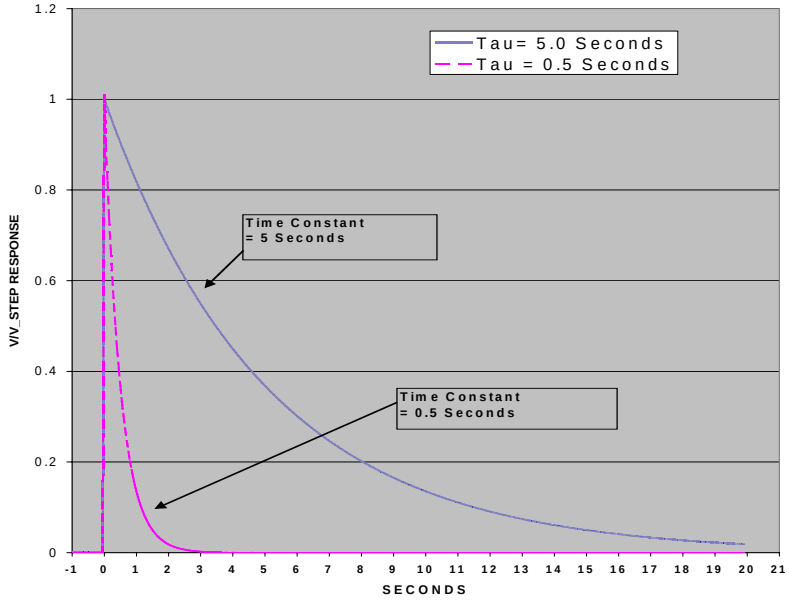
The digital implementation of the CR9052 FIR filters maintains a group delay that is independent of frequency (linear phase response). In addition, the digital filter performance does not change with time, temperature, or component tolerances. The on-board DSP automatically chooses the

appropriate low-pass filter to anti-alias the input data for the user's desired measurement rate. If desired, users may load their own coefficients into the on-board DSP to tailor the FIR filter's frequency response to their own needs (band pass, band reject, etc.).

CR9052IEPE DC Frequency Response

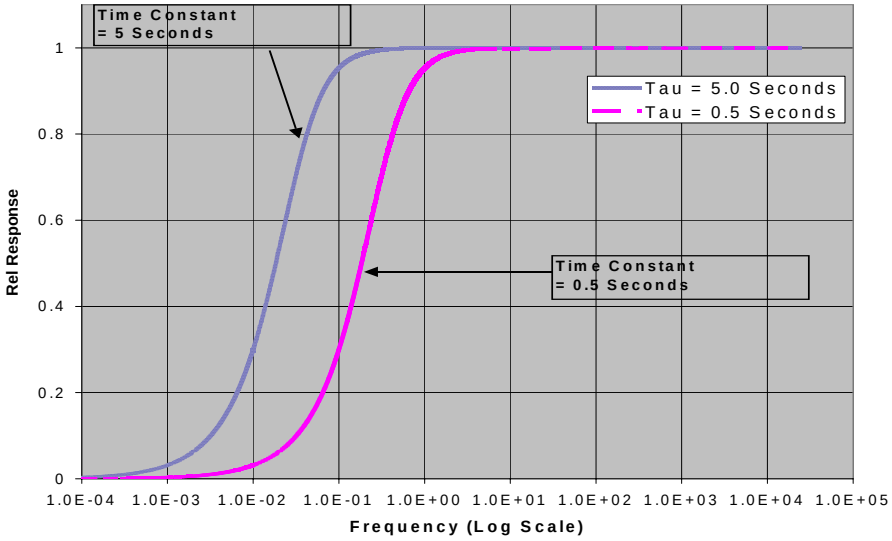
The CR9052IEPE module has two programmable time constants available: 5 seconds and 0.5 seconds. The advantage of the 0.5 second time constant is that if you have a step in the voltage (either from a shock to the sensor or when initially supplying excitation) it will only take 0.5 seconds for 63% of the voltage step to discharge, while with the 5 second time constant, it would take 5 seconds.

Chart 3.3 Step Discharge Rate



The 5.0 second time constant will not result in lower frequencies being attenuated as much (3 dB at 0.03 Hz) as the 0.5 second time constant (3 dB at 0.3 Hz).

Chart 3.4 Frequency Response



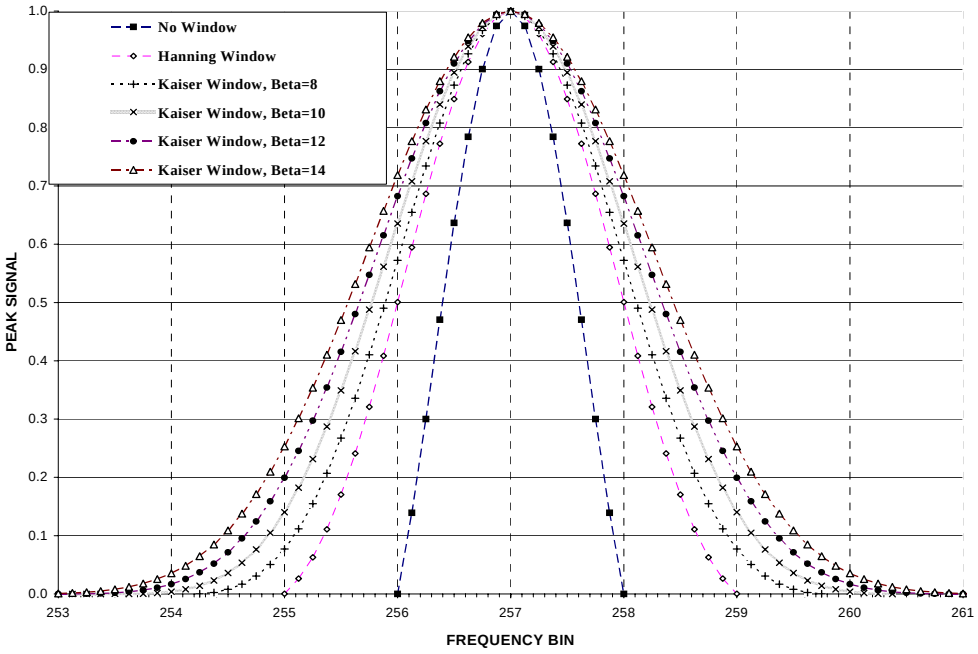
WINDOWING

The FFT option allows radix-two (2^n, where n = 5, 6, …16) transform lengths ranging from 32 to 65,536 samples. Users can optionally apply a Hanning, Hamming, Blackman-Harris, or one from a selection of Kaiser-Bessel beta choices, window function to their time series before transforming them. The beta (Kaiser-Bessel) allows the user to trade spectral leakage for spectral resolution. Table 3.3-1 shows the maximum out of band leakage and the full width, half of maximum (FWHM) spectral resolution monitoring a monochromatic signal using four different betas. Chart 3.5-1 shows graphically the bin resolution (or bin smearing effect) for no windowing, the Hanning window and 4 Kaiser-Bessel betas.

Table 3.3-1. Spectral Leakage vs. Resolution

BETA	MAXIMUM LEAKAGE (dB)	SPECTRAL RESOLUTION (BINS)
8	-63	2.25
10	-74	2.50
12	-95	2.75
14	-110	3.00

CHART 3.5 COMPARISION OF SPECTRAL RESOLUTION FOR VARIOUS WINDOWING FUNCTIONS



Using a Kaiser-Bessler with a beta of around 12 results in a spectral leakage that best matched the attenuation of the CR9052's anti-aliasing filters. Although this spreads the FWHM of a single line source to 2.75 bins, this can be compensated for by increasing the length (or number of bins) of the FFT because the windowing spreads the signal across a finite number of bins, not across an absolute frequency range.

SPECTRAL OUTPUT

The CR9052 offers a variety of spectrum normalizations, including real and imaginary, amplitude and phase, power, power spectral density (PSD), and decibels (dB). In addition, the CR9052 can combine adjacent spectral bins into a single bin to decrease the size of the final spectrum. A built-in function selects an exponentially increasing spectral bin width to give 1/n octave analyses, where n can vary from 1 to 12. A single programming step with either the CRBasic programming language or the CR9000X program generator configures the FFT spectrum analyzer options.

The module has superior noise performance, with an input-referred noise of 8 nV $Hz^{-1/2}$ for the ± 20 mV input range. On the ± 20 mV input range, the total noise for a 20 kHz bandwidth is less than 1.4 uV, and for a 1 Hz bandwidth, 250 nV. The programmable anti-alias filter allows users to trade bandwidth for noise, or vice versa. The DSP's floating-point numeric implementation of the FIR anti-alias filters and Fourier transforms preserve this low-noise performance. A 2048-point FFT gives an instantaneous dynamic range exceeding 126 dB (an amplitude ratio of $2x10^6$), and the 65,536-point FFT gives an instantaneous dynamic range exceeding 140 dB (an amplitude ratio of $1x10^7$). Real-time digital temperature compensation ensures gain accuracy (±0.03 percent of reading) and offset accuracy (±0.03 percent of full-scale) throughout the -40° to 70° C operating temperature range.

The combined capabilities of the CR9052 and the CR9000X offer numerous measurement and data processing possibilities. For example, this combination allows users to mix high-speed, anti-aliased measurements and spectra from accelerometers, strain gages, and microphones with slower measurements from thermocouples, pressure transducers, and serial data streams. The general-purpose programmability of the CR9000X allows users to process their data before saving it to data tables. For example, users may save measured data only if the amplitude of a specific acoustic frequency exceeds some threshold, or only if an acoustic spectral component correlates to measurements from other sensors.

3.4 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 Transputer 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 would still be output.

The resolution of the pulse counters is one count. The resolution of the calculated frequency depends on the scan interval: frequency resolution = 1/scan interval (e.g., a pulse count in a 1 second scan has a frequency resolution of 1 Hz, a 0.5 second scan gives a resolution of 2 Hz, and a 1 ms scan gives a resolution of 1000 Hz). The resultant measurement will bounce around by the resolution. For example, if you are scanning a 2.5 Hz input once a second, in some intervals there will be 2 counts and in some 3 as shown in Figure 3.4-1. If the pulse measurement is averaged, the correct value will be the result.

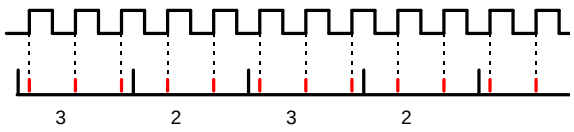


FIGURE 3.4-1. Varying Counts within Pulse Interval

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

The POption parameter in the PulseCount instruction can be used to set an interval period for a running average computation of the frequency output from the sensor. Example: Scan Rate of 10 mSec is for other measurements. The output from the Pulse sensor will vary from 1000 Hz to 10 Hz. Set the POption parameter to 1000 (mSec), and the instruction returns get a running average of the Pulse outputs (getting 100 samples/second) over a 1 second period. This would smooth the output.

Another method for measuring frequency is to use the TimerIO instruction using one of the Pulse channels on the CR9071E Pulse Module to measure the period of the signal (40 nanosecond resolution). The value returned is in milliseconds. You can take the reciprocal of the returned value and multiply by 1000 to get the frequency of the signal.

3.4.1 High Frequency Pulse Measurements

All twelve pulse channels of the CR9070 and CR9071E can be configured for high frequency inputs. The signal is feed through a filter with a time constant of 200 (τ = 200 nanoseconds) nanoseconds to remove higher frequency noise. It is then feed through a Schmitt circuit to convert the signal to a square wave, and to guard against false triggers when the signal is hovering around the threshold level. In the High Frequency mode, the input signal to the Schmitt

trigger must rise from below 1.5 volts to above 3.5 volts in order to trigger an output. Due to the attenuation caused by the filter on the front side of the Schmitt circuit, a larger input voltage transition is required for higher frequencies. The transition required for the input of the Schmitt trigger can be viewed as 2.5 volts ± 1 volt (from below 1.5 volt to above 3.5 volt). The equation to calculate the amount that the signal is attenuated by the front end filter is:

$$\frac{V_{Out}}{V_{In}} = \sqrt{\frac{1}{\left(1+\left((2\pi\tau)f\right)^2\right)}}$$

V_{Out} is the voltage level leaving the filter (level into the Schmitt circuit) when V_{In} is the input voltage. V_{Out} must be at minimum 1 volt for the Schmitt circuit to trigger an output.

Chart 3.6 Required Transition Voltage for High Frequency Pulse

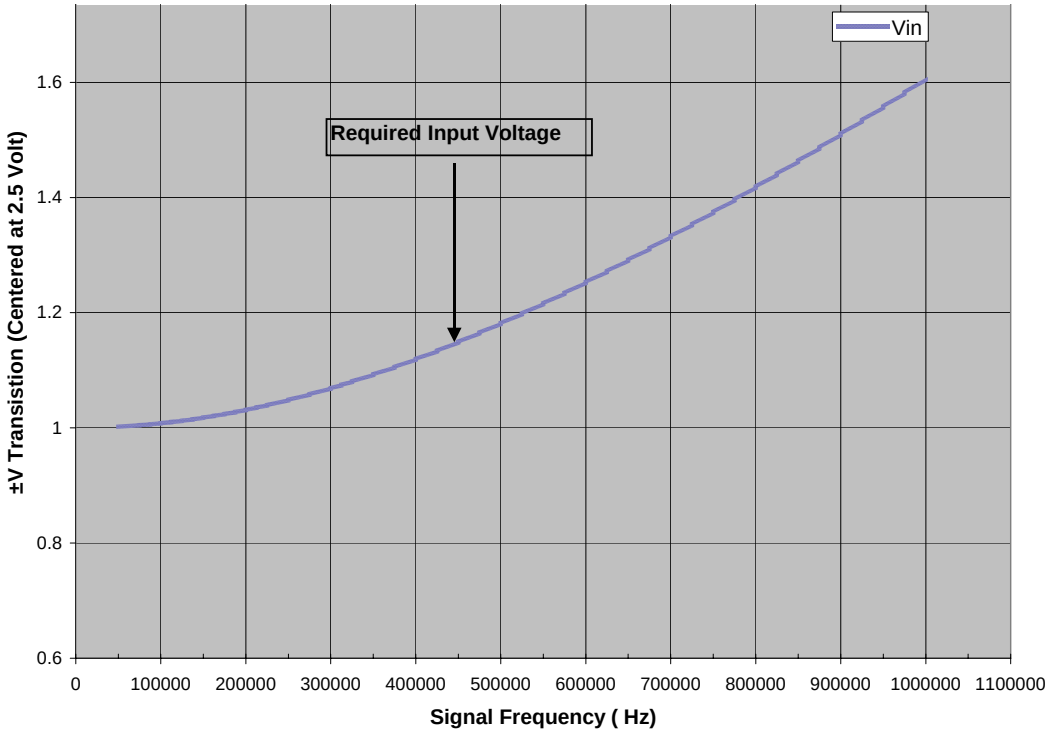


Chart 3.6 plots the trace for the minimum transition voltage about 2.5 volts against the input signal frequency. To demonstrate how to use this plot, for a input frequency of 1 MHz, the voltage signal, centered about 2.5 volts, must have a transition of ± 1.6 volts in order to trigger the Schmitt circuit. In other words, the signal must rise from below 0.9 volts (2.5 volts minus 1.6 volts) to above 4.1 volts (2.5 volts plus 1.6 volts) for a pulse to be counted.

Section 4. CRBasic – Native Language Programming

The CR9000 is programmed in a language that has some similarities to a structured basic. There are special instructions for making measurements and for creating tables of output data. The results of all measurements are assigned variables (given names). Mathematical operations are written out much as they would be algebraically. This section describes a program, its syntax, structure, and sequence.

4.1 Format Introduction

4.1.1 Mathematical Operations

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

TempF = TempC * 1.8 + 32

With the CR9000 there may be 5 or 50 temperature (or other) measurements. Rather than have 50 different names, a *variable array* with one name and 50 elements may be used. A thermocouple temperature might be called TCTemp. With an array of 50 elements the names of the individual temperatures are TCTemp(1), TCTemp(2), TCTemp(3), ... TCTemp(50). The array notation allows compact code to perform operations on all the variables. For example, to convert ten temperatures in a variable array from C to F:

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

4.1.2 Measurement and Output Processing Instructions

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

The instructions for making measurements and outputting data are not found in a standard basic language. The instructions Campbell Scientific has created for these operations are in the form of procedures. The procedure has a keyword name and a series of parameters that contain the information needed to complete the procedure. For example, the instruction for measuring the temperature of the RTD on the 9050 Analog Input Module is:

ModuleTemp(*Dest, Reps, ASlot, Integ*)

ModuleTemp is the keyword name of the instruction. There are four parameters associated with ModuleTemp are: *Dest*ination, the name of the variable in which to put the temperature; *Rep*etitions, the number of sequential 9050 modules to measure the temperature of; *Aslot*, the slot in the CR9000 that the analog card is in; and *Integ*ration, the length of time to integrate the measurement. To place the temperature of the analog module in slot 5 in the variable RefTemp (using a 10 microsecond measurement integration time) the code is:

```
ModuleTemp(RefTemp, 1, 5, 10)
```

The use of these instructions should become more clear 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 CR9000 code. When the CR9000 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 CR9000 program:

Declarations	*Make a list of what to measure and calculate.*
Declare constants	*Within this list, include the fixed constants used,*
Declare Public variables	*indicate the values that the user is able to view while the program is running,*
Dimension variables	*the number of each measurement that will be made,*
Define Aliases	*and specific names for any of the measurements.*
Define data tables.	*Describe, in detail, tables of data that will be saved from the experiment.*
Process/store trigger	*Set when the data should be stored. Are they stored when some condition is met? Are data stored on a fixed interval? Are they stored on a fixed interval only while some condition is met?*
Table size	*Set the size of the table in CR9000 RAM*
Other on-line storage devices	*Should the data also be sent to PC card or Flash memory?*
Processing of Data	*What data are to be output (current value, average, maximum, minimum, etc.)*
Define Subroutines	*If there is a process or series of calculations that need to be repeated several times in the program, it can be packaged in a subroutine and called when needed rather than repeating all the code each time.*

Program	*The program section defines the action of datalogging*
Set scan interval	*The scan sets the interval for a series of measurements*
Measurements	*Enter the measurements to make*
Processing	*Enter any additional processing with the measurements*
Call Data Table(s)	*The Data Table must be called to process output data*
Initiate controls	*Check measurements and Initiate controls if necessary*
NextScan	*Loop back (and wait if necessary) for the next scan*
End Program	

4.3 Example Program

```
Const RevDiff 1
Const Del 0
Const Integ 0
Const Mult 1
Const Offset 0
Public RefTemp
Public DIM TC(6)
Units RefTemp=degC
Units TC=degC

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

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

Declare constants

Declare public variables , dimension array, and declare units.

Define Data Table

Declarations

Measure

Call Data Table

Scan loop

4.3.1 Data Tables

Data storage follows a fixed structure in the CR9000 in order to optimize the time and space required. Data are stored in tables such as:

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

The user's program determines the values that are output and their sequence. The CR9000 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 output 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 CR9000 to fill this row out (e.g., Units RefTemp = degC). The units are strictly for the user's documentation; the CR9000 makes no checks on their accuracy.

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

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

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

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

The DataTable instruction has three parameters: a user specified name for the table, a trigger condition, and the size to make the table in CR9000 RAM. The trigger condition may be a variable, expression, or constant. The trigger is true if it is not equal to 0. Data are output if the trigger is true and there are no other conditions to be met. No output occurs if the trigger is false (=0). The example creates a table name Temp, outputs any time other conditions are met, and retains 2000 records in RAM.

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

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

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

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

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

TABLE 4.3-1 Formats for Output Data

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

4.3.2 The Scan -- Measurement Timing and Processing

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

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

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

Scan*(Interval, Units, BurstOption, Count)*
Scan(1,MSEC,0,0)

The Scan instruction has four parameters. The Interval is the interval between scans. Units are the time units for the interval. The maximum scan interval is one minute. The BurstOption determines if processing is concurrent with measurements (0) or if raw values are buffered in RAM (1) or a PCMCIA card (2), for processing after all measurements are made. At the maximum measurement rate, processing time may exceed the time required for measuring. Count is the number of scans to make before proceeding to the instruction following NextScan. A count of 0 means to continue looping forever (or until ExitScan). In the example the scan is 1 millisecond, measurements are processed as they are made, and the measurements and output continue indefinitely.

4.4 Numerical Entries

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

TABLE 4.4-1 Formats for Entering Numbers in CRBasic

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

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

4.5 Logical Expression Evaluation

4.5.1 What is true?

Several different words get used to describe a condition or the result of a test. The expression, X>5, is either **true** or **false**. However, when describing the state of a port or flag, **on** or **off** or **high** or **low** sounds better. In CRBasic there are a number of conditional tests or instruction parameters the result of which may be described with one of the words in Table 4.5-1. The CR9000 evaluates the test or parameter as a number; 0 is false, not equal to 0 is true.

TABLE 4.5-1. Synonyms for True and False

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

4.5.2 Expression Evaluation

Conditional tests require the CR9000 to evaluate an expression and take one path if the expression is true and another if the expression is false. For example:
If X>=5 then Y=0
will set the variable Y to 0 if X is greater than or equal to 5.

The CR9000 will also evaluate multiple expressions linked with **and** or **or**. For example:
If X>=5 and Z=2 then Y=0
will only set Y=0 if both X>=5 and Z=2 are true.
If X>=5 or Z=2 then Y=0
will set Y=0 if either X>=5 or Z=2 is true (see And and Or in Section 9). A condition can include multiple **and** and **or** links.

4.5.3 Numeric Results of Expression Evaluation

The CR9000 expression evaluator evaluates an expression and returns a number. A conditional statement uses the number to decide which way to branch. The conditional statement is false if the number is 0 and true if the number is not 0. For example:
If 6 then Y=0,
is always true, Y will be set to 0 any time the conditional statement is executed.
If 0 then Y=0
is always false, Y will never be set to 0 by this conditional statement.

The CR9000 expression evaluator evaluates the expression, X>=5, and returns -1, if the expression is true, and 0, if the expression is false.
W=(X>Y)
will set W equal to -1 if X>Y or will set W equal to 0 if X<=Y.

The CR9000 uses -1 rather than some other non-zero number because the **and** and **or** operators are the same for logical statements and binary bitwise comparisons (see **and** and **or** in Section 8). The number -1 is expressed in binary with all bits equal to 1, the number 0 has all bits equal to 0. When -1 is anded with any other number the result is the other number, ensuring that if the other number is non-zero (true), the result will be non-zero

4.6 Flags

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

4.7 Parameter Types

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

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

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

TABLE 4.7-1. Rules for Names

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

4.7.1 Expressions in Parameters

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

DataTable(Name, TrigVar, Size)
DataTable(Temp, TC(1)>100, 5000)

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

4.7.2 Arrays of Multiplier Offsets for Sensor Calibration

If variable arrays are used as the multiplier and offset parameters in measurements that use repetitions, the instruction will automatically step through the multiplier and offset arrays as it steps through the channels. This allows a single measurement instruction to measure a series of individually calibrated sensors, applying the correct calibration to each sensor. If the multiplier and offset are not arrays, the same multiplier and offset are used for each repetition.

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

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

4.8 Program Access to Data Tables

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

Tablename.Fieldname(fieldname index,records back)

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

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

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

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

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

Tablename.Tablesize(1,1) = the size of the table in records.

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

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

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

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

Tablename.EventCount(1,1) is also only valid for a data table using the DataEvent Instruction.

Tablename.EventCount(1,1) = the number of events that have been completed in the table. An event is complete when the table has stopped storing data for the event.

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

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

Section 5. Program Declarations

ALIAS

Used to assign a second name to a variable.

Syntax
Alias *VariableA* = *VariableB*

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

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

Alias Declaration Example

The example shows how to use the Alias declaration.

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

CONST

Declares symbolic constants for use in place of values.

Syntax
Const *constantname* = *expression* [, *constantname* = *expression*] . . .

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

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

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

Caution Constants must be defined before referring to them.

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

Const Declaration Example

The example uses Const to define the symbolic constant PI.

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

DIM

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

Syntax

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

Remarks

The **Dim** statement has these parts:

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

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

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

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

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

When variables are initialized, they are initialized to 0

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

PUBLIC

Dimensions a variable as public and available in the Public table of the CR9000.

Syntax

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

Remarks

More than one Public statement can be made.

Public Declaration Example

The example shows the use of the Public declaration.

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

STATION NAME

Sets the station name.

Syntax
StationName StaName

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

UNITS

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

Syntax
Units *Variable* = UnitName

Remarks
Units allows assigning a unit name to a field. Units are displayed on demand in the real-time windows of PC9000. The unit name also appears in the header of the output files and in the Data Table Info file of PC9000. The unit name is a text field that allows the user to label data. When the user modifies the units, the text entered is not checked by PC9000 or the CR9000.

Example
Dim TCTemp(1)

Units TCTemp(1) = Deg_C

SUB, EXIT SUB, END SUB

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

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

The Sub statement has these parts:

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

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

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

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

Subroutine Example

```
'CR9000
''Declare Variables used in Program:
Public RefT, TC(4),I

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

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

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

BeginProg
   Scan (1,Sec,3,0)
      'Measure Temperatures (module + 4 thermocouples) in deg C
         ModuleTemp (RefT,1,1,250)
      TCDiff (TC(),4,mV50C,1,1,TypeT,RefT,True ,0,250,1.0,0)
         'Call Output Table for C
      CallTable TempsC
         'Convert Temperatures to F using Subroutine:
      Call ConvertCtoF(RefT)          'Subroutine call using Call statement
      For I = 1 to 4
         ConvertCtoF(TC(I))           'Subroutine call without Call statement
      Next I
      'Call Output Table for F:
      CallTable TempsF
   NextScan
EndProg
```

Section 6. Data Table Declarations and Output Processing Instructions

6.1 Data Table Declaration

DataTable (Name, TrigVar, Size)

output trigger modifier
export data destinations
output processing instructions
EndTable

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

Parameter *& Data Type*	**Enter** **DataTable Parameters**
Name *Name*	The name for the data table. The table name is limited to eight characters.
TrigVar *Constant Variable, or Expression*	The name of the variable to test for the trigger. Trigger modifiers add additional conditions. **Value** — **Result** 0 — Do not trigger ≠ 0 — Trigger
Size *Constant*	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 native language 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 CR9000 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 CR9000 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 CR9000 allocates some extra memory for the table, if there are a lot of lapses, it is not possible to store as many records as requested in the DataTable declaration. The **Lapses** parameter allows the programmer to allocate additional space for the number of lapses entered. This is used in particular when the program is written in a way that will create lapses. For example, if the data output is controlled by a trigger (e.g., a user flag) in the DataTable instruction in addition to the DataInterval, lapses would occur each time the trigger was false for a period of time longer than the interval.

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

Entering a negative number tells the CR9000 not to keep track of lapses. Only the periodic time stamps (approximately once per K of data) are inserted.

Parameter *& Data Type*	**Enter** **DataInterval Parameters**
TintoInt *Constant*	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 *Constant*	Enter the time interval on which the data in the table is to be recorded. The interval may be in µs, ms, s, or minutes, whichever is selected with the **Units** parameter. Enter 0 to make the data interval the same as the scan interval.
Units *Constant*	The units for the time parameters, PowerOff is the only instruction that uses hours or days.

Alpha Code	**Numeric Code**	**Units**
USEC	0	microseconds
MSEC	1	milliseconds
SEC	2	seconds
MIN	3	minutes

Lapses *Constant*	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 CR9000 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 CR9000 resets the processing the next time that the table is called after an output interval is skipped. If this next call to the table is on a scheduled interval, it will not output. Output will resume on the next interval. (If Sample is the only output processing instruction in the table, data will be output any time the table is called on the interval because sampling uses only the current value and involves no processing.)

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

OpenInterval Example:

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

```
Const RevDiff 1  'Reverse input to cancel offsets
Const Del 0      'Use default delay
Const Integ 0    'Use default 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)
                ModuleTemp(RefTemp,1,5,30)
                TCDiff(TC(),5,mV50C,5,9,TypeT,RefTemp,RevDiff,Del,Integ,1,0)
                CallTable AvgTemp
        NextScan
EndProg
```

DataEvent (PreTrigRecs, StartTrig, StopTrig, PostTrigRecs)

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

DataEvent Parameters

Parameter *& Data Type*	**Enter**
PreTrigRecs *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.. **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 CR9000 does not start checking for the stop trigger until after the Start Trigger occurs. A non-zero (true) constant may be used to store a fixed number of records when the start trigger occurs (total number of records = PreTrigRecs+ 1 record for the trigger +PostTrigRecs.). Zero (false) could be entered if it was desired to continuously store data once the start trigger occurred. **Value** — **Result** 0 — Do not trigger ≠ 0 — Trigger
PostTrigRecs *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 default integration
Public RefTemp           'Declare the variable used for reference temperature
Public TC(5)             'Declare the variable used for thermocouple measurements
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)
                ModuleTemp(RefTemp,1,5,30)
                TCDiff(TC(),5,mV50C,5,9,TypeT,RefTemp,RevDiff,Del,Integ,1,0)
                CallTable Event
        NextScan
EndProg
```

FillStop

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

Example:

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

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

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

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

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

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

WorstCase must be used with data tables sent to the CPU. It will not work if the event table is sent to the PAM module or CPU Flash memory.

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

WorstCase Example

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

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

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

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

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

6.3 Export Data Instructions

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

Example

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

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

FlashOut (Size)

Used to store data in Flash memory. Used inside DataTable to indicate the table is stored in Flash memory. Flash Memory is always fill and stop. FlashOut cannot be used in a DataTable that uses the WorstCase instruction.

Parameter *& Data Type*	**Enter** **FlashOut Parameters**
Size *Constant*	The size to make the data table. The number of data sets (records) to allocate memory for in Flash Memory. 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.

PamOut (Slot, Card, StopRing, Size)

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

Parameter *& Data Type*	**Enter** **PamOut Parameters**
Slot *Constant*	The number of the slot in the CR9000 card frame that holds the PAM Module.
Card *Constant*	The card (A or B) in which to store the data on the PAM module **Alpha Code** | **Numeric Code** | **Card** CARDA | 0 | A CARDB | 1 | B
StopRing *Constant*	A code to specify if the Data Table on the PCMCIA card is fill and stop or ring (newest data overwrites oldest). **Value** | **Result** 0 | Ring 1 | Fill and Stop
Size *Constant*	The size to make the data table. The number of data sets (records) to allocate memory for in the PCMCIA card. Each time a variable or interval trigger occurs, a line (or row) of data is output with the number of values determined by the output Instructions within the table. This data is called a record. **Note** Enter a negative number and all remaining memory (after creating fixed size data tables) will be allocated to the table or partitioned between all tables with a negative value for size. The partitioning algorithm attempts to have the tables fill at the same time.

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

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

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

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

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

Covariance Parameters

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

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

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

Parameter *& Data Type*	**Enter** **FFT Parameters**
Source *Variable*	The name of the Variable array that contains the input data for the FFT.
DataType *Constant*	A code to select the data storage format. **Alpha Code** / **Numeric Code** / **Data Format** IEEE4 / 24 / IEEE 4 byte floating point FP2 / 7 / Campbell Scientific 2 byte floating point
N *Constant*	Number of points in the original time series. The number of points must be a power of 2 (i.e., 512, 1024, 2048, etc.).
Tau *Constant*	The sampling interval of the time series.
Units *Constant*	The units for Tau. **Alpha Code** / **Numeric Code** / **Units** USEC / 0 / microseconds MSEC / 1 / milliseconds SEC / 2 / seconds MIN / 3 / minutes
Options *Constant*	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 - ϕ); 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 - ϕ), 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 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 real time series. The output is the time series whose FFT would result in the input array.

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

Normalization details:

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

Notes:

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

FFT Example

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

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

DataTable(Amp,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,1)
EndTable
DataTable(AmpPhase,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,2)
EndTable
DataTable(power,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,3)
EndTable
DataTable(PSD,1,1)
  fft(x,fp2,SIZE_FFT,10 msec,4)
EndTable
DataTable(FFT,1,1)
  fft(x,IEEE4,SIZE_FFT,10 msec,0)
EndTable
DataTable(IFFT,1,1)              'inverse FFT
  fft(y,IEEE4,SIZE_FFT,10 msec,5)
EndTable

BeginProg

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

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

EndProg
```

FieldNames “list of fieldnames”

The FieldNames instructions may be used to override the fieldnames that the CR9000 generates for results sent to the data table. **Fieldnames** must immediately follow the output instruction creating the data fields. Field names are limited to 19 characters. Individual names may be entered for each result generated by the previous output instruction or an array may used to name multiple fields. When the program is compiled, the CR9000 will determine how many fields are created. If the list of names is greater than the number of fields the extra names are ignored. If the number of fields is greater than the number names in the list of fieldnames, the default names are used for the remaining fields.

Example

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

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

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

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

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

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

Fieldnames can also be used to put the programmer’s description of the field into the “Process” field. The description for each field is entered following the fieldname as:

FieldNames(“fieldname1:Description1,fieldname2:Description2,…”)

The ‘ : ’ character indicates the start of the description. Descriptions can have any characters in them except commas. The description is optional.

The description is appended to the Processing field as ,:Description. That is, it is an addition to what is already automatically there, with a comma and colon in front of it.

The maximum size of the Processing Field is 64 characters. This leaves about 60 characters for a description of a Sample() instruction where the automatic description is Smp. A compile error is issued if the user’s description won’t fit.

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

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

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

Histogram4D (BinSelect, Source, DataType, DisableVar, Bins1, Bins2, Bins3, Bins4, Form, WtVal, LowLim1, UpLim1, LowLim2, UpLim2, LowLim3, UpLim3, LowLim4, 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

```
'\\\\\\\\\\\\\\\\\\\\ VARIABLES and CONSTANTS ////////////////////
Public mAmps
Public Volts
Dim Bin(2)
Units Bin = Percent

'\\\\\\\\\\\\\\\\\\\\\\\\ OUTPUT SECTION ////////////////////////
DataTable ("HIST4D",1,100)
 DataInterval(0,1,Sec,100)
 Histogram4D(Bin(), FP2, 0, 2, 4, 0, 0, 001, 100, 12, 14, -25, 0, 0, 0, 0, 0)
EndTable

DataTable ("VALUES",1,100)        'Trigger, buffer of 100 Records
 DataInterval(0,100,mSec,100)     'Synchronous, 100 lapses
 Average(1,Volts,FP2,0)    'Reps,Source,Type
 Average(1,mAmps,FP2,0)           'Reps,Source,Type
EndTable

'\\\\\\\\\\\\\\\\\\\\\\\\\\\\ PROGRAM
//////////////////////////
BeginProg
 Scan (1 mSec,0,0)
 Battery(Volts, 0)  'main battery volts
 Battery(mAmps, 1)          'main battery current
 Bin(1) = Volts
 Bin(2) = mAmps
 CallTable VALUES
 CallTable HIST4D
 Next Scan
EndProg
```

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

Parameter *& Data Type*	**Enter** **LevelCrossing Parameters**
Source *Variable or Array*	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 *Constant*	A code to select the data storage format. **Alpha Code** | **Numeric Code** | **Data Format** IEEE4 | 24 | IEEE 4 byte floating point FP2 | 7 | Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the histogram. The histogram that is eventually stored includes the inputs that occurred while the disable variable was 0. **Value** | **Result** 0 | Process current input ≠ 0 | Do not process current input
NumLevels *Constant*	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.
2ndDim *Constant*	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 *Arrayt*	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.
2ndArray *Array*	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 *Constant*	The minimum change in the source that must occur for a crossing to be counted.
Option *Constant*	The Option code is 3 digits - **ABC** **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. 101 means: Count on rising edge, reset count to 0 after each output, output counts.

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

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

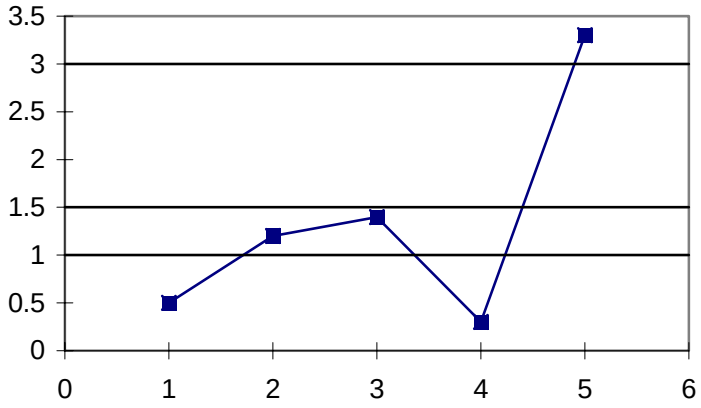


FIGURE 6.4-1. Example Crossing Data

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

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

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

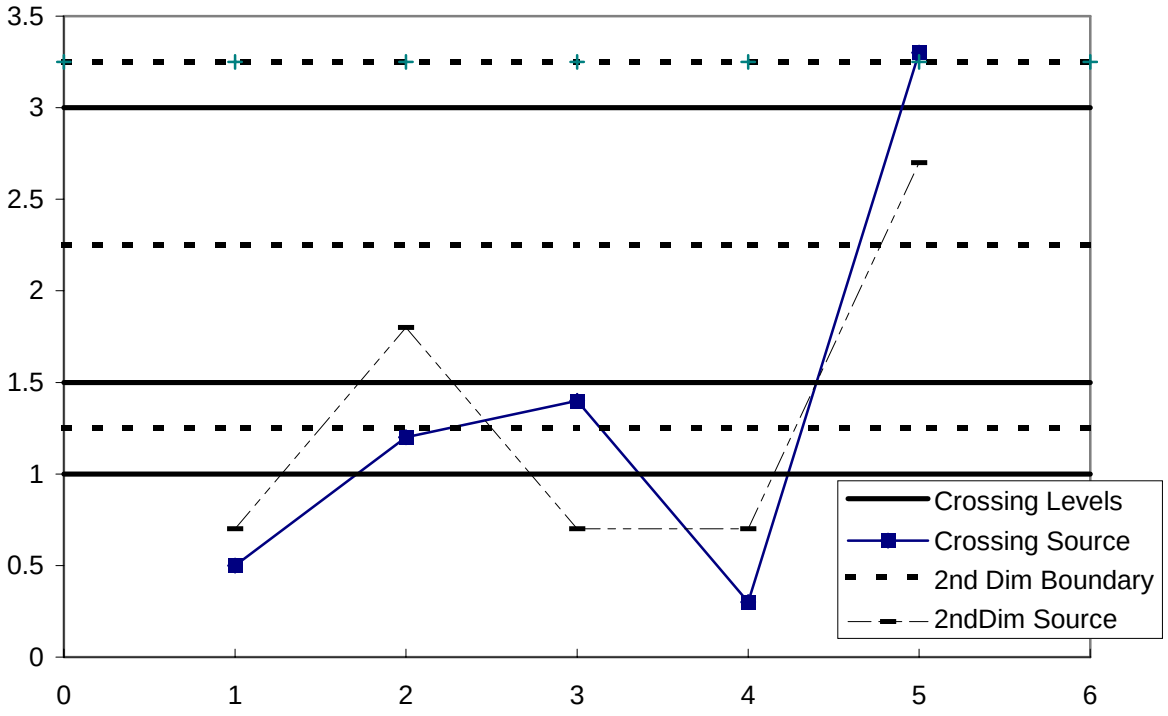


FIGURE 6.4-2. Crossing Data with Second Dimension Value

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

Histogram:

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

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

Histogram:

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

Note that the first bin of the second dimension is always “open”. Any value less than the specified boundary is included in this bin. The last bin of the

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

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

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

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

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

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

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

<table>
<tr><td>Parameter
& Data Type</td><td colspan="3">Enter Maximum Parameters</td></tr>
<tr><td>Reps
Constant</td><td colspan="3">The number of maximum values to determine. When repetitions are greater than 1, the source must be an array..</td></tr>
<tr><td>Source
Variable</td><td colspan="3">The name of the Variable that is the input for the instruction.</td></tr>
<tr><td>DataType
Constant</td><td colspan="3">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</td></tr>
</table>

DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not checked for a new maximum. The maximum that is eventually stored is the maximum that occurred while the disable variable was 0. **Value** / **Result** 0 — Process current input ≠ 0 — Do not process current input
Time *Constant*	Option to store time of Maximum. When time is output, the maximums for all reps are output first followed by the respective times at which they occurred. **Value** / **Result** 0 — Do not store time 1 — Store time

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

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

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

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

Parameter *& Data Type*	**Enter** **RainFlow Parameters**
Source *Variable*	The variable that is tested to determine which bin is selected
DataType *Constant*	A code to select the data storage format. **Alpha Code** / **Numeric Code** / **Data Format** IEEE4 / 24 / IEEE 4 byte floating point FP2 / 7 / Campbell Scientific 2 byte floating point
DisableVar *Constant, Variable, or Expression*	A non-zero value will disable intermediate processing. Normally 0 is entered so all inputs are processed. For example, when the disable variable is ≠0 the current input is not included in the histogram. The histogram that is eventually stored includes the inputs that occurred while the disable variable was 0. **Value** / **Result** 0 / Process current input ≠ 0 / Do not process current input
MeanBins *Constant*	The number of bins or subranges to sort the mean value of the signal during a stress strain cycle into. Enter 1 to disregard the signal value and only sort by the amplitude of the signal. The width of each subrange is equal to the HiLimit - LowLimit divided by the number of bins. The lowest bin's minimum value is the low limit and the highest bin's maximum value is the High limit
AmpBins *Constant*	The number of bins or subranges to sort the amplitude of a stress strain cycle into. The width of each subrange is equal to the HiLimit - LowLimit divided by the number of bins.
LowLim *Constant*	The lower limit of the input signal and the Mean Bins.
UpLim *Constant*	The upper limit of the input signal and the Mean Bins.
MinAmp *Constant*	The minimum amplitude that a stress strain cycle must have to be counted..
Form *Constant*	The Form code is 3 digits - **ABC** **Code** / **Form** A = 0 / Reset histogram after each output. A = 1 / Do not reset histogram. B = 0 / Divide bins by total count. B = 1 / Output total in each bin. C = 0 / Open form. Include outside range values in end bins. C = 1 / Closed form. Exclude values outside range. 101 means: Do not reset. Divide bins by total count. Closed form.

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

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

that were associated with that bin (i.e., number of cycles in bin divided by total number of cycles counted).

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

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

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

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

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

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

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

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

Sample (Reps, Source, DataType)

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

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

StdDev (Reps, Source, DataType, DisableVar)

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

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

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

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

Totalize (Reps, Source, DataType, DisableVar)

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

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

Section 7. Measurement Instructions

7.1 Voltage Measurements

7.2 Thermocouple Measurements

Measure the output of thermocouples and convert to temperature.

Resistance Bridge Measurements

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

7.3 Half Bridges

7.4 Full Bridges

7.5 Excitation / Continuous Analog Output

7.6 Self Measurements

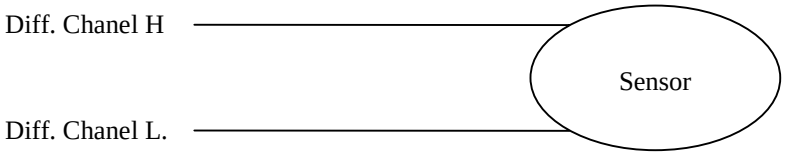
7.7 Peripheral Devices

7.8 Digital I/O

7.9 9052DC Filter Module Measurements

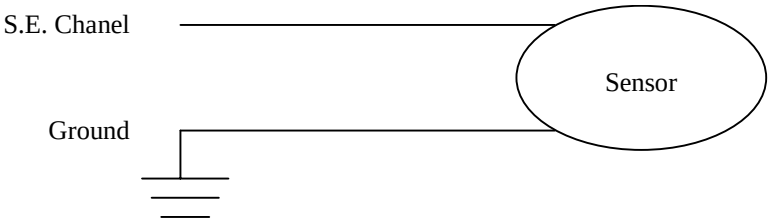
7.1 Voltage Measurements

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



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

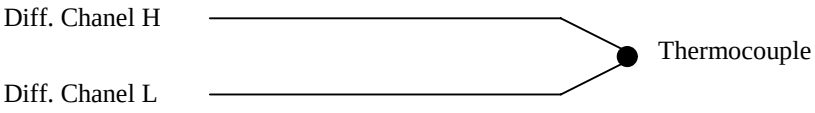
VoltSE (Dest, Reps, Range, ASlot, SEChan, SettlingTime, Integ, Mult, Offset)



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

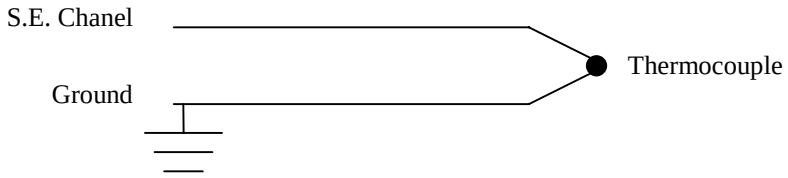
7.2 Thermocouple Measurements

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



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

TCSE (Dest, Reps, Range, ASlot, SEChan, TCType, TRef, 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.

VOLTDIFF AND TCDIFF INSTRUCTION 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. For analog measurements, entering reps as a negative number forces all reps to be on the same channel except for with CR9058E module.
Range *Constant*	The voltage range for the measurement. V ranges output volts, mV ranges output millivolts.

See Section 3.2.2 for more info on the C & R range code options.	**± 5 Volt Analog Input Module**				**± 50 Volt Analog Input Module**				**CR9058E* Isolation Module**			
	Alpha Code	Num Code	R * Option Code	Voltage Range (±mV)	Alpha Code	Num Code	R * Option Code	Voltage Range ±	Alpha Code	Num Code	R * Option Code	Voltage Range
	mV5000	0	100	5000	V50	6	N/A	50 V	V60	24	N/A	± 60 V
	mV1000	1	101	1000	V10	7	N/A	10 V	V20	25	N/A	± 20 V
	mV200	4	104	200	V2	10	N/A	2 V	V2	10	N/A	± 2 V
	mV50	5	105	50	mV500	11	N/A	500 mV	V2C	22	N/A	± 2 V
	mV200C	16	116	200	Alpha Codes ending with a C signify that the channel will be pulled into common mode range & checked for open input. See Section 3.2 for details.							
	mV50C	17	117	50								

ASlot *Constant*	The number of the slot that holds the Analog Input Module to be used for the measurement.
DiffChan *Constant*	The differential channel number on which to make the first measurement. When **Reps** are used, subsequent measurements will be automatically made on the following differential channels.
TCType *Constant*	The code for the thermocouple type.

Alpha Code	**Numeric Code**	**Thermocouple Type**
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 name of the variable that is the reference temperature for the thermocouple measurements.

Parameter	Enter **VOLTDIFF AND TCDIFF INSTRUCTION PARAMETERS**
RevDiff *Constant*	Option to reverse inputs to cancel offsets. The sign corrected average of these measurements is used in the result. This technique cancels voltage offsets in the measurement circuitry but requires twice as much time to complete the measurement. (CR9058E: All channels on a module must have same setting.)
	Value 0 — Signal is measured with the high side referenced to the low 1 — A second measurement is made after reversing the inputs
SettlingTime *Constant*	The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement (10 microsecond resolution). Enter 0 when using the CR9058E (Settling Time not used).

Entry	Voltage Range	Delay	CR9055 Voltage Range	Delay
0	± 50 mV	20 µS (default)	± 500 mV	40 µS (default)
0	± 200 mV	20 µS (default)	± 2 V	40 µS (default)
0	±1000 mV	10 µS (default)	± 10 V	30 µS (default)
0	± 5000 mV	10 µS (default)	± 50 V	30 µS (default)
> 0	all	Truncate to closest 10 µS	all	Truncate to closest 10 µS

Integ *Constant*	The integration time in microseconds for each of the channels measured (10 microseconds resolution). CR9058E*:100 microsecond resolution. All channels on a CR9058 module must have same integration.
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

R*: Place an R at the end of the range code (ex: 50mVCR) in order to perform a common mode range check before making the measurement. If the input is out of common mode range, a NAN will be returned. See section 3.2.2 for more details on the common mode check option.

CR9058E*: Enter -1 for the integration parameter when using a CR9058E and the integration will automatically be set to the maximum allowed for the given Scan Interval. See section 3.2 for CR9058E measurement details.

Parameter *& Data Type*	Enter **VOLTSE & TCSE INSTRUCTION PARAMETERS**
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. For analog measurements, entering reps as a negative number forces all reps to be on the same channel.
Range *Constant*	The voltage range for the measurement. V ranges output volts, mV ranges output millivolts.

± 5 Volt Analog Input Module			± 50 Volt Analog Input Module		
Alpha Code	Numeric Code	Voltage Range	Alpha Code	Numeric Code	Voltage Range
mV5000	0	± 5000 mV	V50	6	± 50 V
mV1000	1	± 1000 mV	V10	7	± 10 V
mV200	4	± 200 mV	V2	10	± 2 V
mV50	5	± 50 mV	mV500	11	± 500 mV

ASlot *Constant*	The number of the slot that holds the Analog Input Module to be used for the measurement.
SEChan *Constant*	The single-ended channel number on which to make the first measurement. When Reps are used, subsequent measurements will be automatically made on the following single-ended channels.

VOLTSE & TCSE INSTRUCTION PARAMETERS

Parameter *& Data Type* — **Enter**

TCType *Constant*

The code for the thermocouple type.

Alpha Code	Numeric Code	Thermocouple Type
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 name of the variable that is the reference temperature for the thermocouple measurements.

RevDiff *Constant*

Option to reverse inputs to cancel offsets. The sign corrected average of these measurements is used in the result. This technique cancels voltage offsets in the measurement circuitry but requires twice as much time to complete the measurement.

Value	Result
0	Signal is measured with the high side referenced to the low
1	A second measurement is made after reversing the inputs

SettlingTime *Constant*

The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement (10 microsecond resolution)

Entry	Voltage Range	Delay	CR9055 Voltage Range	Delay
0	± 50 mV	20 µS (default)	± 500 mV	40 µS (default)
0	± 200 mV	20 µS (default)	± 2 V	40 µS (default)
0	±1000 mV	10 µS (default)	± 10 V	30 µS (default)
0	± 5000 mV	10 µS (default)	± 50 V	30 µS (default)
> 0	All	Truncate to closest 10 µS	all	Truncate to closest 10 µS

Integ *Constant*

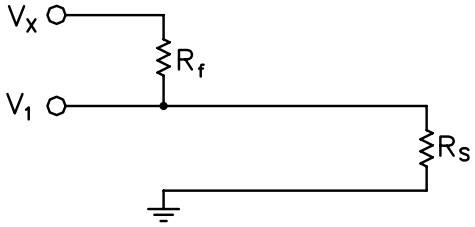
The integration time in microseconds for each of the channels measured (10 microseconds resolution).

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

A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

7.3 Half Bridges

BrHalf (Dest, Reps, Range, ASlot, SEChan, ExSlot, ExChan, MesPEx, ExmV, RevEx, SettlingTime, Integ, Mult, Offset)

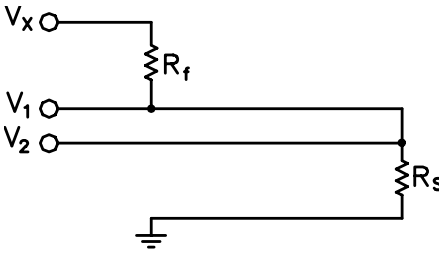


X = result w/mult = 1, offset = 0

$$X = \frac{V_1}{V_X} = \frac{R_S}{R_S + R_f}$$

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

BrHalf3W (Dest, Reps, Range, ASlot, SEChan, ExSlot, ExChan, MesPEx, ExmV, RevEx, SettlingTime, Integ, Mult, Offset)



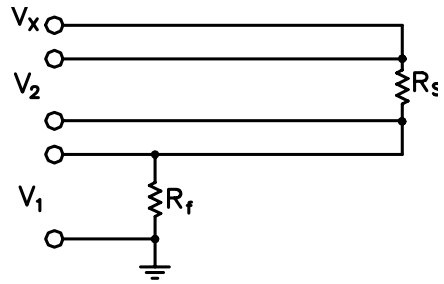
X = result w/mult = 1, offset = 0

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

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

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

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



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

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

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

Parameter *& Data Type* — **Enter** **BRHALF, BRHALF3W, BRHALF4W PARAMETERS**

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. For analog measurements, entering reps as a negative number forces all reps to be on the same channel.

Range *Constant*

The voltage range for the measurement.

± 5 Volt Analog Input Module			**± 50 Volt Analog Input Module**		
Alpha Code	**Numeric Code**	**Voltage Range**	**Alpha Code**	**Numeric Code**	**Voltage Range**
mV5000	0	± 5000 mV	V50	6	± 50 V
mV1000	1	± 1000 mV	V10	7	± 10 V
mV200	4	± 200 mV	V2	10	± 2 V
mV50	5	± 50 mV	mV500	11	± 500 mV

ASlot *Constant*

The number of the slot that holds the Analog Input Module to be used for the measurement.

SEChan *Constant*

The single-ended channel number on which to make the first measurement. When Reps are used, subsequent measurements will be automatically made on the following single-ended channels.

ExSlot *Constant*

The slot that holds the Excitation Module for the measurement.

ExChan *Constant*

Enter the excitation channel number to excite the first measurement.

Channels	**Result**
1 - 6	Continuous analog output channels, will remain at the excitation voltage set by the instruction unless a subsequent instruction changes their voltage
7 - 16	Switched excitation channels, are switched to the excitation voltage for the measurement and switched off between measurements.

MesPEx *Constant*

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

ExmV *Constant*

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

RevEx *Constant*

Option to reverse excitation to cancel offsets.

Value	**Result**
0	Excite only with the excitation voltage entered
1	A second measurement is made with the voltage polarity reversed.

RevDiff *Constant*

Option to reverse inputs to cancel offsets.

Value	**Result**
0	Signal is measured with the high side referenced to the low
1	A second measurement is made after reversing the inputs

SettlingTime *Constant*

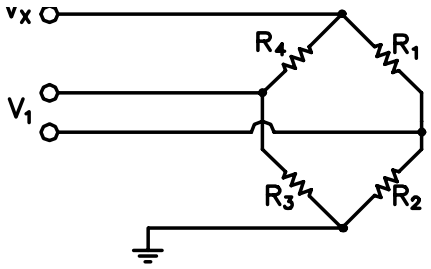
The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement (10 microsecond resolution).

Entry	**Voltage Range**	**Delay**	**CR9055 Voltage Range**	**Delay**
0	± 50 mV	20 µS (default)	± 500 mV	40 µS (default)
0	± 200 mV	20 µS (default)	± 2 mV	40 µS (default)
0	±1000 mV	10 µS (default)	± 10 mV	30 µS (default)
0	± 5000 mV	10 µS (default)	± 50 mV	30 µS (default)
> 0	All	Truncate to closest 10 µS	all	Truncate to closest 10 µS

Parameter *& Data Type*	**Enter** **BRHALF, BRHALF3W, BRHALF4W PARAMETERS**
Integ *Constant*	The integration time in microseconds for each of the channels measured (10 microseconds resolution).
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement.

7.4 Full Bridges

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

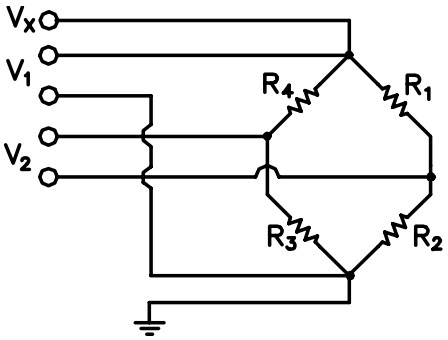


X = result w/mult = 1, offset = 0

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

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, ASlot, DiffChan, ExSlot, ExChan, MesPEx, 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.

7.5 Excitation/Continuous Analog Output

Excite (ExSlot, ExChan, ExmV, SettlingTime)

This instruction sets the selected excitation output to a specific value. Channels 1 through 6 are continuous analog output channels and will remain at the excitation voltage set by the instruction unless a subsequent instruction changes their voltage. Channels 7 through 16 are switched excitation channels, they are switched to the excitation voltage for the time specified for the Delay and then switched off.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter BRIDGEFULL, BRIDGEFULL6W, EXCITE PARAMETERS</td></tr>
<tr><td>Dest
Variable or Array</td><td>The Variable in which to store the results of the instruction. When Reps are used the results are stored in an array with the variable name. An array must be dimensioned to have elements for all the Reps.</td></tr>
<tr><td>Reps
Constant</td><td>The number of repetitions for the measurement or instruction.</td></tr>
<tr><td>Range
Constant</td><td>The voltage range for the measurement.
<table>
<tr><th colspan="3">± 5 Volt Analog Input Module</th><th colspan="3">± 50 Volt Analog Input Module</th></tr>
<tr><th>Alpha Code</th><th>Numeric Code</th><th>Voltage Range</th><th>Alpha Code</th><th>Numeric Code</th><th>Voltage Range</th></tr>
<tr><td>mV5000</td><td>0</td><td>± 5000 mV</td><td>V50</td><td>6</td><td>± 50 V</td></tr>
<tr><td>mV1000</td><td>1</td><td>± 1000 mV</td><td>V10</td><td>7</td><td>± 10 V</td></tr>
<tr><td>mV200</td><td>4</td><td>± 200 mV</td><td>V2</td><td>10</td><td>± 2 V</td></tr>
<tr><td>mV50</td><td>5</td><td>± 50 mV</td><td>mV500</td><td>11</td><td>± 500 mV</td></tr>
</table></td></tr>
<tr><td>ASlot
Constant</td><td>The number of the slot that holds the Analog Input Module to be used for the measurement.</td></tr>
<tr><td>DiffChan
Constant</td><td>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.</td></tr>
<tr><td>ExSlot
Constant</td><td>The slot that holds the Excitation Module for the measurement.</td></tr>
<tr><td>ExChan
Constant</td><td>Enter the excitation channel number to excite the first measurement.
<table>
<tr><th>Channels</th><th>Result</th></tr>
<tr><td>1 - 6</td><td>Continuous analog output channels, will remain at the excitation voltage set by the instruction unless a subsequent instruction changes their voltage</td></tr>
<tr><td>7 - 16</td><td>Switched excitation channels, are switched to the excitation voltage for the measurement and switched off between measurements.</td></tr>
</table></td></tr>
<tr><td>MesPEx
Constant</td><td>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.</td></tr>
<tr><td>ExmV
Constant</td><td>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.</td></tr>
<tr><td>RevEx
Constant</td><td>Option to reverse excitation to cancel offsets.
<table>
<tr><th>Value</th><th>Result</th></tr>
<tr><td>0</td><td>Excite only with the excitation voltage entered</td></tr>
<tr><td>1</td><td>A second measurement is made with the voltage polarity reversed.</td></tr>
</table></td></tr>
<tr><td>RevDiff
Constant</td><td>Option to reverse inputs to cancel offsets
<table>
<tr><th>Value</th><th>Result</th></tr>
<tr><td>0</td><td>Signal is measured with the high side referenced to the low</td></tr>
<tr><td>1</td><td>A second measurement is made after reversing the inputs</td></tr>
</table></td></tr>
</table>

Parameter *& Data Type*	**Enter** **BRIDGEFULL, BRIDGEFULL6W, EXCITE PARAMETERS**
SettlingTime *Constant*	The time in microseconds to delay between setting up a measurement (switching to the channel, setting the excitation) and making the measurement (10 microsecond resolution).

Entry	**Voltage Range**	**Delay**	**CR9055 Voltage Range**	**Delay**
0	± 50 mV	20 µS (default)	± 500 mV	40 µS (default)
0	± 200 mV	20 µS (default)	± 2 mV	40 µS (default)
0	±1000 mV	10 µS (default)	± 10 mV	30 µS (default)
0	± 5000 mV	10 µS (default)	± 50 mV	30 µS (default)
> 0	All	Truncate to closest 10 µS	all	Truncate to closest 10 µS

Integ *Constant*	The integration time in microseconds for each of the channels measured (10 microseconds resolution).
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement.

7.6 Self Measurements

Battery (Dest, BattOpt)

This instruction reads the voltage or current of the battery powering the system or the voltage of the backup lithium battery. The units for battery voltage are volts; current is in milliamperes.

ModuleTemp (Dest, Reps, ASlot, Integ)

This instruction measures the temperature in °C of the specified CR9050(E), CR9051E, or CR9058E analog input module.

Parameter *& Data Type*	**Enter** **BATTERY, MODULETEMP PARAMETERS**
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.
BattOpt *Constant*	The code indicating the desired measurement.

Code	**Measurement**
0	Main battery voltage, volts
1	Main battery current, milliamperes
2	Memory backup battery (lithium), volts

Reps *Constant*	The number of repetitions for the measurement or instruction. If reps is greater than 1, the first element of the Dest array will hold the temperature for the module in the specified Aslot and the modules' temperatures in the sequentially following slots will be loaded into the corresponding elements of the Dest array.
ASlot *Constant*	The number of the slot that holds the first Analog Input Module to be used for the measurement.
Integ *Constant*	The integration time in microseconds for each of the channels measured (10 microseconds resolution). The CR9000 will repeat measurements every 10 microseconds throughout the integration interval (with the appropriate Delay at the beginning and between RevDiff and RevEx if used) and output the average. The random noise level is decreased by the square root of the number of measurements made. An integration time of one 60 Hz cycle (16,670 microseconds) will cancel 60 Hz noise. Enter 0 for no integration and the fastest measurements.

Calibrate

Measures offset and gain on all voltage ranges. This instruction is done automatically at user program compile time. The major factor affecting the calibration of the analog section is temperature. If calibration is not done as part of the program, a typical shift in the calibration is 0.01 % per degree C change from the temperature at which the program compile calibration occurred. When there is adequate time for all measurements, **BiasComp** and **Calibrate** are typically run in a scan in the SlowSequence section of the program to provide continuous adjusting of the calibration as temperature changes. If executed in the SlowSequence, an RC filter is applied with the previous calibration weighted .95 and the new weighted .05. Calibrate uses 54 measurement slots in the Task Sequencer.

BiasComp

Measures bias current and adjusts the bias current DACS accordingly. This instruction is done automatically at user program compile time. The bias current is the amount of current that is required to flow into the input channel in order to make the measurement. This is reduced to a minimum (<3 nanoamps) when the bias current compensation is adjusted correctly. If the bias current compensation is not adjusted correctly, the current could rise as high as 100 nanoamps. The major factor affecting the bias current is temperature. When there is adequate time for all measurements, **BiasComp** and **Calibrate** are typically run in a scan in the SlowSequence section of the program to provide continuous adjusting of the bias current compensation and the calibration as temperature changes. If executed in the SlowSequence, an RC filter is applied with the previous bias compensation weighted .95 and the new weighted .05. BiasComp uses 120 measurement slots in the task sequencer.

The 4 DAC values that are the results of the bias compensation appear in the Status Table: BiasLo, BiasLo X10, BiasHi, BiasHI X10.

7.7 Peripheral Devices

AM25T (AM25TChan, CardAnlg, ChanAnlg, CardPort, ClkPort, ResPort, ChanExcite)

This Instruction controls the AM25T Multiplexer when used with the CR9000. The AM25T instruction precedes the instruction used to make the measurements on the AM25T, and must be inserted before any instruction that makes a measurement on the AM25T. In the measurement that follows, the CR9000 will automatically switch the AM25T so that all repetitions specified in the measurement are made on AM25T channels. If RevDiff is specified in the measurement, the CR9000 will reverse the input at the AM25T. ModuleTemp can be used following the AM25T instruction (set for channel 0) to measure the AM25T temperature.

Parameter *& Data Type*	**Enter** **AM25T INSTRUCTION PARAMETERS**
Am25tChan *Constant*	The starting input channel on the multiplexer for the following measurement. If the following instruction needs to start measuring a AM25T channel other than number 1, the channel is specified here. Specify channel 0 when used before ModuleTemp to measure the module temperature of the AM25T.
CardAnlg *Constant*	The number of the slot that contains the CR9050 Module used to measure the AM25T reference temperature and connected sensors.
ChanAnlg *Constant*	The channel number on the CR9050 Module that will be used to make the actual measurements from the AM25T.
CardPort *Constant*	The number of the slot that contains the CR9060 Module used to Clock and Reset the multiplexer and to provide excitation for the reference temperature PRT.
ClkPort *Constant*	The Digital Output port number on the CR9060 Module that will be used to clock the AM25T. One clock port may be used with several AM25Ts.
ResPort *Constant*	The Digital Output port number on the CR9060 Module that will be used to enable and reset the AM25T. Each AM25T must have it's own unique Reset line
ChanExcite *Constant*	The Excitation Channel number on the CR9060 Module that will be used to provide excitation for the PRT reference temperature measurement.

```
'This example demonstrates using the AM25T thermocouple multiplexer
'with the CR9000.

'\\\\\\\\\\ VARIABLES and CONSTANTS /////////
Const AM25TChan = 1    'starting channel in AM25T
Const CardAnlg = 5     '9050 module slot AM25T connected to
Const ChanAnlg = 14    '9050 module diff channel AM25T connected to
Const CardPort = 6     '9060 module slot
Const ClkPort = 1      '9060 Digital Control Output port
Const ResPort = 2      '9060 Digital Control Output port
Const ChanExcite = 16  '9060 Excitation Channel
Const MuxReps = 25     'number of AM25T channels to use
Const RevIn = 0        'don't reverse TCDiff measurement
Const Delay = 5000     'delay in uSecs between tc measurements on AM25T
Const Integ = 500      'integration time in uSecs of each AM25T measurements

Public RefAM25T, Mux(25)

'\\\\\\\\\\\\\\ OUTPUT SECTION //////////////
DataTable(MUXTC, 1, 2000)
        Sample(MuxReps,Mux(),FP2)
EndTable

'\\\\\\\\\\\\\\\\\\ PROGRAM /////////////////
BeginProg
        Scan (200, mSec, 0, 0)
AM25T(0,CardAnlg, ChanAnlg, CardPort, ClkPort, ResPort, ChanExcite)
ModuleTemp(RefAM25T, 1, CardAnlg, Integ) 'measure ref temp in AM25T

AM25T(AM25TChan, CardAnlg, ChanAnlg, CardPort, ClkPort, ResPort, ChanExcite)
TCDiff(Mux(), MuxReps, mv50,CardAnlg, ChanAnlg, TypeT, RefAM25T, RevIn, Delay, Integ, 1, 0)
        CallTable MUXTC
        NextScan
EndProg
```

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

The CANBUS instruction is used to measure and control the SDM-CAN interface. Multiple CANBUS instructions may be used within a program. The initial function of the instruction is to configure the SDM-CAN interface when the datalogger program is compiled. Subsequent instructions can be used to determine what data is passed between the CAN-bus network and the datalogger, set and/or read the SDM-CAN's internal switches, and read and/or reset detected errors.

The SDMTrigger instruction can be used to trigger simultaneous measurements from one or more SDM-CANs and other SDM devices connected to the datalogger. When the SDMTrigger instruction is encountered in a program, it broadcasts a special SDM message which causes all the SDM-CAN devices to copy the last data values captured from the CAN-bus into the working data buffers. Refer to the SDM-CAN manual for additional help.

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.

Parameter *& Data Type*	**Enter** **CANBUS INSTRUCTION PARAMETERS**
Dest *Variable or Array*	The 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 *Constant*	The address of the SDM-CAN with which to communicate. Valid SDM addresses are 0 through 15.
TimeQuanta *Constant*	Three time segments are used to set the bit rate and other timing parameters for the CAN-bus network, TimeQuanta, TSEG1, and TSEG2. These parameters are entered as integer numbers. The relationship between the three time segments is defined as: $t_{bit} = t_q + t_{TSEG1} + t_{TSEG2}$ The first time segment, the synchronization segment (S-SG), is defined by the TimeQuanta parameter. To calculate a suitable value for TimeQuanta, use the following equation: $TimeQuanta = t_q \times 8 \times 10^6$ where tq = the TimeQuanta. There are between 8 and 25 time quanta in the bit time. The bit time is defined as 1/baud rate.
TSEG1 *Constant*	The second time segment, TSEG1, is actually two time segments known as the propagation segment and phase segment one. The value entered is determined by the characteristics of the network and the other devices on the network. It can be calculated as: $T_{SEG1} = t_{TSEG1} / t_q$
TSEG2 *Constant*	The third time segment, TSEG2 (the phase segment two), is defined by the TSEG2 parameter. The value of TSEG2 can be calculated using the equation: $T_{SEG1} = t_{SEG2} / t_q$ The relative values of TSEG1 and TSEG2 determine when the SDM-CAN samples the data bit.

Parameter *& Data Type*	**Enter** **CANBUS INSTRUCTION PARAMETERS**	
ID *Constant*	Each device on a CAN-bus network prefaces its data frames with an 11 or 29 bit identifier. The ID parameter is used to set this address. The ID is entered as a single decimal equivalent. Enter a positive value to signify a 29 bit ID or a negative value to signify an 11 bit ID.	
DataType *Constant*	The DataType parameter defines what function the CANBUS instruction will perform. This instruction can be used to collect data, buffer data for transmission to the CAN-bus, transmit data to the CAN-bus, read or reset error counters, read the status of the SDM-CAN, read the SDM-CAN's OS signature and version, send a remote frame, or read or set the SDM-CAN's internal switches. Enter the numeric value for the desired option.	
	Value	Description
	1	Retrieve data; unsigned integer, most significant byte first.
	2	Retrieve data; unsigned integer, least significant byte first.
	3	Retrieve data; signed integer, most significant byte first.
	4	Retrieve data; signed integer, least significant byte first.
	5	Retrieve data; 4-byte IEEE floating point number; most significant byte first.
	6	Retrieve data; 4-byte IEEE floating point number; least significant byte first.
	Options 7 through build a data frame in SDM-CAN memory	
	7	Unsigned integer, most significant byte first. Overwrite existing data.
	8	Unsigned integer, least significant byte first. Overwrite existing data.
	9	Signed integer, most significant byte first. Overwrite existing data.
	10	Signed integer, least significant byte first. Overwrite existing data.
	11	4-byte IEEE floating point number; most significant byte first. Overwrite existing data.
	12	4-byte IEEE floating point number; least significant byte first. Overwrite existing data.
	13	Unsigned integer, most significant byte first. Logical "OR" with existing data.
	14	Unsigned integer, least significant byte first. Logical "OR" with existing data.
	15	Signed integer, most significant byte first. Logical "OR" with existing data.
	16	Signed integer, least significant byte first. Logical "OR" with existing data.
	17	4-byte IEEE floating point number; most significant byte first. Logical "OR" with existing data.
	18	4-byte IEEE floating point number; least significant byte first. Logical "OR" with existing data.
	Options 19 through 25 Transmit data to the CAN-bus	
	19	Unsigned integer, most significant byte first.
	20	Unsigned integer, least significant byte first.
	21	Signed integer, most significant byte first.
	22	Signed integer, least significant byte first.
	23	4-byte IEEE floating point number; most significant byte first.
	24	4-byte IEEE floating point number; least significant byte first.
	25	Previously built data frame
	26	Set up previously built data frame as a Remote Frame Response.
	27	Read Transmit, Receive, Overrun, and Watchdog errors. The errors are placed consecutively in the array specified by the Dest parameter.
	28	Read Transmit, Receive, Overrun, and Watchdog errors. The errors are placed consecutively in the array specified by the Dest parameter. Reset error counters to 0 after reading.
	29	Read SDM-CAN status; result is placed into the array specified in the Destination parameter. The result codes are as follows:

Parameter *& Data Type* | **Enter** | **CANBUS INSTRUCTION PARAMETERS**

Status	Description
0000	SDM-CAN involved in bus activities; error counters < than 96.
0001	SDM-CAN involved in bus activities; one or more error counters is greater than or equal to 96.
0002	SDM-CAN is not involved in bus activities; error counters < 96.
0003	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**ination 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.

Digit	Value	Description
A	0	Not Used
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	0	Disable I/O interrupts (default).
	1	Enable I/O interrupts, pulsed mode.
	2	Enable I/O interrupts, fast mode.
	3-7	Currently not used.
	8	Place the SDM-CAN into low power stand-by mode.
	9	Leave switch setting unchanged.
D	0	Listen only (error passive) mode. CAN transmissions are not confirmed.
	1	Transmit once. Data will not be retransmitted in case of error or loss of arbitration. Frames received without error are acknowledged.
	2	Self-reception. A frame transmitted from the SDM-CAN that was acknowledged by an external node will also be received by the SDM-CAN but no retransmission will occur in the event of loss of arbitration or error. Frames received correctly from an external node are acknowledged.
	3	Normal, retransmission will occur in the event of loss of arbitration or error. Frames received correctly from an external node are acknowledged. This is the typical setting to use if the SDM-CAN is to be used to transmit data.
	4	Transmit once; self-test. The SDM-CAN will perform a successful transmission even if there is no acknowledgment from an external CAN node. Frames received correctly from an external node are acknowledged.
	5	Self-reception; self -test. The SDM-CAN will perform a successful transmission even if there is no acknowledgment from an external CAN node. Frames received correctly from an external node are acknowledged. SDM-CAN will receive its own transmission.

Parameter *& Data Type*	**Enter** **CANBUS INSTRUCTION PARAMETERS**
	6 — Normal; self-test. The SDM-CAN will perform a successful transmission even if there is no acknowledgment from an external CAN node. Frames received correctly from an external node are acknowledged. 7,8 — Not Used. 9 — Leave switch setting unchanged. 33 — Read SDM-CAN's internal switches. Place results in the array specified in the **Dest**ination parameter.
StartBit *Constant*	StartBit is used to identify the least significant bit of the data value within the CAN data frame to which the instruction relates. The bit number can range from 1 to 64 (there are 64 bits in a CAN data frame). The SDM-CAN adheres to the ISO standard where the least significant bit is referenced to the right most bit of the data frame. If a negative value is entered, the least significant bit is referenced to the left most bit of the data frame.
NumBits *Constant*	NumBits specifies the number of bits that will be used in a transaction. The number can range from 1 to 64 (there are 64 bits in a CAN data frame). The SDM-CAN can be configured to notify the datalogger when new data is available by setting a control port high. This allows data to be stored in the datalogger tables faster than the program execution interval. This interrupt function is enabled by entering a negative value for this parameter. **Note: This parameter may be overridden by a fixed number of bits, depending upon the data type selected.**
NumVals	The number of values (beginning with the value stored in the Dest array) that will be transferred to or from the datalogger during one operation. For each value transferred, the Number of Bits (NumBits) will be added to the Start Bit number so that multiple values can be read from or stored to one data frame.
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

CSAT3 (Dest, Reps, Address, Command)

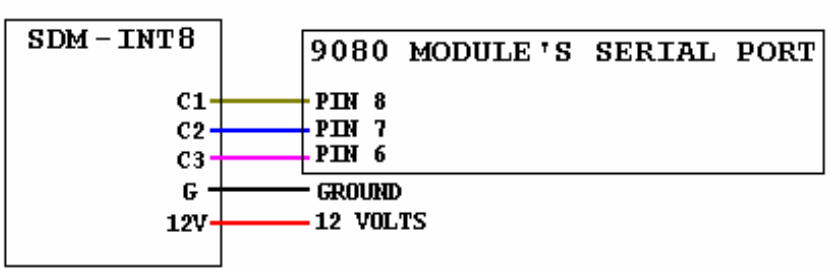
Communicates with the CSAT3 three dimensional sonic anemometer. See CSAT3 manual for more information.

INT8 INTERVAL TIMER

Used to control the INT8, an 8 Channel Interval Timer module, using the CR9000.

Syntax

INT8(Dest, Address,Config8_5,Config4_1,
Funct8_5,Funct4_1,OutputOpt,CaptureTrig, Mult, Offset **)**



Remarks
This Instruction allows the use of the SDM-INT8, 8 Channel Interval Timer, with the CR9000. The INT8 is a (S)ynchronous (D)evice for the (M)easurement of intervals, counts between events, frequencies, periods, and/or time since an event. See the INT8 manual for more information about its capabilities.

NOTE: This instruction must NOT be placed inside a conditional statement.

Parameter *& Data Type*	**Enter INT8 INSTRUCTION PARAMETERS**
Dest *Variable or Array*	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.
Address *Constant*	The INT8 is addressable using internal jumpers. The jumpers are set at the factory for address 00. See Appendix A of the INT8 manual for details on changing the INT8 address.
Config8_5 **Config4_1** *Constants*	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.

Digit	**Edge**
0	High level, rising edge
1	High level, falling edge
2	Low level, rising edge
3	Low level falling edge

<table>
<tr><td>Parameter
& Data Type</td><td>Enter INT8 INSTRUCTION PARAMETERS</td></tr>
<tr><td>Funct8_5
Funct4_1
Constants</td><td>Each of the 8 input channels can be independently programmed for one of eight different timing functions. Funct8_5 is a four digit code to program the timing functions of INT8 channels 5 through 8. Funct4_1 is a four digit code to program the timing functions of INT8 channels 1 through 4. See section 5.3 of the INT8 manual for further details about these functions.
<table>
<tr><th>Digit</th><th>Results</th></tr>
<tr><td>0</td><td>None</td></tr>
<tr><td>1</td><td>Period (msec) between edges on this channel</td></tr>
<tr><td>2</td><td>Frequency (kHz) of edges on the channel</td></tr>
<tr><td>3</td><td>Time between an edge on the previous channel and the edge on this channel (msec)</td></tr>
<tr><td>4</td><td>time between an edge on channel 1 and the edge on this channel (msec)</td></tr>
<tr><td>5</td><td>Number of edges on channel 2 between the last edge on channel 1 and the edge on this channel using linear interpolation</td></tr>
<tr><td>6</td><td>Low resolution frequency (kHz) of edges on this channel</td></tr>
<tr><td>7</td><td>Total number of edges on this channel since last interrogation</td></tr>
<tr><td>8</td><td>Integer number of edges on channel 2 between the last edge on channel 1 and the edge on this channel.</td></tr>
</table>
For example, 4301 in the second function parameter means to return 3 values: the period for channel 1, (nothing for channel 2) the time between an edge on channel 2 and an edge on channel 3, and the time between an edge on channel 1 and an edge on channel 4. The values are returned in the sequence of the channels, 1 to 16.
Note: the destination array must be dimensioned large enough to hold all the functions requested.</td></tr>
<tr><td>OutputOpt</td><td>Code to select one of the five different output options. The Output Option that is selected will be applied to the data collection for all of the INT8 channels. The numeric code for each option is listed below with a brief explanation of each. See the INT8 manual for detailed explanations of each option.
<table>
<tr><th>Code</th><th>Result</th></tr>
<tr><td>0:</td><td>Average of the event data since the last time that the INT8 was interrogated by the datalogger. If no edges were detected, 0 will be returned for frequency and count functions, and 99999 will be returned for the other functions. The INT8 ceases to capture events during communications with the logger, thus some edges may be lost.</td></tr>
<tr><td>32768</td><td>Continuous averaging, which is utilized when input frequencies have a slower period than the execution interval of the datalogger. If an edge was not detected for a channel since the last time that the INT8 was polled, then the datalogger will not update the input location for that channel. The INT8 will capture events even during communications with the datalogger.</td></tr>
<tr><td>nnnn</td><td>Averages the input values over "nnnn" milliseconds. The datalogger program is delayed by this instruction while the INT8 captures and processes the edges for the specified time duration and sends the results back to the logger. If no edges were detected, 0 will be returned for frequency and count functions, and 99999 will be returned for the other functions.</td></tr>
</table></td></tr>
</table>

Parameter *& Data Type*	**Enter**	**INT8 INSTRUCTION PARAMETERS**
	-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.
		RESULT CODE / **DEFINITION**
		0 — Bad ROM
		-0 — Bad ROM, & bad RAM
		Positive integer — ROM signature, good RAM
		Negative integer — ROM signature, bad RAM
CaptureTrig *Constant, Variable, or Expression*	This argument is used when the "Capture All Events" output option is used. When CaptureTrig is true, the INT8 will return the last *nnnn* events.	
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.	

SDMSpeed (SDMSpeed)

Changes the rate that the CR9000 uses to clock the SDM data. Slowing down the clock rate may be necessary when long cables lengths are used to connect the CR9000 and SDM devices.

Parameter *& Data Type*	**Enter** **SDMSPEED INSTRUCTION PARAMETER**
SDMSpeed *Constant or variable*	Valid parameters range from 1 to 255 where 1 is the fastest (default) and 255 is the longest period. The default bit period is approximately 2 microseconds. The time per bit, in microseconds, is: BitPeriod = 2 * SDMSpeed + 5.

SDMTrigger

When SDMTrigger is executed, the CR9000 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 CR9000 to synchronize when the measurements are made. Subsequent Instructions communicate with the SDM devices to collect the measurement results. Not all SDM devices support the group trigger; check the manual on the device for more information.

SIO4 (Dest, Reps, Address, Mode, Command, FirstOp, SecOp, ValuesPerRep, Mult, Offset)

This Instruction communicates with theSDM-SIO4 Serial Input Multiplexer. See the SDM-SIO4 Manual for details.

Parameter *& Data Type*	**Enter** **SIO4 INSTRUCTION PARAMETERS**
Dest *Variable or Array*	The Variable in which to store the results of the instruction or, when the instruction is used to send data, this array becomes the data to send. When Reps or multiple values per rep are used, the results are stored in an array with the variable name. An array must be dimensioned to have elements for Reps multiplied by Values per Rep..
Reps *Constant*	The number of repetitions for the measurement or instruction.
Address *Constant*	The address for the SDM-SIO4 (0-15)
Mode *Constant*	The SIO4 port the instruction applies to. Code: 1 = Send/Receive Port 1; 2 = Send/Receive Port 2; 3 = Send/Receive Port 3 Code: 4 = Send/Receive Port 4; 5 = Send to all four ports (global)
Command, FirstOp, SecOp *Constants1*	Commands to SDM-SIO4. See SDM-SIO4 Manual
ValuesPerRep *Constant*	How many values to send or receive
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the results. A multiplier of one and an offset of 0 are necessary to store the values as received. 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.

7.8 Digital I/O

PortSet (ExSlot, Port, State)

This Instruction will set the specified control port on the 9060 Excitation Module high or low. This instruction should not be placed in a conditional statement.

Parameter *& Data Type*	**Enter** **PORTSET INSTRUCTION**
ExSlot *Constant*	The slot that holds the 9060 Excitation Module on which to set the port.
Port *Constant*	The number of the port to set with the instruction.
State *Constant, Variable, or Expression*	The state (high or low) to set the port to. **Value** 0 = **State** Low; **Value** ≠ 0 = **State** High

PulseCount (Dest, Reps, PSlot, PChan, PConfig, POption, Mult, Offset)

This instruction should not be placed in a conditional or in a Slow Sequence Scan. Sets up pulse measurements using the twelve 16 bit counter channels on the CR9070 or the twelve 32 bit counters channels on the CR9071E Counter module. There are three pulse types or configurations that may be measured using these Counter modules:

High Frequency: All twelve pulse channels can be configured for high frequency inputs. This configuration is used for the higher frequency pulse inputs (up to 1 MHz). The pulse count is incremented when the signal rises from below 1.5 VDC to above 3.5 VDC. Because of the input filter's 200 nanosecond time constant, higher frequencies will require larger input transitions. See section 3.4 for additional measurement information. The minimum pulse width that can be detected is 500 ns. The maximum input voltage is ± 20 V.

Low Level AC: The first 8 frequency input channels can be configured for low level ac inputs. This option is used to count the frequency of low level ac signals from such sensors as a magnetic pick up. The minimum input voltage that can be counted is 25 mV RMS. At this minimum voltage, frequencies up to 10 kHz can be measured. For input voltage greater than 50 mV, frequencies up to 20 kHz can be measured. Again, the maximum input voltage is 20 V.

Switch Closure Channels 9 through 12 can be configured as Switch Closure inputs. The switch closure (dry contact) should be connected between the pulse channel and ground. When the contact is open, the pulse channel is pulled to 5 volts through a 100 kOhm pull up resistor. When the contact is closed, the pulse channel is pulled to ground. The count is incremented when the channel is pulled high. The minimum switch close time is 5 msec. The minimum switch open time is 5 msec. The maximum bounce time without being counted is 1 msec open.

Using the Poption, you can configure the output as Counts, Frequency over the scan interval, or as a running average frequency for a set duration. See section 3.4 for more pulse measurement details.

Parameter *& Data Type*	**Enter** **PULSECOUNT INSTRUCTION**
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.
PSlot *Constant*	The number of the slot that holds the 9070/9071E Counter Timer Module for the measurement.
PChan *Constant*	The number of the pulse channel for the measurement.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter PULSECOUNT INSTRUCTION</td></tr>
<tr><td>Pconfig
Constant</td><td>A code specifying the type of pulse input to measure.
<table>
<tr><th>Code</th><th>Pulse Channels</th><th>Input Configuration</th></tr>
<tr><td>0</td><td>1 - 12</td><td>High Frequency</td></tr>
<tr><td>1</td><td>1 - 8</td><td>Low Level AC</td></tr>
<tr><td>1</td><td>9 - 12</td><td>Switch Closure</td></tr>
</table></td></tr>
<tr><td>Poption
Constant</td><td>A code that determines if the raw result (multiplier = 1, offset = 0) is returned as counts or frequency. The running average can be used to smooth out readings when a low frequency relative to the scan rate causes large fluctuations in the measured frequency from scan to another.
<table>
<tr><th>Code</th><th>Result</th></tr>
<tr><td>0</td><td>Counts</td></tr>
<tr><td>1</td><td>Frequency (Hz) counts/scan interval in seconds</td></tr>
<tr><td>>1</td><td>Running average of frequency. The number entered is the time period over which the frequency is averaged in milliseconds.</td></tr>
</table></td></tr>
<tr><td>Mult, Offset
Constant, Variable, Array, or Expression</td><td>A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the TCDiff instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.</td></tr>
</table>

PulseCountReset

PulseCountReset is used to reset the pulse counters and the running averages used in the pulse count instruction. It resets all counters in all installed CR9070/CR9071E Counter and Digital I/O modules. The CR9070's 16 bit counters can count up to decimal 65535. More counts than 65535 result in an over-range condition. The CR9071E's 32 bit counters can count up to 4.29 billion before over-ranging. This should never occur because at the maximum input frequency of 1 MHz, it would take almost 72 minutes before it fills while the CR9000's maximum scan rate is 1 minute. With each scan, the CR9000 reads the counts accumulated since the last scan and then resets the counter. If the scans stop, as in burst operation or in a program with more that one Scan loop, the counter continues to accumulate counts until another scan is initiated or it over-ranges. If the running averaging is in use, the over-range value will be included in the average for the duration of the averaging period (e.g., with a 1000 millisecond running average, the over-range will be the value from the **PulseCount(...)** instruction until 1 second has passed. Resetting the average prior to (re)starting the scan avoids this.

ReadIO (Dest, PSlot, Mask)

ReadIO is used to read the status of selected digital I/O channels (ports) on the CR9070/CR9071E Counter - Timer / Digital I/O Module. There are 16 ports on the CR9070/CR9071E. 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, just looking at the first 8 ports, 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, 6, &B100) ' read port 3 on the CR9070/CR9071E card in slot 6
 ' if port 3 is high then Port3 = 4, if port 3 is low then Port3 = 0
```

Parameter & Data Type	Enter READIO INSTRUCTION PARAMTERS
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.
PSlot *Constant*	The number of the slot that holds the 9070 Counter Timer Module for the measurement.
Mask *Constant*	The Mask allows the read or write to only act on certain ports. The Mask is ANDed with the value obtained from the 9070 when reading and ANDed with the source before writing.

TimerIO (Dest, PSlot, Edges 16–9, Edges 8–1, Function 16–9, Function 8–1, AllDoneFlag)

The TimerIO instruction is used to measure the time between edges (state transitions) on the digital I/O channels of the CR9070/CR9071E Counter and Digital I/O Module as well as on the Pulse channels on the CR9071E module . The transitions can be either on the rising edge (low to high) or falling edge (high to low) of the signal. The states are nominally 0 V low and 5 V high. When TimerIO is the only measurement in a scan and the time since previous channel is measured on 4 channels, the fastest interval is approximately 140 microseconds. . Instruction cannot rep from one module to the next.

There are six functions that can be performed:

The period (msec) of the signal on a channel (CR9070 or CR9071E)

The frequency (hertz) of the signal (CR9071E only)

The time (msec) since an edge on the previous channel (1 number lower) to an edge on the specified channel. (CR9070 or CR9071E)

The time (msec) from an edge on channel 1 to an edge on the specified channel. (CR9070 or CR9071E)

Number of edges since last execution. (CR9071E only)

Number of edges since last edge on channel 1 (CR9071E only). Only one function may be programmed per channel. The number of values returned is determined by the number of channels for which a result is requested.

There are two AllDoneFlag operating modes for this instruction:

One is to check each scan for transitions that have occurred since the previous scan. A flag is set true when there are results for all measurements requested. The resolution of the timing measurement when using the I/O channels (only option when using the CR9070) is the scan interval. When using the CR9071E's Pulse channels, the resolution is 40 nanoseconds.

The second mode is to stay within the instruction until it has results for each measurement requested. In this mode, the resolution when using the I/O channels is approximately (10 microseconds + 15 microseconds * the number of results requested). The resolution for the CR9071's Pulse channels is 40 nanoseconds.

<table>
<tr><td>Parameter
& Data Type</td><td>Enter TIMERIO INSTRUCTION PARAMETERS</td></tr>
<tr><td>Dest
Variable or Array</td><td>Array for results of the measurements.</td></tr>
<tr><td>Pslot
Constant</td><td>The slot that the CR9070/CR9071E module is in.</td></tr>
<tr><td>Edges
Constant</td><td>There are two Edge parameters, 8 digits each, one digit is for either each of the 16 I/O channels on the CR9070/CR9071E Module when programmed with a 0 or a 1, or for the 12 pulse channels when programmed with a 2, 3, 4, or 5. Each digit configures the respective channel to count a transition on the rising edge (from <1.5V to >3.5V) or on the falling edge (from >3.5V to <1.5V).
<table>
<tr><th>Digit</th><th>Edge</th></tr>
<tr><td>0</td><td>Falling, IO channel, I/O 1 to I/O 16</td></tr>
<tr><td>1</td><td>Rising, IO channel, I/O 1 to I/O 16</td></tr>
<tr><td>2</td><td>Falling, high freq, pulse channel (CR9071E only), P1 to P12</td></tr>
<tr><td>3</td><td>Rising, high freq, pulse channel (CR9071E only), P1 to P12</td></tr>
<tr><td>4</td><td>Falling, low level ac, P1 to P8 (or switch closure, P9 -P12), pulse channel (CR9071E only)</td></tr>
<tr><td>5</td><td>Rising, low level ac, P1 to P8 (or switch closure, P9 - P12), pulse channel (CR9071E only)</td></tr>
</table>
The first edge parameter is either for I/O channels 16 to 9 or for Pulse channels 12 to 9 depending on the edge code used. The second edge parameter is either for I/O channels 8 to 1 or for Pulse channels 8 to 1. The digits represent the channels in descending order left to right. For example, 00000101 in the second edge parameter means channels 1 and 3 count rising edges and channels 2 and 4-8 are to count falling edges (this could also be specified as 101, the leading zeros do not need to be entered). Separate instructions are required when programming both I/O and Pulse channels for TimerIO functions. See PulseCount instruction section for description of high freq, low level ac, and switch closure inputs. Instruction cannot rep over to another module.</td></tr>
<tr><td>Function
Constant</td><td>Two parameters, 8 digits each, one digit to program results for each channel.
<table>
<tr><th>Digit</th><th>Results</th></tr>
<tr><td>0</td><td>None</td></tr>
<tr><td>1</td><td>Period (msec)</td></tr>
<tr><td>2</td><td>Frequency (CR9071E P1 to P12 only)</td></tr>
<tr><td>3</td><td>time since previous channel (msec)</td></tr>
<tr><td>4</td><td>time since channel 1 (msec)</td></tr>
<tr><td>5</td><td>count since last execution (CR9071E only)</td></tr>
<tr><td>6</td><td>count since channel 1 (CR9071E only)</td></tr>
</table>
The digits correspond to the channels using the same layout outlined for the edge parameter. For example, 00004301 in the second function parameter means to return 3 values: the period for channel 1, (nothing for channel 2) the time between an edge on channel 2 and an edge on channel 3, and the time between an edge on channel 1 and an edge on channel 4. The values are returned in the sequence of the channels, 1 to 16.
Note: the destination array must be dimensioned large enough to hold all the functions requested.</td></tr>
<tr><td>AllDoneFlag
Constant or Variable</td><td>If a variable is entered for this parameter, the variable will be set to TRUE (-1.0) when all the functions have a value. This allows a user program to test when the experiment is complete. If a constant is entered, then the task sequencer will stay in this instruction until values can be returned for all channels.</td></tr>
</table>

WriteIO (PSlot, Mask, Source)

WriteIO is used to set the status of selected digital I/O channels (ports) on the CR9070/CR9071E Counter - Timer / Digital I/O Module. (See PortSet for setting the Output channels on the CR9060.) There are 16 ports on the CR9070/CR9071E. 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, just looking at the first 8 ports, 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.

Examples

```
WriteIO (5, &B100, &B100)    ' Set port 3 on the 9070 in slot 5 high.

WriteIO (5, &Hff00, Y*256)    ' Write Y to upper 8 ports (9-16)
```

Parameter *& Data Type*	**Enter WRITEIO INSTRUCTION PARAMETERS**
PSlot *Constant*	The number of the slot that holds the CR9070/CR9071E Counter Timer Module for the measurement.
Mask *Constant*	The Mask allows the read or write to only act on certain ports. The Mask is ANDed with the value obtained from the 9070 when reading and ANDed with the source before writing.
Source *Constant Variable*	The Variable or number that is to be written to the I/O ports.

7.9 CR9052DC Filter Module Measurements

The CR9052DC is a six-channel, analog-input module that includes programmable anti-alias filtering and dc excitation. The CR9052DC can provide filtered voltage measurements or spectra from fast fourier transforms of the voltage measurements. See Section 3.3 for additional measurement details.

The filter module collects alias-free, 50-kHz samples from each of its six analog-to-digital converters; applies additional real-time, finite-impulse-response filtering, and decimates (down samples) the 50-kHz data to the programmed scan rate. The Filter Module supports 726 different scan intervals including the basic ones shown in the table below. For scan intervals not listed, enter the scan interval desired, download the program, and the logger will return suggested operational scan intervals close to the one that was entered.

Scan Interval		Scan Rate	
20	µs	50	kHz
40	µs	25	kHz
100	µs	10	kHz
200	µs	5	kHz
400	µs	2.5	kHz
1	ms	1	kHz
2	ms	500	Hz
4	ms	250	Hz
10	ms	100	Hz
20	ms	50	Hz
40	ms	25	Hz
100	ms	10	Hz
200	ms	5	Hz

VoltFilt (Dest, Reps, Range, FSlot, Chan, FiltOption, Excitation, Mult, Offset)

The VoltFilt instruction is used to obtain voltage measurements from the 9052 Filter Module in much the same way as the VoltDiff instruction is used with the CR9050 module. The program scan interval (or the SubScan Interval, see SubScan) determines the filter module output interval. Data are passed from the Filter Module to the CR9000 CPU for processing and final storage at this scan interval. There is the option of turning on a fixed excitation (10V, 5V, or 10 mA). No ratiometric scaling (as in the bridge measurement instructions) is applied when the excitation is on; the VoltFilt instruction always returns millivolts scaled by the multiplier and offset.

Parameter *& Data Type* — **Enter VOLTFILT PARAMETERS**

Dest *Variable,Array*

The Variable 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 Reps.

Reps *Constant*

The number of times to repeat the measurement on subsequent CR9052 channels..

Range

Constant

The voltage range for the measurement. The CR9052 normally replaces out-of-range measurements with not-a-number (NaN) which is displayed in PC9000 as **Range?**. Users may choose to have out-of-range measurements to be replaced by the analog-to-digital converter saturation value with a special code in **FiltOption.**

Alpha Code	Numeric Code	Voltage Range	Module Excitation Board Supported
mV5000	0	± 5000 mV	CR9052DC, CR9052IEPE
mV1000	1	± 1000 mV	CR9052DC, CR9052IEPE
mV200	4	± 200 mV	CR9052DC
mV50	5	± 50 mV	CR9052DC
mV20	6	± 20 mV	CR9052DC

Fslot *Constant*

The number of the slot that holds the CR9052 Module to be used for the measurement.

Chan *Constant*

The CR9052 channel number on which to make the first measurement. When **Reps** are used, subsequent measurements will be automatically made on the following differential channels.

FiltOption *Constant*

The sample ratio for the measurement (how many measurements are made within one cycle of the highest frequency in the pass band). The sample ratio determines the top of the pass-band (F_{PASS}) and the beginning of the stop-band (F_{STOP}) of the anti-aliasing low-pass filter relative the sample rate (F_{SAMPLE}). The sample rate is the inverse the scan interval in the CRBASIC program. **FiltOption** must be the same for all channels of a single CR9052 Filter Module. Out-of-range measurements may be replaced by the analog-to-digital converter saturation value by adding 1000 to the **FiltOption** codes shown below.

NumericCode	Sampling Ratio	F_{PASS}	F_{STOP}
2	2.5	F_{SAMPLE}/2.5	F_{SAMPLE}/2.01
5	5	F_{SAMPLE}/5	F_{SAMPLE}/3.37
10	10	F_{SAMPLE}/10	F_{SAMPLE}/5.08
20	20	F_{SAMPLE}/20	F_{SAMPLE}/6.81
1*	2.155	23.2 Khz	26.8 kHz

*Option 1 has no additional filtering beyond the CR9052DC analog front-end and the sigma-delta A/D converter, thus freeing the CR9052DC on-board digital signal processor for additional processing. F_{SAMPLE} must be 50 kHz to use this filter option.

Excitation *Constant*

The continuous, dc, output for the excitation channel(s). If **Reps** is greater than one, then the same excitation will be output on sequential excitation channels.

	Numeric Code	Alpha Code	Output Level	IEPE Freq. Response
CR9052IEPE	605	*None*	Constant 6 mA	0.3 Hz to 20 kHz (τ = 5.0 Sec)
CR9052IEPE	405	*None*	Constant 4 mA	0.3 Hz to 20 kHz (τ = 5.0 Sec)
CR9052IEPE	205	*None*	Constant 2 mA	0.3 Hz to 20 kHz (τ = 5.0 Sec)
CR9052IEPE	600	*None*	Constant 6 mA	3.0 Hz to 20 kHz (τ = 0.5 Sec)
CR9052IEPE	400	*None*	Constant 4 mA	3.0 Hz to 20 kHz (τ = 0.5 Sec)
CR9052IEPE	200	*None*	Constant 2 mA	3.0 Hz to 20 kHz (τ = 0.5 Sec)
CR9052DC	7	*V10*	Constant 10 V DC	N/A
CR9052DC	5	*None*	Constant 5 V DC	N/A
CR9052DC	2	*None*	Constant 10 mA	N/A
CR9052DC	1	*None*	None	N/A

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

A multiplier and offset by which to scale the measurement. A multiplier of one and an offset of 0 will return the measurement in millivolts.

The following example program measures 6 channels on the CR9052DC using the **VoltFilt** instruction.

```
'    CR9052 example program #1
'
'    Measure six channels at 1 kHz on +/- 5000 mV range with 5-Volt excitation.
'     Sample ratio is 2.5: top of pass band is 1 kHz / 2.5 = 400 Hz.
'     CR9052 is in slot 8.
'     Turn on flag 1 on to save instantaneous data to output table.

Public sig_in (6)
units sig_in = mV

Public flag (1)

'------------------- Data Tables -------------
DataTable (FiltData, flag (1), -1)            ' save to final storage if flag (1) = True
    Sample (6, sig_in(1),IEEE4)
EndTable

'------------------- Program ---------------
BeginProg
    Scan(1, msec, 0,0)
'       VoltFilt (Destination, Reps, Range,  Fslot,     Chan, FiltOption,   Excitation, Mult, Offset)
        VoltFilt (sig_in(1),    6,    mV5000, 8,         1,        2,             5,        1.0,   0.0)
        CallTable FiltData
     Next Scan
EndProg
```

SubScan (SubInterval, Units, SubRatio)

The **SubScan** instruction makes it possible to measure CR9052DC inputs at one rate and measurements on other modules at a slower rate, all within the same scan.

Parameter *& Data Type*	**Enter** **SUBSCAN INSTRUCTION PARAMETERS**
SubInterval *Constant*	The time interval at which to run the subscan. The interval must be one of the valid intervals for the CR9052 module: 20, 40, 100, 200, or 400 microseconds or 1, 2, 4, 10, 20, 40, 100, or 200 milliseconds. When used with the CR9052 Filter Module, the interval of the scan that contains the SubScan must be an integral multiple of the SubScan interval.
Units *Constant*	The units for the Interval

Alpha Code	**Numeric Code**	**Units**
USEC	0	Microseconds
MSEC	1	Milliseconds

SubRatio *Constant*	The subscan will run **SubRatio** times each time the scan runs. When SubScan is used with the CR9052 Filter Module (the only use as of March 2001) this parameter is redundant but must be entered anyway. (The Scan interval must be an integral multiple of the SubScan interval or a compile error will occur. SubRatio is the ratio between the scan interval and the subscan interval.)

The following program uses the **SubScan** to combine 2.5 kHz filtermodule measurements with 10 Hz measurements on a 9050 card.

```
' CR9052 example program #2
'
' Measure 2 channels on the CR9050 at 10 Hz on the +/- 5000 mV range.
'
' Measure six channels on the CR9052DC at 2.5 kHz on +/- 5000 mV range with 5-Volt excitation.
'   Sample ratio is 2.5: top of pass band is 2.5 kHz / 2.5 = 1 kHz.
'   CR9052 is in slot 8.
'   Turn on flag 1 on to save instantaneous data to output table.

const stats_interval = 2     ' time period over which to compute stats, in seconds

Public Flt_in (6)
units Flt_in = mV
Public Alg_in (2)
units Alg_in = mV
Public flag (1)

'Filter Module Filter Option
const SmplRat_2_5 =  2        'Fpass = Fsr/2.5 = 1/(T_scan*2.5)

'------------------- Data Tables -------------
DataTable (FiltData, flag (1), -1)            ' save to final storage if flag (1) = True
    Sample (6, Flt_in(1),IEEE4)
EndTable

DataTable (AlgData, flag (1), -1)             ' save to final storage if flag (1) = True
    Sample (2, Alg_in(1),IEEE4)
EndTable

'------------------- Program------------------
BeginProg
    Scan (100, msec, 0, 0)
        VoltDiff (Alg_in(1), 2, mV5000, 6, 1, False, 0, 0, 1.0, 0.0)
        CallTable AlgData
            SubScan (400, usec, 250)
'               VoltFilt (Destination, Reps, Range,  FSlot,      Chan, FiltOption,  Excitation, Mult, Offset)
                VoltFilt (Flt_in(1),     6,    mV5000,   8,           1,  SmplRat_2_5,  5,        1.0,  0.0)
                CallTable FiltData
        Next SubScan
    Next Scan

    SlowSequence
        Scan (1, Sec, 0, 0)
             Calibrate              ' run calibrations for cr9050 measurements
            BiasComp
        Next Scan
EndProg
```

Bursting to CPU and PAM Memory

The maximum continuous throughput rate for measurements made by the CR9052DC to be processed and stored by the CR9031 CPU is 2500 Hz for 1 to 24 channels. The measurement rate can be increased by bursting data into CPU memory or PC cards in the CR9080 PAM Module. In this mode, the CR9031 and CR9052DC support a sustained aggregate sample rate (the sum of the sample rates on all channels) of 50 ksamples/sec (100 ksamples/sec to rotating-media PC cards). The burst buffer size is limited by the capacity of the CPU memory, or by the capacity of the PC card. The triggered burst option allows experimenters to save only the data of interest before, during, or after, a trigger event.

The following program bursts CR9052DC measurements onto a PC card.

```
' CR9052 example program #3
' Measure 5 channels on the CR9050 at 10 kHz on the +/- 5000 mV range with triggered
'   burst buffering on a PCMCIA card.
'   Burst trigger when channel 1 exceeds 4000 mV, or when flag 1 is on.
'   Subsequent recordings are appended to end of preceding recording.
const pam_slot = 4
const cr9052_slot = 8
const sample_len = 1000                         ' record 1000 samples for each trigger
const scans_past_trig = sample_len-200          ' this makes the trigger the ~200th sample point
Public Flt_in (5)
units Flt_in = mV
Public flag (1)
'------------------- Data Tables -------------
DataTable (FiltData, True, -1)
    DataInterval (0, 0, usec, 100)          ' do not explicitly save the time stamp with each record,
                            'data can still be collected to the pc with time stamps
    PamOut(PAM_Slot, CardA, 0, -1)      ' data table is ring memory, maximum size
    Sample (5, Flt_in(1), ieee4)
EndTable

'------------------- Program--------------------
BeginProg
    ResetTable (FiltData)               ' start with fresh data table
    while (True)
        Scan(100, usec, 1, sample_len)
            BeginBurstTrigger (1)
'               VoltFilt (Dest,    Reps, Range, FSlot,     Chan, FiltOption, Excit,  Mult, Offset)
                VoltFilt (Flt_in(1), 5,  mV5000, cr9052_slot, 1,       2,        1,     1.0,    0.0)
            EndBurstTrigger ((Flt_in(1)>4000) or flag(1), scans_past_trig)
            CallTable FiltData
        Next Scan
        Flag(1) = False         ' turn flag 1 off to eliminate multiple triggers
    wend
EndProg
```

Bursting to Filter Module Memory

Each CR9052DC Filter Module includes an 8-Msample (32-Mbyte) burst memory buffer. Experimenters may use this memory to increase CR9052DC measurement rates to 50 ksamples/sec per channel, giving a sustained aggregate sample rate of 300 ksamples/sec for a single Filter Module, 600 ksamples/sec for two Filter Modules, etc. The 8-Msample buffer allows 26.7-second recordings for six channels running at 50 kHz, 80-second recordings for two channels running at 20 kHz, etc. Because each CR9052DC Filter Module includes its own burst memory buffer, the total burst buffer capacity increases as experimenters add additional Filter Modules within the CR9000 chassis.

The Filter Module will burst measurements to its on-board memory when the CRBASIC program specifies the burst option in the **Scan** instruction, and when the program contains a **VoltFilt** instruction within a **SubScan**.

To monitor trigger conditions, the CR9031 CPU collects data from the CR9052DC Filter Module at the scan rate (not the subscan rate). While the Filter Module passes this decimated data to the CPU, it also stores non-decimated measurements in its on-board burst buffer at the subscan rate. Once a trigger has occurred and the CR9052DC has recorded the appropriate number of scans after the trigger, the CPU collects and processes the stored burst data from the CR9052DC memory.

The following example program uses the **SubScan** instruction to buffer measurements into the CR9052DC burst memory.

```
' CR9052 example program #4
'' Measure 6 channels on the CR9050 at 25 kHz on the +/- 5000 mV range with triggered
'   burst buffering on CR9052 memory.
'   Trigger when channel 1 exceeds 4000 mV, or when flag 1 is on.
'   Subsequent recordings are appended to end of the preceding recording in table FiltData.
const pam_slot = 4
const cr9052_slot = 8
const num_scans = 100        ' record 100*25 = ~2500 samples for each trigger
                             '  (the number of samples recorded is *25 because
                             '   the subscan ratio is 25, see below)
const scans_after_trig = 95  ' this makes the trigger the (100-95)*25 ~= 125th
                             '   sample point
Public Flt_in (6)
units Flt_in = mV
Public flag (1)
'------------------- Data Tables -------------
DataTable (FiltData, True, -1)
    DataInterval (0, 0, usec, 100)'do not explicity save the time stamp with each record,
                                  ' data can still be collected to the pc with time stamps
    PamOut(PAM_Slot, CardA, 0, -1)       ' data table is ring memory, maximum size
    Sample (6, Flt_in(1), ieee4)
EndTable
'------------------- Program----------------
BeginProg
    ResetTable (FiltData)       ' start with fresh data table
    while (True)
        Scan(1, msec, 1, num_scans)
            SubScan (40, usec, 25)
                BeginBurstTrigger (1)
'                   VoltFilt (Dest,   Reps, Range, FSlot,     Chan, FiltOption, Excit,  Mult, Offset)
```

```
                VoltFilt (Flt_in(1), 6,mV5000, cr9052_slot,  1,    2,          1,       1.0,     0.0)
                if (Flt_in(1) > 4000) then flag(1) = True
            EndBurstTrigger (flag(1), scans_after_trig)
            CallTable FiltData
          Next SubScan
        Next Scan
        Flag(1) = False      ' turn flag 1 off to eliminate multiple manual triggers
    wend
EndProg
```

FFTFilt (Dest, Reps, Range, Fslot, Channel, FiltOption, Excitation, Mult, FSampRate, FFTLen, TSWindow, SpectOption, Fref, SBin, lLow, lHigh)

The CR9052 filter module can perform real-time fast Fourier transform (FFT) analyses on the voltages measured on its inputs, and then pass the resulting spectra to the CR9000 CPU for further processing and storage into data tables. The FFT operation is specified with the **FFTFilt** instruction.

With the **VoltFilt** instruction the Scan (or SubScan) interval determines the rate at which individual measurements are passed to the CPU. With **FFTFilt** the Scan interval is how often an entire spectrum for each channel is sent to the CPU. The sample rate for the FFT time-series is set within the instruction.

FFTFilt can provide spectra from "seamless" time-series snapshots if the Scan interval is set equal to it's minimum value: the FFT length divided by the time-series sample rate (i.e., measurements are continuously sampled, an FFT is calculated each time the required number of measurements are sampled, no samples are missed.) When the scan interval is greater than this minimum value there will be gaps between acquiring the FFT time series.

The first eight parameters of the **FFTFilt** instruction are similar to the first eight parameters of **VoltFilt**. The **Fslot**, **FiltOption**, **FSampRate**, and **FFTLen** parameters must be the same for all channels of a single CR9052DC module. The other parameters may be unique for each channel.

Parameter	Enter **FFTFILT PARAMETERS**
Dest *Variable or Array*	The Variable in which to store the results of the instruction. Because **FFTFilt** returns all or part of an entire spectrum (see **ILow** and **IHigh)** for each **Rep**, **Dest** usually must be an array.
Reps *Constant*	The number of times to repeat the measurements and subsequent FFTs on consecutive CR9052DC channels. Spectra from multiple **Reps** are placed head-to-tail in the **Dest** array.
Range *Constant*	The voltage range for the measurement. The CR9052 normally replaces out-of-range measurements with not-a-number (NaN) which is displayed in PC9000 as **Range?**. Users may choose to have out-of-range measurements to be replaced by the analog-to-digital converter saturation value with a special code in **FiltOption.**

Alpha Code	Numeric Code	Voltage Range	Module Excitation Board Supported
mV1000	0	± 5000 mV	CR9052DC, CR9052IEPE
mV1000	1	± 1000 mV	CR9052DC, CR9052IEPE
mV200	4	± 200 mV	CR9052DC
mV50	5	± 50 mV	CR9052DC
mV20	6	± 20 mV	CR9052DC

Fslot *Const*	The number of the slot that holds the CR9052 Module to be used for the measurement.
Chan *Constant*	The CR9052 channel number on which to make the first measurement. When **Reps** are used, subsequent measurements will be automatically made on the following differential channels.

FiltOption *Constant*

The sample ratio for the measurement (how many measurements are made within one cycle of the highest frequency in the pass band). The sample ratio determines the top of the pass-band (F_{PASS}) and the beginning of the stop-band (F_{STOP}) of the anti-aliasing low-pass filter relative the sample rate (F_{SAMPLE}). The sample rate is the inverse the scan interval in the CRBASIC program. **FiltOption** must be the same for all channels of a single CR9052DC Filter Module. The CR9052 normally replaces out-of-range measurements with not-a-number (NaN) which is displayed in PC9000 as **Range?**. Out-of-range measurements may be replaced by the analog-to-digital converter saturation value by adding 1000 to the **FiltOption** codes shown below.

Numeric Code	Sampling Ratio	F_{PASS}	F_{STOP}
2	2.5	F_{SAMPLE}/2.5	F_{SAMPLE}/2.01
5	5	F_{SAMPLE}/5	F_{SAMPLE}/3.37
10	10	F_{SAMPLE}/10	F_{SAMPLE}/5.08
20	20	F_{SAMPLE}/20	F_{SAMPLE}/6.81
1*	2.155	23.2 Khz	26.8 kHz

FiltOption 1 is available only when **FsampRate** is 50 kHz. At this sample rate, no additional filtering beyond that provided by the CR9052DC hardware is required to anti-alias the data. Because the CR9052DC processor is not performing additional anti-alias filtering, this **FiltOption** increases the CR9052DC's FFT throughput. To achieve spectra from seamless snapshots with **FsampRate** equal to 50 kHz on six channels, **FiltOption** must be 1.

Parameter — **Enter** **FFTFILT PARAMETERS**

Excitation
Constant

The continuous, dc, output level for the excitation channel(s). If **Reps** is greater than one, then the Filter Module drives the same excitation level on sequential excitation outputs.

Numeric Code	Alpha Code	Output Level	IEPE Freq. Response
CR9052IEPE 605	*None*	Constant 6 mA	0.3 Hz to 20 kHz (τ = 5.0 Sec)
CR9052IEPE 405	*None*	Constant 4 mA	0.3 Hz to 20 kHz (τ = 5.0 Sec)
CR9052IEPE 205	*None*	Constant 2 mA	0.3 Hz to 20 kHz (τ = 5.0 Sec)
CR9052IEPE 600	*None*	Constant 6 mA	3.0 Hz to 20 kHz (τ = 0.5 Sec)
CR9052IEPE 400	*None*	Constant 4 mA	3.0 Hz to 20 kHz (τ = 0.5 Sec)
CR9052IEPE 200	*None*	Constant 2 mA	3.0 Hz to 20 kHz (τ = 0.5 Sec)
CR9052DC 7	*V10*	Constant 10 V DC	
CR9052DC 5	*None*	Constant 5 V DC	
CR9052DC 2	*None*	Constant 10 mA	
CR9052DC 1	*None*	None	

Mult
Constant,Variable, Array, Expression

A factor by which to multiply the raw time-series voltage measurements. **Mult**$^{-1}$ provides the reference value for the deciBell (dB) spectral option.

FsampRate
Constant

The sample rate in samples per second, at which the CR9052 will collect time series data before performing the FFT. **FsampRate** must be the same for all channels in a CR9052 module.

FsampRate	Sample Rate	Sample Interval
50000	50 kHz	20 µs
25000	25 kHz	40 µs
10000	10 kHz	100 µs
5000	5 kHz	200 µs
2500	2.5 kHz	400 µs
1000	1 kHz	1 ms
500	500 Hz	2 ms
250	250 Hz	4 ms
100	100 Hz	10 ms
50	50 Hz	20 ms
25	25 Hz	40 ms
10	10 Hz	100 ms
5	5 Hz	200 ms

FFTLen
Constant

FFTLen
65536
32768
16384
8192
4096
2048
1024
512
256
128
64
32

The length of (number of points in) the time series snapshot on which to perform the FFT. If the scan period equals FFTLen/FsampRate, then the consecutive time seriessnapshots processed into spectra are “seamless”. If the scan period is greater than FFTLen/FsampRate, then the time series snapshots will have gaps between them. A compile error occurs if the scan Period is less than FFTLen/FsampRate

The FFT throughput for transforms of 2048 points or less is much higher than the throughput for transforms longer than 2048 points. The CR9052DC can produce spectra from seamless 50-kHz snapshots on six channels for **FFTLen** equal to 2048 or less. The CR9052DC can produce spectra from seamless 50-kHz snapshots on two channels for any **FFTLen.**

Parameter *& Data Type*

Enter **FFTFILT INSTRUCTION PARAMETERS**

TSWindow *Constant*

TSWindow designates whether the CR9052 should apply a window (also known as a taper, or apodization) function to the time series snapshot before performing the FFT. Typical window functions give more weight to the middle of the time series while tapering the ends to avoid spectral leakage caused by a non-integral number of periods of a repetitive signal in the snapshot.

Numeric Code	Window Function	
0	*None*	
1	Hanning	
2	Hamming	
3	Blackman	
4nn	Kaiser-Bessel	nn: represents Beta (β) nn range: 5 - 16 (integer)

SpectOption *Constant*

Designates the output option for the computed spectrum.

Numeric Code	Spectra Result	Maximum Spectrum Length
0	Real and Imaginary	(FFTLen/2 + 1) pairs
1	Amplitude	(FFTLen/2 + 1) values
2	Amplitude and Phase	(FFTLen/2 + 1) pairs
3	Power Spectrum	(FFTLen/2 + 1) values
4	Power Spectral Density Function	(FFTLen/2 + 1) values
6	RMS Amplitude	(FFTLen/2 + 1) values
7	DeciBels	(FFTLen/2 + 1) values

Add 100 to the 100 to the SpectOption codes above and the original time series data will be returned along with the spectrum. The CR9000X places the time series data in the array **Dest** immediately following the spectrum. When this option is enabled, **Dest** must be dimensioned large enough to hold this additional time series data. If **Reps** is more than one, the CR9000X places the spectrum for the first channel in **Dest**, followed by the time series for the first channel. Next, the CR9000X places the spectrum for the second channel in **Dest**, followed by the time series for the second channel, etc.

FRef *Constant*

Reference Frequency for Logarithmic rebinning. Set to 0 for linear or no rebinning.

SBin *Constant*

For linear rebinning: the number of adjacent spectral bins to combine. For logarithmic rebinning: the number of bins per octave in the rebinned spectrum. Set to 0 or 1 for no rebinning. The DC component, bin 0, is left alone and not combined with other bins. Bin combination starts with the first AC component. Combining bins is not allowed for the Real and Imaginary or Amplitude and Phase spectral options.

ILow, IHigh *Constants*

ILow and IHigh make it possible to return a subset of the spectrum that results from the Spectral option and bin combining specified by the previous parameters. I is the bin number. ILow is the number of first bin to return, IHigh the number of the last bin to return. To get all the components set ILow equal to the lowest bin number and IHigh to the maximum bin number. With linear spectral bins (Fref = 0), the lowest bin number is 0 and the highest bin number is the integer portion of FFTLen/(2*Sbin). See the text for details and logarithmic rebinning (Fref≠0).

Window Function

TSWindow is a constant designating whether the CR9052 should apply a window (also known as taper, or apodization) function to the time series snapshot before performing the FFT. Typical window functions give more weight to the middle of the time series while tapering the ends to avoid spectral leakage caused by a non-integral number of periods of a repetitive signal in the snapshot.

The CR9052 applies the selected window function by multiplying each point of the original time series by the corresponding point of the window function. Because this windowing process removes some of the original signal variation, the CR9052 uses the following procedure to correct the resulting spectra.

The CR9052 first computes the mean and standard deviation of the original time series for use in additional processing. Next, the CR9052 subtracts the mean from each point of the original time series, and then multiplies the mean-subtracted time series by the selected window function. The CR9052 then computes the standard deviation of this windowed time series. The CR9052 then computes the FFT of the windowed time series, and multiplies each ac component of the complex spectrum by the ratio of the standard deviations of the time series computed before and after the window function was applied. The CR9052 then sets the dc component of the spectrum to the mean of the original time series, normalized for the FFT length.

The CR9052 computes the Hanning window function from:

$$0.5 - 0.5\cos\left(\frac{2\pi k}{N-1}\right) \text{ for } 0 \le k \le (N-1).$$

N is the length of the original time series (**FFTLen)**.

The CR9052 computes the Hamming window function from:

$$0.54 - 0.46\cos\left(\frac{2\pi k}{N-1}\right) \text{ for } 0 \le k \le (N-1).$$

The CR9052 computes the Blackman window function from

$$0.42 - 0.5\cos\left(\frac{2\pi k}{N-1}\right) + 0.08\cos\left(\frac{4\pi k}{N-1}\right) \text{ for } 0 \le k \le (N-1).$$

The Kaiser-Bessel window function is calculated using:

$$\frac{I_0\left(\beta\sqrt{1-\left(\frac{k-\frac{N-1}{2}}{\frac{N-1}{2}}\right)^2}\right)}{I_0(\beta)}$$

for $0 \le k \le (N-1)$

where $I_0(\ )$ is the modified zeroth order Bessel function and

where $5 \ge \beta(\text{integer}) \ge 16$

Spectral Options

The CR9052DC supports the following spectral options. The first five spectral options are the same as the CR9000 FFT instruction. RMS Amplitude and Decibels are new for the **FFTFilt** instruction.

Real and Imaginary

The real and imaginary option returns the raw real (r) and imaginary (i) components from the FFT. The FFT calculation produces FFTLen/2 +1 pairs of real and imaginary components. **ILow** and **IHigh,** described below, determine which of these *pairs* of values are loaded into the destination array by **FFTFilt**.

Amplitude

The amplitude option returns the amplitude of each spectral component. The FFT calculation produces FFTLen/2 +1 amplitude components. **ILow** and **IHigh ,**described below, determine the number of values returned by **FFTFilt**. The amplitude of a sinusoid represented by $A\cos(\omega t)$ is A. The CR9052DC computes the amplitude from: $\frac{2\sqrt{r^2 + i^2}}{N}$

for all components except the dc and Nyquist components. The dc and Nyquist components are computed from $\frac{\sqrt{r^2 + i^2}}{N}$.

N is the length of the original time series (**FFTLen)**. The units of the amplitude spectrum are mV.

Amplitude and Phase

The amplitude and phase option returns the amplitude as described above, plus the phase in radians given by: $\tan^{-1}\left(\frac{i}{r}\right)$.

The FFT calculation produces FFTLen/2 +1 pairs of amplitude and phase components. **ILow** and **IHigh,** described below, determine which of these *pairs* of values are returned by **FFTFilt**. The phase is between -π and π.

Power

The power spectrum option gives the power for each of the spectral components. The FFT calculation produces FFTLen/2 +1 power components. **ILow** and **IHigh,** described below, determine the number of values returned by **FFTFilt**. The CR9052DC computes the power from: $\frac{2(r^2 + i^2)}{N^2}$

for all spectral components except the dc and Nyquist components. The dc

component is computed from $\frac{\left(r^2 + i^2\right)}{N^2}$,

and the Nyquist component is computed from $\frac{\left(r^2 + i^2\right)}{2N^2}$.

The sum of all of the ac components of the power spectrum gives the variance of the original time series. The units of the power spectrum are $(mV)^2$.

Power Spectral Density

The power spectral density (PSD) function normalizes the power spectrum by the bandwidth of each spectral component. The FFT calculation produces FFTLen/2 +1 PSD components. **ILow** and **IHigh,** described below, determine the number of values returned by **FFTFilt**. The CR9052DC computes the psd from: $\frac{2\left(r^2 + i^2\right)}{N f_{SR}}$

for all components except the dc and Nyquist components. f_{SR} is the sample rate of the original time series (**FSampRate)**. The dc component is computed from: $\frac{\left(r^2 + i^2\right)}{N f_{SR}}$,

and the Nyquist component is computed from: $\frac{\left(r^2 + i^2\right)}{2N f_{SR}}$.

The integral of the PSD over all of the ac components gives the variance of the original time series. The units of the PSD are $\frac{(mV)^2}{Hz}$.

RMS Amplitude

The RMS (root-mean-square) amplitude is computed from the square root of the power spectrum for all spectral components, or equivalently, the amplitude spectrum divided by the $\sqrt{2}$ for all ac components. The dc component of RMS amplitude spectrum is the same as the dc component of the amplitude spectrum. The FFT calculation produces FFTLen/2 +1 RMS amplitude components. Spectral binning and **ILow** and **IHigh,** described below, determine the number of values returned by **FFTFilt**. The units of the RMS amplitude spectrum are mV RMS.

deciBell

The deciBell (dB) spectrum normalizes the RMS amplitude spectrum according to $20\log_{10}\left(\frac{A}{A_{ref}}\right)$

where A is value from the RMS amplitude spectrum, and A_{ref} is RMS amplitude reference level. The inverse of the multiplier parameter (**Mult^{-1})** of the **FFTFilt** instruction gives A_{ref}. Because the square of the RMS amplitude

is equal to power, an equivalent normalization to dB is $10\log_{10}\left(\frac{P}{P_{ref}}\right)$ where P is the value from the power spectrum, and P_{ref} is power reference level. The square of the inverse of the multiplier parameter (**Mult**$^{-2}$**)** gives P_{ref}. The multiplier parameter of the **FFTFilt** performs two functions for the dB spectrum option. The first function is to convert the raw signal measurements from mV to the units in which the dB reference is specified, and the second function gives the dB reference. For example, users may convert signals from a microphone to sound pressure level (SPL) spectra in dB relative to 20 μPascals RMS, by setting **Mult** to: $\frac{k}{20\times10^{-6}\,\text{Pascals RMS}}$ where k is the microphone calibration in Pascals per mV. The FFT calculation produces FFTLen/2 +1 deciBell components. **ILow** and **IHigh,** described below, determine the number of values returned by **FFTFilt**. The dB spectrum is unitless.

FFT Spectral Bins

The FFT calculation produces N/2 +1 spectral bins, where N is the number of points in the original time series. These bins may contain a single value (i.e., amplitude) or a pair of values (i.e., Real and Imaginary). Each of these bins represents a frequency range. Let i be the bin number, ranging from 0 for the DC component to N/2 for the highest frequency range. The center frequency of each range is: $f_c(i)=\frac{f_{SR}}{N}i$

where f_{SR} is the sample rate of the time series processed by the FFT (parameter **FSampRate**), and N is the length of the FFT (parameter **FFTLen)**. $f_c(0)$ is the center frequency of the first spectral component calculated by the FFT, $f_c(1)$ is the center frequency of the second spectral component, and so on.

The difference between the center frequencies of adjacent spectral bins is $\frac{f_{SR}}{N}$, and bandwidth of each bin is also $\frac{f_{SR}}{N}$.

The results described above are returned by the FFTFilt Instruction when **Fref** is set to zero, **SBin** is either zero or one, ILow is 0, and IHigh equals N/2. ILow and IHigh refer to the bin numbers of the first and last bins to load into the destination array. For example, if the number of points in the original time series, N=1024 then the resulting FFT would have 1024/2 +1 = 513 bins numbered from 0 to 512. To get the entire FFT, ILow would be set to 0 and IHigh would be set to 512.

ILow and IHigh can be used to return only a part of the spectrum. For example, If only the higher frequencies were of interest, say bin 200 to bin 512, ILow could be set to 200 and IHigh to 512.

In terms of frequency:

To limit the lower end of the spectrum, select a minimum frequency of interest, f_{low}, and then set **Ilow** to: $\text{round}\left(\frac{N}{f_{SR}} f_{low}\right)$,

where $\text{round}(x)$ is x rounded to the nearest integer.

To limit the upper end of the spectrum, select a maximum frequency of interest, f_{high}, and then set **IHigh** to: $\text{round}\left(\frac{N}{f_{SR}} f_{high}\right)$.

Not saving the higher frequency bins is particularly useful if you are used to using some of the rules of thumb on over sampling that evolved to avoid aliasing higher frequencies present because of the prolonged rolloff of analog filters. For example, suppose you are interested in frequencies up to 1 kHz. To get a 5 times oversample, FSampRate of 5 kHz is used with FiltOpt sampling ratio = 5 and N=1024. The bin containing the 1 kHz information will be Round((1024/5000)x1000) = 205. Bins containing spectra beyond the filter stop frequency of 5000/3.37 = 1484 Hz will be drastically attenuated (≥ 90 dB). The bin containing the stop frequency is: I = Round ((1024/5000)x1484) = 304). Set IHigh to bin 205 and only spectra up to 1 kHz will be returned. Set IHigh to 304 get the spectra through the filter roll off but discard the 208 bins containing spectra beyond the stop frequency.

The total number of spectral components (spectral *pairs* for real and imaginary, or amplitude and phase, spectral options) loaded into the destination array by FFTFilt is **IHigh - ILow** + 1. Note that the bin numbers ILow and IHigh are not the same as the array index numbers of the destination array. For example, with a single (1Rep) 1024 point Amplitude FFT, if all the bins were returned (ILow=0, and I High=512) into the destination: FFTResult(1), FFTResult(1) would equal the amplitude for bin 0, FFTResult(2) = bin(1), … FFTResult(513) = bin(512). If ILow were set equal 200 and IHigh equal 512, then FFTResult(1) = bin(200), FFTResult(2) = Bin(201), … FFTResult(313) = bin(512).

Frequency Range

Maximum Frequency

The maximum non-attenuated frequency in the FFT is a function of the Sampling Frequency, f_{SR}, (**FSampRate)** and the Filter option (**FiltOption)**

The maximum frequency in the spectrum calculated by an FFT is half the sampling frequency ($f_{SR}/2$). This is also called the Nyquist frequency.

FSampRate must be at least twice the maximum frequency of interest, f_{high}.

Any frequencies higher than the Nyquist frequency that were present in the time series will be aliased, contributing to the lower frequency components. Aliasing is not a concern with the CR9052 because the Pass frequency *and* the stop frequency are both less than FSampRate/2 for all filter options except 1.

Alaising is not a problem with filter option 1 because any signals in the transition band up to the stop frequency of 26.8 kHz will be alaised to frequencies higher than the pass frequency of 23.2 kHz.

The pass frequency (***F*****PASS**) is the maximum frequency that is not attenuated by the filter. Be sure that the selected filter option **FiltOption** in combination with **FSampRate** makes ***F*****PASS** greater than or equal to the maximum frequency of interest, f_{high}. (i.e., that $f_{high} \leq f_{pass}$).

One effect of the filter option used is on the number of spectral bins calculated by the FFT beyond the pass frequency. The pass frequency is defined in terms of the sampling ratio, R_{samp}, the ratio of the sample rate to the pass frequency : $f_{pass} = f_{SR} / R_{samp}$. For the smallest sampling ratio of 2.5, the number of bins representing frequencies greater than f_{pass} is approximately 20% of the bins calculated by the FFT. This goes up to 90% of the calculated bins for the maximum sampling ratio of 20. It is easy to set IHigh to not return bins beyond f_{pass}. However, the fewer calculations required for the same maximum frequency, $f_{\max} = f_{pass}$, when using a sampling ratio of 2.5 vs a sampling ratio of 20 may make the difference between seamless and intermittent FFTs if the FFT length has to be increased at the higher sample rate to obtain the desired minimum frequency.

Minimum Frequency

Once **FsampRate** is selected to include the highest frequency of interest, **FFTLen** can be set to determine the lowest non-zero frequency.

The lowest frequency AC component of an FFT (bin 1 in the description of the FFT Spectra above) has a center frequency, $f_c(1) = \frac{f_{SR}}{N} \times 1 = \frac{f_{SR}}{N}$.

Where f_{SR} is the sample rate (**FsampRate**, samples/second) and N is the number of samples (**FFTLen**). This frequency is the reciprocal of the time required to complete the sampling. In other words, exactly one cycle of this low frequency is completed in the time it takes to sample the time series for the FFT. To be sure the spectrum output by the FFT includes the lowest frequency of interest, f_{low}, set N (**FFTLen**) so that: $\frac{f_{SR}}{N} \leq f_{low}$.

Frequency Resolution

Frequency resolution goes hand in hand with the minimum frequency. The difference between the center frequencies of adjacent spectral bins is $\frac{f_{SR}}{N}$, and bandwidth of each bin is also $\frac{f_{SR}}{N}$.

For a given sample rate, f_{SR}, if better frequency resolution is required (i.e., more bins, each covering a narrower frequency range) increase the number of points in the FFT, N. If less resolution is required (i.e., fewer bins each covering a wider frequency range) decrease the number of points in the FFT or (to keep the minimum frequency from slipping into the DC bin) combine bins as described below.

Spectral ReBinning

An FFT spectrum can be "rebinned" into a spectrum containing fewer bins where each of the new bins contains a component that covers the frequency range of the bins that were combined. The dc component (bin 0 of the original FFT) is not combined with other bins but may be returned with a linear rebinned spectrum. The first bin to be combined is the first ac component. Bins can be combined in two different ways:

1)Linearly with the resulting bins all having a fixed bandwidth equal to the distance between center frequencies of adjacent bins (as in a the spectrum created by the FFT).

2) Logarithmically with the bandwidth increasing with frequency.

The mathematical operations to combine bins depends on the spectrum type (**SpectOption)**. Amplitude, RMS amplitude, and dB spectra are combined by summing the power in the adjacent bins and then converting this summed power to the desired spectrum type (amplitude, RMS amplitude, or dB). Power spectral density (PSD) functions are combined by averaging adjacent frequency-normalized bins into to give the frequency-normalized result. Combining Real and Imaginary, or Amplitude and Phase spectra is not allowed.

Fref and **SBin** are constants that determine the type of spectral binning. **ILow** and **IHigh** are constants that determine which part of the rebinned spectrum is returned.

Linear Spectral Rebinning

Linear spectral rebinning combines the spectral components from a fixed number of adjacent bins into a single component of the final spectrum. Linear spectral rebinning is selected by setting **Fref** equal to zero and **SBin** to two or more. The parameter **SBin** determines the number of bins to combine.

Let i be the bin number of the rebinned spectrum. The center frequency of each spectral component with linear spectral rebinning is

$$f_c(i) = \frac{f_{SR}}{N}\left(i \times S_{bin} - \frac{S_{bin} - 1}{2}\right)$$

Where i ranges from 0 for the DC component to $\text{Floor}\left(\frac{N}{2 \times S_{bin}}\right)$ for the bin containing the highest frequency component. where the $\text{Floor}(x)$ is the largest integer that is not greater than x, f_{SR} is sample rate of the original time series (parameter **FSampRate**), N is the length of the FFT

(parameter **FFTLen)**, and S_{bin} is the number of bins to combine (parameter **SBin**).

The difference between the center frequencies of adjacent spectral components after linear spectral rebinning is $\frac{f_{SR}}{N} S_{bin}$, and bandwidth of each spectral component (except the dc component) is also $\frac{f_{SR}}{N} S_{bin}$. The bandwidth of the dc component is $\frac{f_{SR}}{N}$.

As with the original FFT results, **ILow** and **IHigh** . determine which part of the rebinned spectrum to return. To return the entire spectrum, set **ILow** to its minimum value, 0, and **IHigh** to its maximum value. The maximum **IHigh** is:

$$\text{floor}\left(\frac{N}{2\times S_{bin}}\right)$$

where the $\text{floor}(x)$ is the largest integer that is not greater than x. To limit the lower end of the spectrum, users first select a minimum frequency of interest, f_{low}, and then set **ILow** to $\text{round}\left(\frac{1}{S_{bin}}\left(\frac{N\times f_{low}}{f_{SR}}+\frac{S_{bin}-1}{2}\right)\right)$, where $\text{round}(x)$ is x rounded to the nearest integer. To limit the upper end of the spectrum, users select a maximum frequency of interest, f_{high}, and then set **IHigh** to: $\text{round}\left(\frac{1}{S_{bin}}\left(\frac{N\times f_{high}}{f_{SR}}+\frac{S_{bin}-1}{2}\right)\right)$.

The total number of spectral components returned by the FFTFilt instruction is **IHigh** - **ILow** + 1.

Logarithmic Spectral ReBinning (1/n Octave Analyses)

Logarithmic spectral rebinning combines the spectral components from a variable number of adjacent bins into a single component of the final spectrum. The number of bins that are combined increases logarithmically with frequency. FFTFilt is programmed to return a logarithmic spectrum by setting **Fref** to a non-zero value and **SBin** between one and twelve. The parameter **SBin** determines the number of bins per octave in the rebinned spectrum. An octave is a factor of two increase in frequency.

The dc component is never part of the final logarithmic spectrum.

Let i be the bin number of the rebinned spectrum. The center frequency of each spectral component with logarithmic spectral binning is $f_c(i)=f_{ref}\,2^{\frac{i}{S_{bin}}}$ for $i_{low}\le i\le i_{high}$

where f_{ref} is an arbitrary reference frequency selected by the user (parameter

Fref), and S_{bin} is the bins per octave in the final logarithmic spectrum (parameter **SBin**). In many acoustic applications, **Fref** is set to 1 kHz.

The ratio (not the difference) between center frequencies of adjacent spectral components in the logarithmic spectrum is $2^{\frac{1}{S_{bin}}}$. The absolute bandwidth of each spectral component is not constant, but rather, increases with increasing frequency. The bandwidth of each spectral component, expressed as a fraction of the center frequency, is $2^{\frac{1}{2S_{bin}}} - 2^{\frac{-1}{2S_{bin}}}$.
Many acoustic applications call for 1/3 octave analyses (three points per octave). For this case, the center frequency of a given bin is a factor of about 1.26 greater than the center frequency of the preceding bin. The bandwidth of each bin is about 23 percent of the bin's center frequency.

Note that in this logarithmic spectrum the integer bin number, i, may be negative as well as positive. Fref is the center frequency of bin 0,

$$f_c(0) = f_{ref}\, 2^{\frac{0}{S_{bin}}} = f_{ref}$$

This is not to say that bin 0 is always a valid output. The valid frequency bins to output are determined by frequency range of the original FFT and the values entered for Sbin and Fref (e.g., if the original sample rate (FSampRate) was 1kHz and Fref was entered as 1 kHz bin 0 (1 kHz center frequency) could not be output because the highest frequency in the original FFT is 500 Hz.)

The minimum i is: $\text{ceiling}\left(S_{bin} \frac{\log_{10}\left(\frac{f_{SR}}{N f_{ref}} \right)}{\log_{10}(2)} - \frac{1}{2} \right)$

where $\text{ceiling}(x)$ is the smallest integer that is not less than x. The maximum i is: $\text{floor}\left(S_{bin} \frac{\log_{10}\left(\frac{f_{SR}}{2 f_{ref}} \right)}{\log_{10}(2)} + \frac{1}{2} \right)$

where $\text{floor}(x)$ is the largest integer that is not greater than x.

Users can select whether the CR9052DC returns the entire spectrum or only part of the spectrum by setting **ILow** and **IHigh**. To return the entire spectrum, set **ILow** to its minimum value, and set **IHigh** to its maximum value. As an alternative to computing the minimum **ILow** and maximum **IHigh** from the equations given above, let the CR9000 perform the calculations: Set **ILow** a very negative value (like -1000) and set **IHigh** to a very positive value (like 1000). When the program is downloaded, the CR9000 compiler will issue an error that gives the minimum **ILow** and maximum **IHigh** for the current

FFTFilt programming. These values can then be entered into the program and used to calculate the size required for the destination array.

To limit the lower end of the final spectrum by frequency, select a minimum frequency of interest, f_{low}, and then calculate **ILow:**

$$\textbf{ILow} = \text{round}\left(S_{bin} \frac{\log_{10}\left(\frac{f_{low}}{f_{ref}} \right)}{\log_{10}(2)} \right),$$

where $\text{round}(x)$ is x rounded to the nearest integer.

To limit the upper end of the final spectrum, select a maximum frequency of interest, f_{high}, and then calculate **IHigh:**

$$\textbf{IHigh} = \text{round}\left(S_{bin} \frac{\log_{10}\left(\frac{f_{high}}{f_{ref}} \right)}{\log_{10}(2)} \right).$$

The total number of spectral components returned by FFTFilt is **IHigh** - **ILow** + 1.

FFTSample (Source, DataType)

FFTSample is an output instruction it used to sample a variable array written to by an FFTFilt instruction. FFTSample is used in place of the Sample instruction because gets the FFT programming from the FFTFilt instruction and stores this processing information in the header of the data table. Without the processing information, PC9000 would not be able to automatically detect and plot the FFT.

Parameter *& Data Type*	**Enter** **FFTSAMPLE INSTRUCTION PARAMETERS**		
Source *Variable*	The variable that in the FFTFilt Destination array that contains the start of the spectrum returned by the FFTFilt instruction. This must be the same variable array that was used as the FFTFilt Destination. All of the spectral values returned by the FFTFilt Instruction for that 9052 channel will be output. Separate FFTSample instructions are required to output each of the Reps used in an FFTFilt instruction. The datalogger will return a compile error if it cannot find an FFTFilt instruction which uses this source variable as the destination for a spectrum.		
DataType *Constant*	A code to select the data storage format.		
	Alpha Code	**Numeric Code**	**Data Format**
	IEEE4	24	IEEE 4 byte floating point
	FP2	7	Campbell Scientific 2 byte floating point

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

Absolute Value

Returns the absolute value of a number.

Syntax

Abs(*number***)**

Remarks

The argument *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 Precision, Value, X, X1, X2          'Declare variables.
Precision = .00000000000001
Value = Volt(3)                          'Volt(3) will be evaluated.
X1 = 0: X2 = Value                       'Make first two guesses.
'Loop until difference between guesses is less than precision.
Do Until Abs(X1 - X2) < Precision
X = (X1 + X2) / 2
If X * X * X - Value < 0 Then            'Adjust guesses.
                                         X1 = X
Else
                                         X2 = X
End If
Loop

'X is now the cube root of Volt(3).
```

And Operator

Used to perform a bit-wise conjunction on two numbers.

Syntax

result = *number1* **And** *number2*

The And operator performs a bit-wise comparison of identically positioned bits in two numbers and sets the corresponding bit in result according to the following truth table:

If bit in *number1* **is**	**And bit in** *number2* **is**	**The** *result* **is**
0	0	0
0	1	0
1	0	0
1	1	1

Although AND is a bit wise operator, it is often used to test Boolean (True/False) conditions. The CR200 decides if something is true or false on the criteria that 0 is false and any non-zero number is true (Section 4.5). Because AND is a bit wise operation it is possible to AND two non-zero numbers (e.g., 2 and 4) and get 0. The binary representation of –1 has all bits equal 1. Thus any number AND –1 returns the original number. That is why the pre defined constant, True = -1.

The predefined constant True = -1
The predefined constant False = 0

If *number1* is:	AND *number2* is:	The *result* is:
-1	Any number	*number2*
-1	NAN (not a number)	NAN
0	Any number	0
0	NAN	NAN

Expressions are evaluated to a number (Section 4.5) and can be used in place of one or both of the numbers. Comparison expressions evaluate as True (-1) or False (0) For example:

```
If Temp(1) > 50 AND Temp(3) < 20 Then
  X = True
Else
  X = False
EndIf
```

and

```
X = Temp(1) > 50 AND Temp(3) < 20
```

Both have the same effect, X will be set to –1 if Temp(1) is greater than 50 and Temp(4) is less than 40. X will be set to 0 if either expression is false.

Atn()

Returns the arctangent of a number.

Syntax
Atn(*number***)**

Remarks
The argument *number* can be any valid numeric expression.

The **Atn** function takes the ratio (*number*) 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. Pi is approximately 3.141593.

To convert degrees to radians, multiply degrees by Pi/180. To convert radians to degrees, multiply radians by 180/Pi.

Note Atn is the inverse trigonometric function of Tan, which takes an angle as its argument and returns the ratio of two sides of a right triangle. Do not confuse Atn with the cotangent, which is the simple inverse of a tangent (1/tangent).

Atn FunctionExample

The example uses Atn to calculate Pi. By definition, Atn(1) is 45 degrees; 180 degrees equals Pi radians.

```
Dim  Pi              'Declare variables.
Pi = 4 * Atn(1)      'Calculate Pi.
```

Spatial Average

Computes the spatial average of a measurement.

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** **AvgSpa Parameters**
Dest *Variable*	The variable in which to store the results of the instruction.
Swath *Constant*	The number of values of the source array to average.
Source *Array*	The name of the variable array that is the input for the instruction.

Average Spatial Output Example

This example uses AvgSpa to find the average value of the five elements Temp(6) through Temp(10) and store the result in the variable AvgTemp.

```
AvgSpa(AvgTemp, 5, Temp(6))
```

Running Average

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** **AvgRun Parameters**
Dest *Variable or Array*	The variable or array in which to store the average(s).
Reps *Constant*	When the source is an array, this is the number of variables in the array to calculate averages for. When the source is not an array or only a single variable of the array is to be averaged, reps should be 1.
Number *Constant*	The number of values to include in the running average..
Source *Array*	The name of the variable or array that is to be averaged.

Example

```
BeginProg                          'Program begins here
    Scan( RATE, RUNITS, 0, 0 )     'Scan 1(mSecs),
    '_____________________ Volt Blocks _____________________
    VoltDiff(HiVolts, VREP1, VRNG1, 5, 1, 0, VDLY1, VINT1, VMULT1, VOSET1)
    AvgRun(AvgOut,1,HiVolts,100 )  'Put the average of 100 HiVolts in AvgOut
    CallTable MAIN                 'Go up and run Table MAIN
    Next Scan                      'Loop up for the next scan
EndProg                            'Program ends here
```

Cosine

Returns the cosine of an *angle*.

Syntax
Cos(*angle***)**

Remarks
The argument *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. Pi is approximately 3.141593.

Cos FunctionExample

The example uses Cos to calculate the cosine of an angle with a user-specified number of degrees.

```
Dim Degrees, Pi, Radians, Ans      'Declare variables.
BeginProg
Pi = 4 * Atn(1)                    'Calculate Pi.
Degrees = Volts(1)                 'Get value to convert.
Radians = Degrees * (Pi / 180)     'Convert to radians.
Ans = Cos(Radians)                 ‘The Cosine of Degrees.
EndProg
```

Spatial Covariance

The CovSpa instruction computes the covariance(s) of sets of data that are loaded into arrays.

Syntax

CovSpa(Dest, NumOfCov, SizeOfSets, CoreArray; DatArray)

CovSpa calculates the covariance(s) between the data in the CoreArray and one or more data sets in the DatArray. The covariance of the sets of data X and Y is calculated as:

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

Where n is the number of values in each data set (**SizeofSets**). X_i and Y_i are the individual values of X and Y.

Parameter *& Data Type*	**Enter** **CovSpa Parameters**
Dest *Variable or Array*	The Variable in which to store the results of the instruction. When multiple covariances are calculated, the results are stored in an array with the variable name. An array must be dimensioned to at least the value of NumOfCov.
NumOfCov *Constant*	The number of covariances to be calculated. If four data sets are to be compared against a fifth set, this would be set to four.
SizeOfSets *Constant*	The number of values in the data sets for the covariance calculations.
CoreArray Array	The array that holds the core data set. The covariance of core data with each of the other sets is calculated independently. The data need to be consecutive in the array. If the first data value is not the first point of the array, the first point of the data set must be specified in this parameter.
DatArray *Array*	The array that contains the data set(s) for calculating the covariance with the CoreSet. When multiple covariances are calculated, the data sets have to be loaded consecutively into one array. The array must be dimensioned to at least the value of NumOfCov multiplied by SizeOfSets. For example, if each set of data has 100 elements (SizeOfSets), and there are 4 covariances (NumOfCov) to be calculated, then the DatArray needs to be dimensioned to 4 x 100 = 400. If the first value of the first set is not the first point of the array, the first point of the data set must be specified in this parameter.

Exp

Returns e (the base of natural logarithms) raised to a power.

Syntax

Exp(*number***)**

Remarks

If the value of *number* exceeds 709.782712893, an Overflow error occurs. The constant e is approximately 2.718282.

Note The Exp function complements the action of the Log function and is sometimes referred to as the antilogarithm.

Exp FunctionExample

The example uses Exp to calculate the value of e. Exp(1) is e raised to the power of 1.

'Exp(x) is e ^x so Exp(1) is e ^1 or e.

```
Dim ValueOfE      'Declare variables.
BeginProg
ValueOfE = Exp(1)      'Calculate value of e.
EndProg
```

FFTSpa (Dest, N, Source, Tau, Units, Option)

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

The difference between the FFT instruction (Section 6) and FFTSpa is that FFT is an output instruction that stores the results in a data table and FFTSpa stores its results in an array.

Parameter *& Data Type* — **Enter** **FFTSPA Parameters**

Dest *Array*
The array in which to store the results of FFT.

Source *Variable*
The name of the Variable array that contains the input data for the FFT.

N *Constant*
Number of points in the original time series. The number of points must be a power of 2 (i.e., 512, 1024, 2048, etc.).

Tau *Constant*
The sampling interval of the time series.

Units *Constant*
The units for Tau.

Alpha Code	Numeric Code	Units
USEC	0	Microseconds
MSEC	1	Milliseconds
SEC	2	Seconds
MIN	3	Minutes

Options *Constant*
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 - ϕ); 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 - ϕ), 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 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 real time series. The output is the time series whose FFT would result in the input array.

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

Normalization details:

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

Notes:

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

Fractional Part

Returns the fractional part of a number.

Syntax
Frac(*number***)**

Remarks
Returns the fractional portion of the *number* within the parentheses.

Frac FunctionExample

The example uses Frac function.

Get Record

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** **GetRecord Parameters**
Dest *Array*	The destination variable array in which to store the fields of the record. The array must be dimensioned large enough to hold all the fields in the record.
TableName *name*	The name of the data table to retrieve the record from.
RecsBack *Const. Or variable*	The number of records back from the most recent record stored to go to retrieve the record (1 record back is the most recent).

GetRecord Example

IfTime

The IfTime instruction is used to return a number indicating True (-1) or False (0) based on the datalogger's real-time clock.

Syntax

IfTime (TintoInt, Interval, Units)

The IfTime function returns True (-1) or False (0) based on the scan clock. Time is kept internally by the datalogger as the elapsed time since January 1, 1990, at 00:00:00 hours. The interval is synchronized with this elapsed time (i.e., the interval is true when the Interval divides evenly into this elapsed time). The time into interval allows an offset to the interval. The IfTime instruction can be used to set the value of a variable or it can be used as an expression for a condition.

The scan clock that the IfTime function checks has the time resolution of the scan interval (i.e., it remains fixed for an entire scan and increments for the next scan). IfTime must be within a scan to function.

The window of time in which the IfTime instruction is true is 1 of its specified **Units**. For example, if IfTime specifies 0 into a 10 minute interval, it could be true when the scan clock specified any time within the first minute of the ten minute interval. With 0 into a 600 second interval, the interval is still 10 minutes but it could only be true during the first 1 second of that interval.

IfTime will only return true once per interval. For example, a program with a 1 second scan that tests IfTime(0,10, min) -- 0 minutes into a 10 minute interval – each scan will execute the instruction 60 times during the minute that it could be true. It will only return true the first time that it is executed, it will not return true again until another interval has elapsed.

Parameter *& Data Type*	**Enter** **IfTime Parameters**
TintoInt *constant*	The time into interval sets an offset from the datalogger's clock to the interval at which the IfTime will be true. For example, if the Interval is set at 60 minutes, and TintoInt is set to 5, IfTime will be True at 5 minutes into the hour, every hour, based on the datalogger's real-time clock. If the TintoInt is set to 0, the IfTime statement is True at the top of the hour.
Interval *constant*	The Interval is how often IfTime will be True.
Units *Constant*	The time units for **TintoInt** and **Interval** **Alpha Code** / **Numeric Code** / **Units** Sec / 2 / seconds Min / 3 / minutes Hr / 4 / hours Day / 5 / days

IIF

The IIF function evaluates a variable or expression and returns one of two results based on the outcome of that evaluation.

Syntax

Result = **IIF**(Expression, TrueValue, FalseValue)

Parameter *& Data Type*	**Enter** **IIf Parameters**
Expression *Expression or Variable*	The Variable or expression to test. Value / Result ≠0 / True: return **TrueValue** 0 / False: return **FalseValue**
TrueValue *Constant Variable or Expression*	The Value (or expression determining the value) to return if the test condition is true
FalseValue *Constant Variable or Expression*	The Value (or expression determining the value) to return if the test condition is False

Int, Fix Functions

Return the integer portion of a number.

Syntax

Int(*number***)**

Fix(*number***)**

Remarks

The argument *number* can be any valid numeric expression. Both **Int** and **Fix** remove the fractional part of *number* and return the resulting integer value.

If the numeric expression results in a Null, **Int** and *Fix* return a Null.

The difference between **Int** and **Fix** is that if *number* is negative, **Int** returns the first negative integer less than or equal to *number*, whereas **Fix** returns the first negative integer greater than or equal to *number*. For example, **Int** converts -8.4 to -9, and **Fix** converts -8.4 to -8.

Fix(*number***)** is equivalent to:

Int and Fix Function Example

This example illustrates the use of Int and Fix.

```
Dim A, B, C, D      'Declare
variables.
BeginProg
A = Int(-99.8)      'Returns -
100
B = Fix(-99.8)      'Returns -99
C = Int(99.8)       'Returns 99
D = Fix(99.8)       'Returns 99
EndProg
```

Log Function

Returns the natural logarithm of a number.

Syntax

Log(*number***)**

Remarks

The argument *number* can be any valid numeric expression that results in a value greater than 0. The natural logarithm is the logarithm to the base e. The constant e is approximately 2.718282.

You can calculate base-n logarithms for any *number* x by dividing the natural logarithm of x by the natural logarithm of n as follows:

Logn(x) = **Log**(x) / **Log**(n)

The following example illustrates a procedure that calculates base-10 logarithms:

Log10 = **Log**(X) / **Log**(10)

Log Function Example

The example calculates the value of e, then uses the Log function to calculate the natural logarithm of e to the first, second, and third powers.

```
Dim I, M                          'Declare variables.
BeginProg
M = Exp(1)
For I = 1 To 3                    'Do three times.
      M =Log(Exp(1) ^ I)
Next I
EndProg
```

Memory Test

Stores the results of the most recent memory test in a variable.

Syntax
MemoryTest (Dest**)**

Remarks
The CR9000 tests CPU RAM and Task Sequencer memory when it compiles and runs a program. MemoryTest stores the results of this compile test in a variable

Parameter *& Data Type*	**Enter** **MemoryTest Parameters**	
Dest *Variable*	The variable in which to store the results of the memory test	
	Result	**Meaning**
	0	No problems found
	1	Error in CPU RAM
	2	Error in Task Memory
	3	Error in both CPU RAM and Task Memory

Spatial Maximum

Finds the maximum value in a swath of variables in an array.

Syntax
MaxSpa(Dest, Swath, Source**)**

Remarks
Find the maximum value in the given array and place the result in the array 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** **MaxSpa Parameters**
Dest *Array*	The array in which to store the maximum value. The next element in the destination array will be loaded with which element of the swath (ranging from 1 to swath) held the maximum value.
Swath *Constant*	The number of values of the source array in which to search for the maximum.
Source *Array*	The element of the source array in which to start looking for the maximum.

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 MaxTemp(1) and MaxTemp(2).

```
MaxSpa(MaxTemp(1), 5, Temp(6))
```

Spatial Minimum

Finds the minimum value in a swath of variables in an array.

Syntax

MinSpa(Dest, Swath, Source**)**

Remarks

Find the minimum value in the given array and place the result in the array 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** **MinSpa Parameters**
Dest *Array*	The array in which to store the minimum value. The next element in the destination array will be loaded with which element of the swath (ranging from 1 to swath) that held the minimum value.
Swath *Constant*	The number of values of the source array in which to search for the minimum.
Source *Array*	The element of the source array in which to start looking for the minimum.

MinSpa Function Example

This example uses MinSpa to find the minimum value of the five elements Temp(6) through Temp(10) and store the results in MinTemp(1) and MinTemp(2).

```
MinSpa(MinTemp(1), 5, Temp(6))
```

Mod

Divides two numbers and returns only the remainder.

Syntax

result = operand1 **Mod** *operand2*

Remarks

The modulus, or remainder, operator divides *operand1* by *operand2* (rounding floating-point numbers to integers) and returns only the remainder as *result*. For example, in the expression A = 19 **Mod** 6.7, A (which is result) equals 5. The operands can be any numeric expression.

Mod Operator Example

The example uses the Mod operator to determine if a 4-digit year is a leap year.

```
Dim TestYr, LeapStatus                          'Declare variables.
TestYr = 1995
If TestYr Mod 4 = 0 And TestYr Mod 100 = 0 Then  'Divisible by 4?
    If TestYr Mod 400 = 0 Then                   'Divisible by 400?
        LeapStatus = True
    Else
        LeapStatus = False
    End If
ElseIf TestYr Mod 4 = 0 Then
    LeapStatus = True
Else
    LeapStatus = False
End If
```

Move

Moves a block or fills an array.

Syntax
Move(Dest, Reps, Source, Reps**)**

Remarks
Block Move or fill array.

Parameters: Dest array, destination reps; Source array or expression; Source reps. If source reps is less than destination reps, the remainder of destination is filled with that last value of source.

Parameter *& Data Type*	**Enter** **Move Parameters**
Dest *Variable or Array*	The variable in which to store the results of the instruction.
Reps *Constant*	The number of repetitions for the measurement or instruction.
Source *Array*	The name of the variable array that is the input for the instruction.
Reps *Constant*	The number of repetitions for the measurement or instruction.

Move Function Example

The example uses the Move function.

Move(x, 20, y, 20**)** 'move array y into array x
Move(x, 20, 0.0, 1**)** 'fill x with 0.0.

NOT

The NOT function is used to perform a bit-wise negation on a number.

Syntax
result = NOT (number)

The NOT operator inverts the bit values of any variable and sets the corresponding bit in result according to the following truth table:

If bit is	The result is
0	1
1	0

Although NOT is a bit wise operator, it is often used to test Boolean (True/False) conditions. The CR200 decides if something is true or false on the criteria that 0 is false and any non-zero number is true (Section 4.5). Because NOT is a bit wise operation, the only non-zero number that NOT can operate on and return 0 is –1. The binary representation of –1 has all bits equal 1. That is why the pre defined constant, True = -1.

The predefined constant True = -1
The predefined constant False = 0

NOT (-1) = 0
NOT (0) = -1
NOT (NAN) = NAN

(NAN= Not A Number)

OR Operator

Used to perform a bit-wise disjunction on two numbers.

Syntax
result = *number1* **Or** *number2*

The **Or** operator performs a bit-wise comparison of identically positioned bits in two numeric expressions and sets the corresponding bit in result according to the following truth table:

If bit in *expr1* **is**	**And bit in** *expr2* **is**	**The** *result* **is**
0	0	0
0	1	1
1	0	1
1	1	1

Although OR is a bit wise operator, it is often used to test Boolean (True/False) conditions. The CR200 decides if something is true or false on the criteria that 0 is false and any non-zero number is true (Section 4.5). In the CR200, the pre-defined constant, True = -1.The binary representation of –1 has all bits equal 1. Thus any number OR -1 returns -1. Any number AND -1 returns the original number.

The predefined constant True = -1
The predefined constant False = 0

If *number*1 is:	*Number*2 is:	The *result* is:
-1	Any Number	-1
-1	NAN (not a number)	NAN
0	Any Number	*Number* 2
0	NAN	NAN

Expressions are evaluated to a number (Section 4.5) and can be used in place of one or both of the numbers. Comparison expressions evaluate as True (-1) or False (0) For example:

```
If Temp(1) > 50 OR Temp(3) < 20 Then
  X = True
Else
  X = False
EndIf
```

and

```
X = Temp(1) > 50 OR Temp(3) < 20
```

Both have the same effect, X will be set to –1 if Temp(1) is greater than 50 OR Temp(4) is less than 40. X will be set to 0 if both expressions are false.

PeakValley (DestPV, DestChange, Reps, Source, Hysteresis)

PeakValley is used to detect peaks and valleys (local maxima and minima) in a signal. When a new peak or valley is detected, the new peak or valley and the change from the previous peak or valley are stored in variables.

Parameter *& Data Type*	**Enter** **PeakValley Parameters**
DestPV *Variable or array*	Variable or array in which to store the new peak or valley. When a new peak or valley is detected, the value of the peak or valley is loaded in the destination. PeakValley will continue to load the previous peak or valley until the next peak or valley is detected.
DestChange *Variable or array*	Variable or array in which to store the change from the previous peak or valley. When a new peak or valley is detected, the change from the previous peak or valley is loaded in the destination. When a new peak or valley has not yet been reached, 0 is stored in the destination. When Reps are greater than 1, the array must be dimensioned to Reps+1. The additional element is used to flag when a new peak or valley is detected in any of the source inputs. The flag element is stored after the changes [e.g., *changevar*(Reps+1)] and is set to -1 (true) when a new peak or valley is detected and set to 0 (false) when none are detected.
Reps *Constant*	The number inputs to track the peaks and valleys for. Each input is tracked independently. When reps are greater than 1 the source and DestPV arrays must be dimensioned to at least the number of repetitions; DestChange must be dimensioned to Reps+1.
Source *Variable or Array*	The variable or array containing the inputs to check for peaks and valleys.
Hysteresis *Constant Variable or expression*	The minimum amount the input has to change to be considered a new peak or valley. This would usually be entered as a constant.

```
Public PeakV(2), Change(3),Deg
Public Dim XY(2)

Const Pi=4*ATN(1)                  'Define Pi for converting degrees to radians

DataTable(PV1,Change(1),500)       'Peaks and valleys for first signal, triggered when
                                   'Change(1) is not 0.
    Sample(1,PeakV(1),IEEE4)       'DataTable PV1 holds the peaks and valleys for XY(1)
EndTable

DataTable(PV2,Change(2),500)       'Peaks and valleys for second signal, triggered when
                                   'Change(2) is not 0.
    Sample(1,PeakV(2),IEEE4)       'DataTable PV2 holds the peaks and valleys for XY(2)
EndTable

'The Following table is an alternative to using separate tables for each signal.  It stores both
'signals whenever there is a new peak or valley in either signal. The value stored for the signal
'that does not have a new peak will be a repeat of its last peak or valley.
'Normally a program would not have a table storing peaks and valleys for several signals if 'individual tables it
used individual tables for the signals.
```

```
DataTable(PVBoth,Change(3),500)
    Sample(2,PeakV(1),IEEE4)
EndTable

BeginProg
    Scan(500,mSec,0,0)
        Deg=Deg+5
        XY(1)=Cos(Deg*Pi/180)                    'Compute the cosine as input XY(1)
        XY(2)=Sin(Deg*Pi/180)                    'Compute the sine as input XY(2)

        PeakValley(PeakV(1),Change(1),2,XY(1),0.1)   'Find the peaks and valleys for both
                                                     'inputs. Hysteresis = 0.1
        CallTable PV1
        CallTable PV2
        CallTable PVBoth
    Next Scan
EndProg
```

PRT (Dest, Reps, Source, Mult, Offset)

Used to calculate temperature from the resistance of an RTD.

Syntax
PRT(Dest, Reps, Source**)**

Remarks
This instruction uses the result of a previous RTD bridge measurement to calculate the temperature. The input (Source) must be the ratio Rs/R0, where Rs is the RTD resistance and R0 the resistance of the RTD at 0° C.

The temperature is calculated according to the DIN 43760 specification adjusted (1980) to the International Electrotechnical Commission standard. The range of linearization is -200° C to 850° C. The error in the linearization is less than 0.001° C between -200 and +300° C, and is less than 0.003° C between -180 and +830° C. The error (T calculated - T standard) is +0.006° at -200° C and -0.006° at +850° C.

Parameter *& Data Type*	**Enter** **PRT Parameters**
Dest *Variable or Array*	The variable in which to store the temperature in degrees C.
Reps *Constant*	The number of repetitions for the measurement or instruction.
Source *Variable or Array*	The name of the variable or array that contains the Rs/RO value(s).
Mult, Offset *Constant, Variable, Array, or Expression*	A multiplier and offset by which to scale the raw results of the measurement. See the measurement description for the units of the raw result; a multiplier of one and an offset of 0 are necessary to output in the raw units. For example, the **TCDiff** instruction measures a thermocouple and outputs temperature in degrees C. A multiplier of 1.8 and an offset of 32 will convert the temperature to degrees F.

Randomize

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, day of week, and/or day of year from the CR9000 clock.

Syntax
RealTime(Dest**)**

Remarks
RealTime Example

This example uses **RealTime** to place all time segments in the Destination array. If the remark (‘) is removed from the first 8 Sample statements and the last Sample statement is remarked, the results will be exactly the same.

```
Public rTime(9)                     'declare as public and dimension rTime to 9
Alias rTime(1) = Year               'assign the alias Year to rTime(1)
Alias rTime(2) = Month              'assign the alias Month to rTime(2)
Alias rTime(3) = Day                'assign the alias Day to rTime(3)
Alias rTime(4) = Hour               'assign the alias Hour to rTime(4)
Alias rTime(5) = Minute             'assign the alias Minute to rTime(5)
Alias rTime(6) = Second             'assign the alias Second to rTime(6)
Alias rTime(8) = WeekDay            'assign the alias WeekDay to rTime(8)
Alias rTime(9) = Day_of_Year        'assign the alias Day_of_Year to rTime(9)

DataTable (VALUES, 1, 100)          'set up data table
'   Sample(1, Year, IEEE4)          'place Year in VALUES table
'   Sample(1, Month, IEEE4)         'place Month in VALUES table
'   Sample(1, Day, IEEE4)           'place Day in VALUES table
'   Sample(1, Hour, IEEE4)          'place Hour in VALUES table
'   Sample(1, Minute, IEEE4)        'place Minute in VALUES table
'   Sample(1, Second, IEEE4)        'place Second in VALUES table
'   Sample(1, WeekDay, IEEE4)       'place WeekDay in VALUES table
'   Sample(1, Day_of_Year, IEEE4)   'place Day_of_Year in VALUES table
   Sample(9, rTime(), IEEE4)        'place all 9 segments in VALUES table
EndTable

BeginProg
   Scan (1, mSec, 0, 0)
      RealTime(rTime())
      CallTable VALUES
   Next Scan
EndProg
```

RectPolar (Dest, Source)

Converts from rectangular to polar coordinates.

Parameter *& Data Type*	**Enter** **RectPolar Parameters**
Dest *Variable array*	Variable array in which to store the 2 resultant values. The length of the vector is stored in the specified destination element and the angle, in radians(± π), in the next element of the array
Source *Variable Array*	The variable array containing the X and Y coordinates to convert to Polar coordinates. The X value must be in the specified array element and the Y value in the next element of the array.

Example: In the following example, a counter (Deg) is incremented from 0 to 360 degrees. The cosine and sine of the angle are taken to get X and Y in rectangular coordinates. RectPolar is then used to convert to polar coordinates.

```
Dim XY(2),Polar(2),Deg,AnglDeg
Const Pi=4*ATN(1)

Alias XY(1)=X
Alias XY(2)=Y
Alias Polar(1)=Length
Alias Polar(2)=AnglRad

DataTable(RtoP,1,500)
    Sample(1,Deg,IEEE4)
    Sample(2,XY,IEEE4)
    Sample(2,Polar,IEEE4)
    Sample(1,AnglDeg,IEEE4)
EndTable

BeginProg
    For Deg=0 to 360
        XY(1)=Cos(Deg*Pi/180)      'Cos and Sin operate on radians
        XY(2)=Sin(Deg*Pi/180)
        RectPolar(Polar,XY)
        AnglDeg=Polar(2)*180/Pi    'Convert angle to degrees for comparison w/Deg
        CallTable RtoP
    Next Deg
EndProg
```

Spatial RMS

Used to compute the RMS value of an array.

Syntax

RMSSpa(Dest, Swath, Source**)**

Remarks

Spatial RMS, Calculate the root mean square of values in an array.

$$Dest = \sqrt{\frac{\sum_{i=j}^{i=j+swath} \left(X(i)\right)^2}{swath}}$$

Where *X(j)* = Source

Parameter *& Data Type*	**Enter** **RMSSpa Parameters**
Dest *Variable*	The variable in which to store the RMS value.
Swath *Constant*	The number of values of the array to include in the RMS calculation.
Source *Array*	The name of the variable array that is the input for the instruction.

RmsSpa Function Example

The example uses the RmsSpa function to

StrainCalc

Converts the output of a bridge measurement instruction to microstrain.

Syntax

StrainCalc (Dest, Reps, BrConfig, Source, Zero, GF, *v*)

Remarks

Calculates microstrain, με, 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** **StrainCalc Parameters**
Dest	Variable to store strain in.
Reps	Number of strains to calculate, Destination, source, and zero variables must be dimensioned accordingly.
BrConfig	Bridge configuration code for strain gages The bridge configuration code can be entered as a positive or negative number: + code: $V_r = 0.001(Source - Zero)$; bridge configured so its output decreases with increasing strain. - code: $V_r = -0.001(Source - Zero)$; bridge configured so output increases with strain. This is the configuration for a quarter bridge using CSI's 4WFB350 Terminal Input Module (i.e., enter the bridge configuration code as -1 for 1/4 bridge with TIM.)

Code	**Configuration**
1	Quarter bridge strain gauge $\mu\varepsilon = \frac{-4 \cdot 10^6 V_r}{GF(1+2V_r)}$
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 $+\varepsilon$, the other parallel to $-\varepsilon$: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF}$
4	Full bridge strain gage, 2 gages parallel to $+\varepsilon$, the other 2 parallel to $-\varepsilon$: $\mu\varepsilon = \frac{-10^6 V_r}{GF}$
5	Full bridge strain gage, half the bridge has 2 gages parallel to $+\varepsilon$ and $-\varepsilon$: the other half $+\nu\varepsilon$ and $-\nu\varepsilon$: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF(\nu+1)}$
6	Full bridge strain gage, one half $+\varepsilon$ and $-\nu\varepsilon$, the other half $-\nu\varepsilon$ and $+\varepsilon$.: $\mu\varepsilon = \frac{-2 \cdot 10^6 V_r}{GF[(\nu+1) - V_r(\nu-1)]}$

Source	The source variable array for the measurement(s), the input is expected as millivolts out per volt in (the result of the full bridge instruction with a multiplier of 1 and an offset of 0.
Zero	The variable array that holds the unstrained reading(s) in millivolts out per volt in.
GF	Gage Factor. The gage factor can be entered as a constant used for all repetitions or a variable array can be loaded with individual gage factors which are automatically used with each rep. To use an array enter the parameter as *arrayname*(), with no element number in the parentheses.
v	Poisson ratio, enter 0 if it does not apply to configuration.

StrainCalc Example

This example uses StrainCalc to find the microstrain value of a bridge output.

```
'Program name: STRAIN.DLD

Public Count, ZStrain, StMeas, Strain, Flag(8)          ' Declare all variables as public

'Data Table STRAINS samples every measurement when user Sets Flag(1) High

DataTable(STRAINS,Flag(1),-1)
    DataInterval(0,0,0,100)                             'Interval = Scan, 100 lapses
    Sample (1,Strain,Ieee4)
EndTable

'DataTable ZERO_1 stores the "zero" measurements

DataTable(ZERO_1,Count>99,100)                          'Trigger on Count 100
    Average(1,ZStrain,IEEE4,0)
EndTable

'Subroutine to measure Zero, Called on first pass or when user sets Flag(2)low

Sub Zero
    Count = 0                                           'Reset Count
    Scan(10,mSec,0,100)                                 'Scan 100 times
        BrFull(ZStrain,1,mV50,5,1,6,7,1,5000,1,0,0,100,1,0)
        Count = Count + 1                               'Increment Counter used By DataTable
        CallTable ZERO_1                                'Zero_1 outputs on last scan (Count=100)
    Next Scan
    ZStrain = ZERO_1.ZStrain_Avg(1,1)                   'Set ZStrain = averaged value
    Flag(2) = True
End Sub

BeginProg
    Scan(10,mSec,0,0)                                   'Scan 10(mSecs)
        If Not Flag(2) Then Zero
        BrFull(StMeas,1,mV50,5,1,6,7,1,5000,1,0,0,100,1,0)
        StrainCalc(Strain,1,StMeas,ZStrain,-1,2,0)
        CallTable STRAINS                               'Strains outputs only when Flag(1)=True
    Next Scan
EndProg
```

Rnd Function

Returns a random number.

Syntax

Rnd[(*number*)]

Remarks

The argument *number* can be any valid numeric expression.

The **Rnd** function returns a Single value less than 1 but greater than or equal to 0.

The value of *number* determines how **Rnd** generates a random number:

Value of number	Number returned
< 0	The same number every time, as determined by number.
> 0	The next random number in the sequence.
= 0	The number most recently generated.
number omitted	The next random number in the sequence.

The same random-number sequence is generated each time the instruction is encountered because each successive call to the **Rnd** function uses the previous random number as a seed for the next number in the random-number sequence.

To have the program generate a different random-number sequence each time it is run, use the Randomize statement without an argument to initialize the random-number generator before **Rnd** is called.

To produce random integers in a given range, use this formula:

Int((upperbound - lowerbound + 1) * **Rnd** + lowerbound)

Here, upperbound is the highest number in the range, and lowerbound is the lowest number in the range.

Rnd Function Example

The example uses the Rnd function to generate random integer values from 1 to 9. Each time this program is run, Randomize generates a new random-number sequence.

```
Dim Wild1, Wild2                'Declare variables.
Randomize                       'Seed random number generator.
Wild1 = Int(9 * Rnd + 1)        'Generate first random value.
Wild2 = Int(9 * Rnd + 1)        'Generate second random value.
```

Sgn Function

Used to find the sign value of a number.

Syntax
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 Sgn(X) = 0.

If X < 0, then Sgn(X) = -1.

Sgn Function Example

The example uses Sgn to determine the sign of a number.

```
Dim Msg, Number                 'Declare variables.
Number = Volt(1)                'Get user input.
Select Case Sgn(Number)         'Evaluate Number.
    Case 0                      'Zero.
        Msg = 0
    Case 1                      'Positive.
        Msg = 1
    Case -1                     'Negative.
        Msg = -1
End Select
```

Sine Function

Returns the sine of an angle.

Syntax
Sin(*angle***)**

Remarks
The argument *angle* 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. Pi is approximately 3.141593.

Returns the sine of the value in parentheses. The input must be in radians.

Sin Function Example

The example uses Sin to calculate the sine of an angle from a Volt input.

```
Dim Degrees, Pi, Radians, Ans      'Declare variables.
Pi = 4 * Atn(1)                    'Calculate Pi.
Degrees = Volt(1)                  'Get input.
Radians = Degrees * (Pi / 180)     'Convert to radians.
Ans = Sin(Radians)                 ‘The Sine of Degrees.
```

Sqr Function

Returns the square root of a *number*.

Syntax
Sqr(*number***)**

Remarks
The argument *number* can be any valid numeric expression that results in a value greater than or equal to 0.

Returns the square root of the value in parentheses.

Sqr Function Example

The example uses Sqr to calculate the square root of Volt(1) value.

```
Dim Msg, Number      'Declare variables.
Number = Volt(1)     'Get input.
If Number < 0 Then
    Msg = 0          ‘Cannot determine the square root of a negative number.
Else
    Msg = Sqr(Number)
End If
```

Spatial Standard Deviation

Used to find the standard deviation of an array.

Syntax
StdDevSpa(Dest, Swath, Source**)**

Remarks
Spatial standard deviation.

$$Dest = \left(\left(\sum_{i=j}^{i=j+swath} X(i)^2 - \left(\sum_{i=j}^{i=j+swath} X(i) \right)^2 / swath \right) / swath \right)^{\frac{1}{2}}$$

Where *X(j)* = Source

Parameter *& Data Type*	**Enter** **StdDevSpa Parameters**
Dest *Variable or Array*	The variable in which to store the results of the instruction.
Swath *Constant*	The number of values of the array over which to perform the specified operation.
Source *Array*	The name of the variable array that is the input for the instruction.

Tangent Function

Returns the tangent of an angle.

Syntax
Tan(*angle***)**

Remarks
The argument *angle* 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. Pi is approximately 3.141593.

Tan Function Example

The example uses Tan to calculate the tangent of an angle from a Volt(1) input.

```
Dim Degrees, Pi, Radians, Ans     'Declare variables.
Pi = 4 * Atn(1)                   'Calculate Pi.
Degrees = Volt(1)                 'Get user input.
Radians = Degrees * (Pi / 180)    'Convert to radians.
Ans = Tan(Radians)                ‘The Tangent of Degrees.
```

XOR

The XOR function is used to perform a binary logical exclusion on two numbers.

Syntax

result = *number1* XOR *number2*

The XOR operator also performs a bit-wise comparison of identically positioned bits in two numbers (may be variables or the results of expressions) and sets the corresponding bit in result according to the following truth table:

If bit in X is	And bit in Y is	The result is
0	0	0
0	1	1
1	0	1
1	1	0

Derived Math Functions

The following is a list of nonintrinsic mathematical functions that can be derived from the intrinsic math functions provided with CRBasic:

Function	CRBasic equivalent
Secant	Sec = 1 / Cos(X)
Cosecant	Cosec = 1 / Sin(X)
Cotangent	Cotan = 1 / Tan(X)
Inverse Sine	Arcsin = Atn(X / Sqr(-X * X + 1))
Inverse Cosine	Arccos = Atn(-X / Sqr(-X * X + 1)) + 1.5708
Inverse Secant	Arcsec = Atn(X / Sqr(X * X - 1)) + Sgn(Sgn(X) -1) * 1.5708
Inverse Cosecant	Arccosec = Atn(X/Sqr(X * X - 1)) + (Sgn(X) - 1) * 1.5708
Inverse Cotangent	Arccotan = Atn(X) + 1.5708
Hyperbolic Sine	HSin = (Exp(X) - Exp(-X)) / 2
Hyperbolic Cosine	HCos = (Exp(X) + Exp(-X)) / 2
Hyperbolic Tangent	HTan = (Exp(X) - Exp(-X)) / (Exp(X) + Exp(-X))
Hyperbolic Secant	HSec = 2 / (Exp(X) + Exp(-X))
Hyperbolic Cosecant	HCosec = 2 / (Exp(X) - Exp(-X))
Hyperbolic Cotangent	HCotan = (Exp(X) + Exp(-X)) / (Exp(X) - Exp(-X))
Inverse Hyperbolic Sine	HArcsin = Log(X + Sqr(X * X + 1))
Inverse Hyperbolic Cosine	HArccos = Log(X + Sqr(X * X - 1))
Inverse Hyperbolic Tangent	HArctan = Log((1 + X) / (1 - X)) / 2
Inverse Hyperbolic Secant	HArcsec = Log((Sqr(-X * X + 1) + 1) / X)
Inverse Hyperbolic Cosecant	HArccosec = Log((Sgn(X) * Sqr(X * X + 1) +1) / X)
Inverse Hyperbolic Cotangent	HArccotan = Log((X + 1) / (X - 1)) / 2
Logarithm	LogN = Log(X) / Log(N)

Section 9. Program Control Instructions

BeginProg, EndProg

Used to mark the beginning and end of a program.

Syntax

BeginProg

...

...

EndProg

Marks the end of Variable, DataTable declarations and the beginning of the main program.

BeginProg Example

This example uses BeginProg and EndProg to mark the beginning and end of a program.

```
BeginProg
  ...
  ...
EndProg
```

BeginBurstTrigger, EndBurstTrigger

or

BeginBurstTrigger, EndBurstTriggerSequence

Without the burst trigger, all measurements within a burst scan are made and the raw values buffered until the specified number of times through the scan are complete. The count in the **Scan** instruction determines the total number of scans stored in the burst buffer. The measurements start as soon as the scan is executed and stop as soon as the specified number of scans are complete. Processing is delayed until after all measurements are complete.

When **BeginBurstTrigger …** is inserted into a burst scan, the scan continues until a trigger is true. Until the trigger(s) is true, the burst buffer is a ring memory that holds the number of scans specified in the scan instruction.

The trigger condition(s) is specified with either the **EndBurstTrigger** or **EndBurstTriggerSequence** instruction. **EndBurstTriggerSequence** is not in CR9000 operating systems prior to OBJ 3.00

Syntax

BeginBurstTrigger

[*statementblock*]

EndBurstTrigger(BurstTrigger, ScansAfterTrigger**)**

or

BeginBurstTrigger

[statementblock]

EndBurstTriggerSequence (ScansBefore, StartTrigger, StopTrigger, ScansAfterTrigger, FillandStop)

The BeginBurstTrigger... EndBurstTrigger(...) statement has these parts:

Part	Description
BeginBurstTrigger	Marks the start of the instructions that must be processed to test the Burst Trigger.
Statementblock	Typically this would consist of the measurement to be tested for the trigger. While any number of CRBasic statements could be inserted here, the processing associated with any instructions within the statement block must be able to be completed within 400 microseconds or the burst scan (whichever is longer). Keep the number of instructions as small as possible.
EndBurstTrigger *or* **EndBurstTriggerSequence**	Specifies the trigger(s) and marks the end of the instructions to process for testing the trigger.

While other measurements in burst scan are buffered, the processing for the instructions between **BeginBurstTrigger** and **EndBurstTrigger** or **EndBurstTriggerSequence** is completed as the scans are occurring. These instructions must be kept to a minimum to avoid processing overruns.

The maximum rate at which the trigger is evaluated is 400 microseconds, this allows some time to complete the processing. If the scan interval is greater than 400 microseconds, the trigger is evaluated at the scan interval. If the scan interval is less than 400 microseconds, the trigger is evaluated at the first integer multiple of the scan interval that is greater than 400 microseconds. For example, if the scan interval is 300 microseconds, the trigger will be tested every 600 microseconds.

If the measurement(s) within the Burst trigger are to be output, set the optional processing parameter on BeginBurstTrigger to 1. This parameter forces the instructions within the burst trigger to be processed with the rest of the scan. (Because the measurements and instructions within the burst trigger are processed as the scans take place, the processing for these instructions is

normally skipped when the processing for the rest of the scan takes place after the burst is completed. This processing includes converting the raw measurement result and storing it in the destination variable.)

EndBurstTrigger

The EndBurstTrigger instruction is typically used to output a fixed number of scans per burst. (The number of scans equal the count in the scan instruction.) Once the ***BurstTrigger*** is true, the burst will continue for the number of scans entered for ***ScansAfterTrigger***. This allows the program to be written to store data before, after, or surrounding the trigger condition. The number of pre-trigger scans is equal to the count in the Scan(...) instruction minus the ScansAfterTrigger. For example, assume the count on the Scan is 10 and **EndBurstTrigger** specifies 5 scans after the trigger. At the end of the burst the burst buffer will hold: 4 scans before the trigger, the scan made at the same time as the trigger, and 5 scans after the trigger. To store all of these values, the program is written to store all measurements made in the scan. Once the burst is complete, the CR9000 loops through the instructions in the scan the number of times specified in the count, pulling data for measurements from the burst buffer.

EndBurstTriggerSequence

The **EndBurstTriggerSequence** instruction can be used to output a variable number of scans per burst (up to the number entered for count in the scan instruction). When the scan starts, only the **StartTrigger** is tested. Once the **StartTrigger** tests true, only the **StopTrigger** is tested. Once the **StopTrigger** is true, the burst will continue for the number of scans entered for **ScansAfterTrigger**.

The number of scans that are processed after the burst is complete is the lesser of:

The Scan Count parameter *or*

ScansBefore + **StartTrigger** scan (1) + Scans between StartTrigger and StopTrigger + **StopTrigger** Scan (1) + **ScansAfter**

The variability in the number of scans processed is in the number of scans between the Start and Stop triggers.

The **FillandStop** option determines which scans are processed if the number of scans determined by the triggers is greater than the Count parameter.

If FillandStop is **true**, the **first** scans will be processed, starting at the specified number of scans before the start trigger. (The burst actually stops after the StartTrigger when the burst buffer is full.)

If FillandStop is **false** the last scans will be processed. (The burst continues until the ScansAfter are complete.)

Parameter *& Data Type*	**Enter**
Process *empty or 1*	This is an optional parameter on the BeginBurstTrigger instruction. If set to 1 the measurements or calculations within BeginBurstTrigger … EndBurst… will be available to output when the burst is complete. If not present, the variables set by these measurements or calculations will retain the value(s) from the last pass through the scan.
BurstTrigger *Variable, or Expression*	The variable or expression to test for the trigger. When the trigger condition is true, a counter is started which increments with each following scan. When the counter equals ScansAfterTrigger the program will exit the Scan(...) Next Scan loop and process the data in the burst buffer. **Value** / **Result** 0 — Do not trigger ≠0 — Trigger
ScansBefore *Constant*	The ScansBeforeStartTrigger parameter specifies the number of scans that occur prior to the StartTrigger to include in the burst output. If the Start Trigger is detected before this number of scans have been completed, there will be fewer values before the trigger.
StartTrigger *Constant Variable or Expression*	If the StartTrigger condition is true (nonzero), all measurements in the scan are executed until the StopTrigger evaluates as true and the ScansAfterTrigger have been completed. **Value** / **Result** 0 — Do not trigger ≠0 — Trigger
StopTrigger *Constant Variable or Expression*	The variable or expression to test for the trigger. Once the StopTrigger is true (nonzero) measurements in the scan are executed until the ScansAfterStopTrigger have been completed **Value** / **Result** 0 — Do not trigger ≠0 — Trigger
ScansAfter Trigger *Constant*	The number of scans to execute after the BurstTrigger or StopTrigger evaluates true. For the **EndBurstTrigger** instruction, the number of pre-trigger scans is equal to the total scans in the Scan(...) instruction minus the ScansAfterTrigger
FillandStop *Constant*	State of burst buffer, determines which data are saved if the triggered scans exceed the capacity of the burst buffer **Value** / **Result** 0 — Ring Memory: The last data are saved and processed ≠0 — Fill and Stop = true. The first data are saved and processed

Burst Trigger Example

This example program monitors two accelerometers. When the first accelerometer exceeds 5 G's there will be 2000 pre-trigger scans and 8000 post-trigger scans recorded.

```
'\\\\\\\\\\\\\\\\\\\\\\ TIMING CONSTANTS //////////////////////

Const RATE = 100                    'Scan interval number
Const RUNITS = 0                    'Scan interval units (uSecs)

'\\\\\\\\\\\\\\\\\\\\\\ BRIDGE CONSTANTS //////////////////////

'_____________________________ Bridge Block #1 _____________________________
Const BRNG1 = 4                     'Block #1 measurement range (200 mV)
Const BREP1 = 2                     'Block #1 repetitions
Const BEXCIT1 = 5000                'Block #1 excitation mVolts
Const BDLY1 = 0                     'Block #1 Delay time (usecs)
Const BINT1 = 0                     'Block #1 integration time (usecs)
Const BMULT1 = 1                    'Block #1 default multiplier
Const BOSET1 = 0                    'Block #1 default offset
Dim BBlk1(BREP1)                    'Block #1 dimensioned source
Units BBlk1ZeroMv = mVperV          'Block #1 default units (mVperV)
Units BBlk1 = G_Force               'Block #1 default units (G_Force)

'\\\\\\\\\\\\\\\\\ ALIASES & PUBLIC VARIABLES /////////////////

Alias BBlk1(1) = Accel_1            'Assign alias name Accel_1 to BBlk1(1)
Alias BBlk1(2) = Accel_2            'Assign alias name Accel_2 to BBlk1(2)
Public Flag(8)                      'General Purpose Flags

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

'Table #1 - Name,TrigVar,Records
DataTable(ACCEL,True,100000)    'Trigger, 100000 records
        DataInterval(0,0,0,1000)    'Synchronous, 1000 lapses
        '_____________________________ Bridge Blocks _____________________________
        Sample (BREP1,BBlk1(),FP2)      '2 Reps,Source,Res
EndTable                            'End of table ACCEL

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

BeginProg                           'Program begins here

Do      'Loop to allow scan on flag repeat
        Do Until Flag(1): Loop      'Loop Until Flag(1) is pressed
        Scan(RATE,RUNITS,1,10000)       'Scan 100(uSecs), buffer 10,000 records to CPU S-RAM
        '_____________________________ Bridge Blocks _____________________________
        BrFull(BBlk1(),BREP1,BRNG1,5,1,6,7,1,BEXCIT1,0,0,BDLY1,BINT1,BMULT1,BOSET1)
        ‘_____________________________ Trigger Sequence_____________________________
        BeginBurstTrigger           'Begin burst scan
        BrFull(BBlk1(),1,BRNG1,5,1,6,7,1,BEXCIT1,0,0,BDLY1,BINT1,BMULT1,BOSET1)
        ‘ Note that BBlk1(1) is measured again here for evaluation as trigger.
        EndBurstTrigger(BBlk1(1)>=5,8000) 'Stop scan 8000 scans after trigger
        CallTable ACCEL             ‘Store every record in the burst buffer.
        Next Scan           'Loop up for the next scan
        Flag(1) = False     'Set Flag(1) back to false
Loop    'Loop up to allow scan on flag
EndProg             ‘Program ends here
```

Call

The Call statement is used to transfer program control from the main program to a subroutine.

Syntax
Call Name**(**List of Variables**)**

Remarks
Use of the Call keyword when calling a subroutine is optional.

The Call statement has these parts:

Part	Description
Call	Call is an optional keyword used to transfer program control to a subroutine..
name	The Name parameter is the name of the subroutine to call.
List of Variables or Constants	The list may contain variables, constants, or expressions that evaluate to a constant (i.e., do not contain 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.

You are never required to use the **Call** keyword when calling a subroutine. If you use the **Call** keyword to call a procedure that requires *arguments*, the *arguments* list must be enclosed in parentheses. If you omit the **Call** keyword, you also must omit the parentheses around the *arguments* list.

You can pass *arguments* to a procedure by reference or by value. Values of *arguments* passed by reference can be altered by the procedure when the *arguments* are returned; CRBasic supplies the actual address of the *argument*.

Call Statement Example

See Sub description in Section 5.

Call Table

Used to call a data table.

Syntax
CallTable Name

Remarks
Calls DataTable that has been declared above BeginProg. When the DataTable is called, it will process data as programmed and check the output condition.

CallTable Example

This example uses CallTable to transfer program control to 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 CR9000 clock from the values in an array. The most likely use for this is where the CR9000 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 CR9000 clock.

Source *Array*	The source must be a seven element array . *array*(1)..*array*(7) should hold respectively year, month, day, hours, minutes, seconds, and microseconds..

Delay

Used to delay the program.

Syntax
Delay(Delay, Units**)**

Remarks
Delay processing the specified time before continuing.

Parameter *& Data Type*	**Enter**		
Delay *Constant*	The time to delay before continuing with the next measurement.		
Units *Constant*	The units for the delay.		
	Alpha Code	**Numeric Code**	**Units**
	USEC	0	microseconds
	MSEC	1	milliseconds
	SEC	2	seconds
	MIN	3	minutes

Delay Function Example

This example uses Delay to cause the program to pause for 20 milliseconds.

Delay(20, msec**)**

Do

Repeats a block of statements while a condition is true or until a condition becomes true.

Syntax 1
Do [{**While** | **Until**} *condition*]
[*statementblock*]
[**Exit Do**]
[*statementblock*]
Loop

Syntax 2
Do
[*statementblock*]
[**Exit Do**]
[*statementblock*]
Loop [{**While** | **Until**} *condition*]

The **Do...Loop** statement has these parts:

Part	**Description**
Do	Must be the first statement in a **Do...Loop** control structure.
While	Indicates that the loop is executed while *condition* is true.
Until	Indicates that the loop is executed until *condition* is true.

condition	Numeric expression that evaluates true (nonzero) or false (0 or Null).
statementblock	Program lines between the **Do** and **Loop** statements that are repeated while or until *condition* is true.
Exit Do	Only used within a **Do...Loop** control structure to provide an alternate way to exit a **Do...Loop**. Any number of **Exit Do** statements may be placed anywhere in the **Do...Loop**. Often used with the evaluation of some condition (for example, If...Then), **Exit Do** transfers control to the statement immediately following the **Loop**. When **Do...Loop** statements are nested, control is transferred to the **Do...Loop** that is one nested level above the loop in which the **Exit Do** occurs.
Loop	Ends a **Do...Loop**.

Do...Loop Statement Example

The example creates an infinite Do...Loop that can be exited only if Volt(1) is within a range.

```
Dim Reply                                  'Declare variable.
Do
        Reply = Volt(1)
        If Reply > 1 And Reply < 9 Then    'Check range.
        Exit Do                            'Exit Do Loop.
        End If
Loop
```

Alternatively, the same thing can be accomplished by incorporating the range test in the Do...Loop as follows:

```
Dim Reply                                  'Declare variable.
Do
        Reply = Volt(1)
Loop Until Reply > 1 And Reply < 9
```

The next example show the use of Wend.

```
While X > Y                                'Old fashioned way of looping.
   .........
   .........
Wend

Do While X > Y                             'Much better
   .........
   .........
Loop
```

FileManage

Used to manage files from within a running DLD program.

Syntax

FileManage("d:FileName", Attribute **)**

Remarks

FileManage is a function that allows a running DLD program to manipulate files that are stored in the CR9000. "d:FileName" is the device and name of the file that must have previously been stored in the CR9000. The device must be **CPU:** or a possible Pam device such as **P4A:**. **P4A:** means **P**am in slot **4**, card **A**. The quote marks (") are necessary. The attributes are actually a bit field as follows:

Bit	Decimal	Description
bit 0	1	not used
bit 1	2	Run On Power Up
bit 2	4	Run Now
bit 3	8	Delete
bit 4	16	Delete All

FileManage Example

The example uses FileManage to run CPU:TEMPS.DLD when Flag(2) becomes high.

If Flag(2) then **FileManage(** "CPU:TEMPS.DLD" 4 **)** **'4 means Run Now**

FileMark(TableName)

Parameter *& Data Type*	**Enter**
TableName *name*	The name of the data table in which to insert the filemark..

FileMark is used to insert a filemark into a data file. The filemark can be used by the decoding software to indicate that a new file should be started at the mark. This capability to create multiple files only exists in the binary to ASCII converter. To make use of it files must be stored to a PCMCIA card and retrieved from the logger files screen or by removing the card and transferring the file directly to the computer.

FileMark is placed within a conditional statement in order to write the filemark at the desired time.

For ... Next Statement

Repeats a group of instructions a specified number of times.

Syntax

For *counter* = *start* **To** *end* [**Step** *increment*]
    [*statementblock*]
    [**Exit For**]
    [*statementblock*]
Next [*counter* [, *counter*][, ...]]

The **For...Next** statement has these parts:

Part	Description
For	Begins a **For...Next** loop control structure. Must appear before any other part of the structure.
counter	Numeric variable used as the loop counter. The variable cannot be an array element or a record element.
start	Initial value of *counter*.
To	Separates *start* and *end* values.
end	Final value of *counter*.
Step	Indicates that *increment* is explicitly stated.
increment	Amount *counter* is changed each time through the loop. If you do not specify **Step**, *increment* defaults to one.
statementblock	Program lines between **For** and **Next** that are executed the specified number of times.
Exit For	Only used within a **For...Next** control structure to provide an alternate way to exit. Any number of **Exit For** statements may be placed anywhere in the **For...Next** loop. Often used with the evaluation of some condition (for example, If...Then), **Exit For** transfers control to the statement immediately following the **Next**.
Next	Ends a **For...Next** loop. Causes *increment* to be added to *counter*.

The *Step* value controls loop execution as follows:

When *Step* is	Loop executes if
Positive or 0	*counter* <= *end*
Negative	*counter* >= *end*

Once the loop has been entered and all the statements in the loop have executed, *Step* is added to *counter*. At this point, either the statements in the loop execute again (based on the same test that caused the loop to execute in the first place), or the loop is exited and execution continues with the statement following the **Next** statement.

Tip Changing the value of *counter* while inside a loop can make the program more difficult to read and debug.

You can nest **For...Next** loops by placing one **For...Next** loop within another. Give each loop a unique variable name as its *counter*. The following construction is correct:

```
For I = 1 To 10
    For J = 1 To 10
        For K = 1 To 10
            ...
        Next K
    Next J
Next I
```

Note If you omit the variable in a **Next** statement, the value of **Step** increment is added to the variable associated with the most recent **For** statement. If a **Next** statement is encountered before its corresponding **For** statement, an error occurs.

For...Next Statement Example

The example runs one For..Next loop inside another.

```
Dim I, J                                  'Declare variables.
For J = 5 To 1 Step -1                    'Loop 5 times backwards.
    For I = 1 To 12                       'Loop 12 times.
        . . . .                           'Run some code.
    Next I
    . . .       .                         'Run some code.
Next J
. . . .                                   'Run some code.
```

This next example fills odd elements of X up to 40 * Y with odd numbers.

```
For I = 1 To 40 * Y Step 2
  X( I ) = I
Next I
```

If ... Then ... Else Statement

Allows conditional execution, based on the evaluation of an expression.

Syntax 1

If *condition* **Then** *thenpart* [**Else** *elsepart*]

Syntax 2

If *condition1* **Then**
    [*statementblock-1*]
[**ElseIf** *condition2* **Then**
    [*statementblock-2*]]
[**Else**
    [*statementblock-n*]]
End If

Syntax 1 Description

The single-line form is often useful for short, simple conditional tests. Syntax 1 has these parts:

Part	Description
If	Begins the simple **If...Then** control structure.
condition	An expression that evaluates true (nonzero) or false (0 and Null).
Then	Identifies actions to be taken if *condition* is satisfied.
thenpart	Statements or branches performed when *condition* is true.
Else	Identifies actions taken if *condition* is not satisfied. If the **Else** clause is not present, control passes to the next statement in the program.
elsepart	Statements or branches performed when *condition* is false.

The *thenpart* and the *elsepart* fields both have this syntax:

{statements | [GoTo] linenumber | GoTo linelabel }

The *thenpart* and *elsepart* syntax has these parts:

Part	Description
statements	One or more CRBasic statements, separated by colons.
Note	You can have multiple statements with a *condition*, but they must be on the same line and separated by colons, as in the following statement: **If** A > 10 **Then** A = A + 1 : B = B + A : C = C + B

Syntax 2 Description

The block form of **If...Then...Else** provides more structure and flexibility than the single-line form and is usually easier to read, maintain, and debug. Syntax 2 has these parts:

Part	Description
If	Keyword that begins the block **If...Then** decision control structure.
condition1	Same as *condition* used in the single-line form shown above.
Then	Keyword used to identify the actions to be taken if a condition is satisfied.
statementblock-1	One or more CRBasic statements executed if *condition1* is true.
ElseIf	Keyword indicating that alternative conditions must be evaluated if *condition1* is not satisfied.
condition2	Same as *condition* used in the single-line form shown above.
statementblock-2	One or more CRBasic statements executed if *condition2* is true.
Else	Keyword used to identify the actions taken if none of the previous conditions are satisfied.
statementblock-n	One or more CRBasic statements executed if *condition1* and *condition2* are both false.
End If	Keyword that ends the block form of the **If...Then**.

In executing a block If, CRBasic tests *condition1*, the first numeric expression. If the expression is true, the statements following **Then** are executed.

If the first expression is false, CRBasic begins evaluating each **ElseIf** condition in turn. When CRBasic finds a true condition, the statements immediately following the associated **Then** are executed. If none of the **ElseIf** conditions is true, the statements following the **Else** are executed. After executing the statements following **Then** or **Else**, the program continues with the statement following **End If**.

The **Else** and **ElseIf** clauses are both optional. You can have as many **ElseIf** clauses as you like in a block **If**, but none can appear after an **Else** clause. Any of the statement blocks can contain nested block **If** statements.

CRBasic looks at what appears after the **Then** keyword to determine whether or not an **If** statement is a block **If**. If anything other than a comment appears after **Then**, the statement is treated as a single-line If statement.

A block **If** statement must be the first statement on a line. The **Else**, **ElseIf**, and **End If** parts of the statement can have nothing but spaces in front of them. The block **If** must end with an **End If** statement.

For Example

```
If a > 1 And a <= 100 Then

    ...

ElseIf a = 200 Then

    ...

End If
```

Tip Select Case may be more useful when evaluating a single expression that has several possible actions.

If...Then ... Else Statement Example

The example illustrates the various forms of the If...Then...Else syntax.

```
Dim X, Y, Temp( 5 )             'Declare variables.
X = Temp( 1 )
If X < 10 Then
    Y = 1                       '1 digit.
ElseIf X < 100 Then
    Y = 2                       '2 digits.
Else
    Y = 3                       '3 digits.
End If
. . . .                         'Run some code
. . . .                         'Run some code
```

LowPriority

...

...

HighPriority

LowPriority is used to set the following code to background execution. It is used in the declaration section of the program to mark a datatable or subroutine that will be called from the slow sequence. HighPriority is used to mark the end of the LowPriority code.

OutLink (Source, Reps, Link)

Used to synchronize scans and to send data to another CR9000 connected via either the Tlink or the Fiber Optic Link. **WaitLinkTrig** is used in the receiving CR9000 to input the data. When OutLink is executed, it sends the data out on the specified link. Link communication requires an acknowledgment from the receiving CR9000. OutLink will time out in 5 seconds if this acknowledgment is not received (320 seconds if executed in the Slow Sequence).

Parameter *& Data Type*	**Enter**
Source *Variable or Array*	The Variable or array that is the source of the values sent out the link
Reps *Constant*	The number of values to send. When greater than 1, the source must be an array dimensioned to at least the number being sent
Link *Constant*	The link to send the data out on. 1 Tlink 2 Fiber Optic Link

Power Off

Used to turn the CR9000 off until a designated time.

Syntax

PowerOff(StartTime, Interval, Units**)**

Parameter *& Data Type*	**Enter**
Start Time *Array*	The name of a six element array that contains the start time: Year, month, day, hour, minutes, and seconds, respectively.
Interval *Constant*	Enter the time interval on which the CR9000 is to be powered up.
Units *Constant*	The units for the time parameters.

Alpha Code	**Numeric Code**	**Units**
SEC	2	seconds
MIN	3	minutes
HR	4	hours
DAY	5	days

Remarks

This instruction sets a time to power up and then shuts off CR9000 power. Only the clock continues running while the CR9000 is powered down. When the time to power up arrives, the power is restored, the CR9000 reloads its program from Flash memory and begins running.

The interval allows the CR9000 to periodically power up and execute a program. StartTime is a time value. If StartTime is in the future when PowerOff is executed, it is the time the CR9000 will be programmed to power up. If StartTime is in the past when PowerOff is executed, The CR9000 will set the time to power up to the next occurrence of the interval (using StartTime as the start of the first interval)

The units for the interval are days, hours, minutes, or seconds.

Power off can also be used in conjunction with the digital inputs on the 9011 Power Supply Board to set up the CR9000 to power up in response to external trigger, make a series of measurements, and then power off.

When the CR9000 is in this power off state the ON Off switch on the 9011 Power Supply Board is in the on position but the internal relay is open. The power LED is not lit. If the "<0.5 " input is switched to ground or if the ">2" input has a voltage greater than 2 volts applied, the CR9000 will awake, load the program in memory and run. If the "< 0.5" input continues to be held at ground while the CR9000 is powered on and goes through its 2–5 second initialization sequence, the CR9000 will not run the program in memory. This is extremely useful if the program executes the PowerOff instruction immediately or after a short measurement period.

The following example is a good one to play with to become familiar with the PowerOff instruction. The CR9000 "scans" once a second for two minutes. At the end of that time it powers down. It is programmed to wake up on a 4 minute interval. After the first PowerOff, it will wake up every four minutes, count for 2 minutes and turn itself off. You can load this program and use the Power On inputs on the 9011 Module to wake the CR9000 before the interval is up. A program for an actual application would have measurements within the scan.

```
Public Start(6), count          'Declare the start time array and count
    'Start() is initialized to 0 at compile time.  0 time is Midnight the start of 1990
    'count is initialized to 0 at compile time
BeginProg
    Scan(1,SEC,0,120)           'Scan once per second for 2 minutes
        Count=count+1           'Increment counter
    NextScan
    PowerOff(Start,4,min)       'Power off, wake up on 4 minute interval
EndProg
```

Print *list of variables or quoted text*

Print is used as a tool in debugging a program to print text or the value of variables at different points in the program. “Printing” occurs over the active link (Tlink or Fiber Optic) and can be observed from Tools | Diagnostics | Terminal Mode in PC9000.

Reset Table

Used to reset a data table under program control.

Syntax

ResetTable(TableName **)**

Remarks

ResetTable is a function that allows a running DLD program to reset 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
```

RunDLDFile

Used to run one DLD file from another.

Syntax
RunDLDFile("d:FileName", Attribute**)**

Remarks
RunDLDFile is a function that allows a running DLD program to call another DLD file that is stored in the CR9000. "d:FileName" is the device and name of the DLD file that must have previously been stored in the CR9000. The device must be **CPU:** or a possible Pam device such as **P4A:**. **P4A:** means **P**am in slot **4**, card **A**. The quote marks (") are necessary. The attribute parameter is evaluated as a binary number where bits one and two are used to indicate if the program is to become the program that runs on power up and/or if it is to replace the current program and run when the instruction is executed.

Bit	Decimal	Description
bit 0	1	not used
bit 1	2	Run On Power Up
bit 2	4	Run Now

Only bit1 and bit2 are available for this function.

For example,

```
RunDLDFile("CPU:TEMPS.DLD", &B100)
```

means to load TEMPS.DLD from CPU flash memory and run it. Whatever DLD file is currently run on power up would be loaded and run if the CR9000 was powered off and then on again. In the above example, the attribute parameter is entered as a binary number (&B100); it could also be entered in decimal format as 4.

```
RunDLDFile("CPU:TEMPS.DLD", &B110)
```

means to load and run TEMPS.DLD and to make it the file that the CR9000 will run when powered up. The attribute parameter could also be entered as 6.

The example uses RunDLDFile to run CPU:TEMPS.DLD when Flag(2) becomes high.

```
If Flag(2) then RunDLDFile("CPU:TEMPS.DLD" 4)      '4 means Run Now
```

Reset Table

Used to reset a data table under program control.

Syntax

ResetTable(TableName **)**

Remarks

ResetTable is a function that allows a running DLD program to reset a data table. TableName is the name of the table to reset.

ResetTable Example

The example program line uses ResetTable to reset table MAIN when Flag(2) is high.

```
If Flag(2) then ResetTable( MAIN )          'resets table MAIN
```

Scan

Used to establish the program scan rate.

Syntax

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

...

...[Exit Scan]

...

Next Scan

Parameter *& Data Type*	**Enter**
Interval *Constant*	Enter the time interval at which the scan is to be executed. The interval may be in µs, ms, s, or minutes, whichever is selected with the **Units** parameter. The maximum scan interval is one minute. When followed by Wait Digital Trigger, 0 may be entered to ensure there is no delay before looking for the trigger.
Units *Constant*	The units for the time parameters. **Alpha Code** / **Numeric Code** / **Units** USEC / 0 / microseconds MSEC / 1 / milliseconds SEC / 2 / seconds MIN / 3 / minutes
BurstOpt *Constant*	Option to Buffer a Burst of measurements before processing. **Option** / **Result** 0 / Process measurements as they are made 1 / Buffer raw measurements in RAM 2 / Buffer raw measurements on PCMCIA Card
Count *Integer*	The number of times to execute the Scan/NextScan loop. Enter 0 for infinite looping.

Remarks
The measurements, processing, and calls to output tables bracketed by the Scan and NextScan instructions determine the sequence and timing of the datalogging.

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

The Scan instruction has four parameters. The Interval is the interval between scans. Units are the time units for the interval. The BurstOption determines if processing is concurrent with measurements or if raw values are buffered for processing after all measurements are made. At the maximum measurement rate, processing time may exceed the time required for measuring. Count is the number of scans to make before processing buffered values (if burst is selected) and proceeding to the instruction following NextScan. A count of 0 means to continue looping forever (or until ExitScan).

If option = 1 or 2, measurements inside **Scan .. Next Scan** are buffered for number of scans, then number of scans are processed.

Scan Example

This example uses Scan and Next Scan in a simple program.

```
Scan(10, usec, 0, 2000)      'Record 2000 samples
   VoltSE(...)
   CallTable
Next Scan
```

Select Case Statement

Executes one of several statement blocks depending on the value of an expression.

Syntax

Select Case *testexpression*
[**Case** *expressionlist1*
 [*statementblock-1*]]
[**Case** *expressionlist2*
 [*statementblock-2*]]
[**Case Else**
 [*statementblock-n*]]
End Select

The Select Case syntax has these parts:

Part	Description
Select Case	Begins the **Select Case** decision control structure. Must appear before any other part of the **Select Case** structure.
testexpression	Any numeric or string expression. If *testexpression* matches the *expressionlist* associated with a **Case** clause, the *statementblock* following that **Case** clause is executed up to the next **Case** clause, or for the final one, up to the **End Select**. Control then passes to the statement following **End Select**. If *testexpression* matches more than one **Case**

	clause, only the statements following the first match are executed.
Case	Sets apart a group of CRBasic statements to be executed if an expression in *expressionlist* matches *testexpression*.
expressionlist	The *expressionlist* consists of a comma-delimited list of one or more of the following forms. expression expression To expression Is compare-operator expression statementblock Elements *statementblock-1* to *statementblock-n* consist of any number of CRBasic statements on one or more lines.
Case Else	Keyword indicating the *statementblock* to be executed if no match is found between the *testexpression* and an *expressionlist* in any of the other **Case** selections. When there is no **Case Else** statement and no expression listed in the **Case** clauses matches *testexpression*, program execution continues at the statement following **End Select**.
End Select	Ends the **Select Case**. Must appear after all other statements in the **Select Case** control structure.

The argument expression list has these parts:

Part	Description
expression	Any numeric expression.
To	Keyword used to specify a range of values. If you use the To keyword to indicate a range of values, the smaller value must precede To.

Although not required, it is a good idea to have a **Case Else** statement in your **Select Case** block to handle unforeseen *testexpression* values.

You can use multiple expressions or ranges in each **Case** clause. For example, the following line is valid:

```
Case 1 To 4, 7 To 9, 11, 13
```

Select Case statements can be nested. Each **Select Case** statement must have a matching **End Select** statement.

Select Case Statement Example

The example uses Select Case to decide what action to take based on user input.

```
Dim X, Y                        'Declare variables.
If Not X = Y Then               'Are they equal
    If X > Y Then
        Select Case X           'What is X.
        Case 0 To 9             'Must be less than 10.
            . . . .             'Run some code.
            . . . .             'Run some code.
        Case 10 To 99           'Must be less than 100.
            . . . .             'Run some code.
            . . . .             'Run some code.
        Case Else               Must be something else.
            . . . .             'Run some code.
        End Select
    End If
Else
    Select Case Y               'What is Y.
    Case 1, 3, 5, 7, 9          'It's odd.
    .    . . .                  'Run some code.
    Case 0, 2, 4, 6, 8          'It's even.
        . . . .                 'Run some code.
    Case Else                   'Out of range.
        . . . .                 'Run some code.
        . . . .                 'Run some code.
    End Select
End If
. . . .                         'Run some code.
. . . .                         'Run some code.
```

Slow Sequence

Allows slower measurements and low priority processing to take place in background.

Syntax

SlowSequence

Remarks

Ends the main program and begins a low priority program. There must be a Scan … NextScan loop following SlowSequence.

It is possible to have a scan in the SlowSequence for measurements that are needed at a slower rate of the primary scan interval. The CR9000 tags on measurement instructions from the slow sequence scan to the normal scan as time allows. At least one A/D conversion from the slow sequence scan is added to each normal scan (the appropriate settling time occurs before the A/D conversion). Thus, the primary scan interval must be long enough to make the primary scan measurements plus the longest single measurement fragment (settling time + A/D conversion) from the scan in the slow sequence. In the case where the primary scan interval is only long enough to allow one measurement fragment from the slow sequence per primary scan, the minimum time for the slow sequence scan interval is the product of the number of slow sequence measurement segments and the primary scan interval. A consequence of the way a measurement scan in the slow sequence may be parceled into several primary scans is that the measurements in a single "scan" of the slow sequence may be spread over a greater time than if they were in the primary scan. Also, if integration is used in a measurement that is included in

the slowsequence scan, the measurements that go into that integration may not occur sequentially, but may be broken up into multiple integration segments that are separated by the time that the primary scan measurements require. If settling time is used for a measurement whose integration is broken up, that settling time will take place before each integration period. Processing instructions within the slow sequence are executed in the time available after processing in the main program is completed.

SlowSequence Example

The example uses SlowSequence to calibrate the CR9000 every ten seconds.

```
SlowSequence
    Scan(10 Secs, 0, 0)
    Calibrate
    BiasComp
    Next Scan
```

Timer

Used to return the value of a timer.

Syntax

Timer(TimNo, Units, TimOpt**)**

Remarks

Timer is a function that returns the value of a timer. TimOpt is used to start, stop, reset and start, stop and reset, or read without altering the state (running or stopped). Multiple timers, each identified by a different number (TimNo), may be active at the same time.

Syntax

variable = Timer(1,usec,2)

Parameter *& Data Type*	**Enter**
TimNo *Constant, Variable, or Expression*	An integer number for the timer (e.g., 0, 1, 2, . . .) Use low numbers to conserve memory; using TimNo 100 will allocate space for 100 timers even if it is the only timer in the program.
Units *Constant*	The units in which to return the timer value.

Alpha Code	**Numeric Code**	**Units**
USEC	0	microseconds
MSEC	1	milliseconds
SEC	2	seconds
MIN	3	minutes

TimOpt *Constant*

The action on the timer. The timer function returns the value of the timer after the action is performed

Code	**Result**
0	start
1	stop
2	reset and start
3	stop and reset
4	read only

Timer Example
The example uses Timer.

Wait Digital Trigger

Used to wait for a digital trigger before making measurements.

Syntax
WaitDigTrig(PSlot, Mask, Word**)**

Remarks
Measurement sequencer will wait until the digital inputs on the CR9071E Counter - Timer / Digital I/O Module matches the specified word. It should be noted that the CR9000 time stamp is clocked by the scan. Thus, if the scan rate is set at 2 seconds, but the trigger is activated every 4 seconds, the time stamp will still increment only 2 seconds every time the trigger activates the scan (increment value will be off by a factor of 2).

Wait Digital Trigger is used to trigger measurement scans from an external digital input to the CR9071E Counter - Timer / Digital I/O Module. WaitDigTrig is placed in the program following Scan; the task sequencer will delay until the digital trigger occurs. Mask and word are in terms of the binary number represented by the 16 digital I/O channels. Mask is used to determine which digital inputs to read. Word is the digital input pattern to match. When the input matches the "word", the scan takes place.

Parameter *& Data Type*	**Enter**
PSlot *Constant*	The number of the slot in the CR9000 card frame that holds the CR9071E Module.
Mask *Constant*	The Mask allows the read or write to only act on certain ports. The Mask is ANDed with the value obtained from the CR9071E when reading and ANDed with the source before writing.
Word *Constant or Variable*	

Example:

```
Scan (1, msec, 0, 0)
WaitDigTrig(6,&B0000000000000100, &B0000000000000100)
'read only port 3, wait until 3 is high.  mask and word entered as binary numbers.
measurements and processing instructions
next Scan
```

example 2

```
WaitDigTrig(6,4,4)
'same as above: read only port 3, wait until 3 is high.
'mask and word entered as decimal numbers.
measurements and processing instructions
Next Scan
```

WaitLinkTrig (Dest, Reps, Link)

WaitLinkTrig is used to trigger CR9000 scans by a message sent over a communication link from another CR9000 sent using the OutLink instruction. WaitLinkTrig is placed immediately after the Scan instruction and pauses the scan until values are received over the specified link. The Scan interval is entered as a negative number to tell the CR9000 that it is to use an external scan trigger input. It should be noted that the CR9000 time stamp is clocked by the scan. Thus, if the scan rate is set at 2 seconds, but the trigger is activated every 4 seconds, the time stamp will still increment only 2 seconds every time the trigger activates the scan (increment value will be off by a factor of 2).

Parameter *& Data Type*	**Enter**
Dest *Variable or array*	Variable or array in which to store the value(s) received over the link. This instruction does not require that any values be stored. If Reps are set to 0, no values will be loaded into the destination variable.
Reps *Constan*	The number of values to expect over the link and store in the destination variable. Enter 0 and the scan still waits for the link but does not store any values.
Link *Constant*	A code specifying the link to receive the data. **Value** — **Link** 1 — **T-Link,** (Link 1**)** 2 — **Fiber Optic Link,** (Link 2)

Appendix A. Keywords and Predefined Constants

The following is a list of keywords and predefined constants in CRBasic. These keywords are not case sensitive and must not be used as variable names. It is possible to use a keyword as part of a variable name if there are additional letters preceding or following the letters that make up the keyword.

ABS	*function*
Alias	*declaration*
AM25T	*measurement*
AND	*operator*
ATN	*function*
Average	*output processing*
AvgRun	*processing*
AvgSpa	*processing*
Battery	*measurement*
BeginBurstTrigger	*program control*
BeginProg	*program control*
BiasComp	*CSI Calibration*
BrFull	*measurement*
BrFull6W	*measurement*
BrHalf	*measurement*
BrHalf3W	*measurement*
BrHalf4W	*measurement*
Calibrate	*calibration*
Call	*program control*
CallTable	*program control*
CanBus	*measurement*
CardA	*=1, predefined constant*
CardAB	=0, *predefined constant*
CardB	*=2, predefined constant*
Case	*program control*
ClockSet	*program control*
Const	*declaration*
COS	*function*
Covariance	*output processing*
COVSpa	*processing*
CSAT3	*measurement*
CSGN	*function*
Data	*processing*
DataEvent	*output processing*
DataInterval	*output processing*
DataTable	*output processing*
day	*=5, predefined constant*
Delay	*program control*
DIM	*declaration*
Do	*program control*
DSP4	*output control*
Else	*program control*
ElseIf	*program control*
End	*program control*
EndBurstTrigger	*program control*
EndBurstTriggerSequence	*program control*
EndIf	*program control*
EndProg	*program control*
EndSelect	*program control*
EndSub	*program control*
EndTable	*output processing*
EQV	*operator*
Excite	*measurement*
Exit	*program control*
ExitDo	*program control*
ExitFor	*program control*
ExitScan	*program control*
ExitSub	*program control*
EXP	*function*
Expr	*function*
False	*=0, predefined constant*
FFT	*output processing*
FFTFilt	*measurement*
FFTSpa	*processing*
FieldNames	*output processing*
FileManage	*program control*
FileMark	*output*
FillStop	*output*
FIX	*function*
FlashOut	*output processing*
FOR	*program control*
FP2	*=7, predefined constant*
FRAC	*function*
GetRecord	*processing*
HighPriority	*program control*
Histogram	*output processing*
Histogram4D	*output processing*
hr	*=4, predefined constant*
IEEE4	*=24, predefined constant*
If	*program control*
IfTime	*function*
IIF	*program control*
IMP	*operator*
INT	*function*
INT8	*measurement*
IS	*operator*
LevelCrossing	*output processing*
LOG	*function*

Long	*=20, predefined constant*
Loop	*program control*
LowPriority	*program control*
Maximum	*output processing*
MaxSpa	*processing*
MemoryTest	*calibration*
min	*=3, predefined constant*
Minimum	*output processing*
MinSpa	*processing*
MOD	*operator*
ModuleTemp	*measurement*
Move	*processing*
msec	*=1, predefined constant*
mV1000	*=1, predefined constant*
mV200	*=4, predefined constant*
mV200c	=16, *predefined constant*
mV50	*=5, predefined constant*
mV500	*=11, predefined constant*
mV5000	*=0, predefined constant*
mV500c	=23, *predefined constant*
mV50c	=17, *predefined constant*
mVX10500	*=3, predefined constant*
mVX1500	*=2, predefined constant*
Next	*program control*
NextScan	*program control*
NextSubScan	*program control*
NOT	*program control*
OpenInterval	*output*
OR	*operator*
OutLink	*program control*
PamOut	*output processing*
PeakValley	*processing*
PF	*function*
PortSet	*measurement*
PowerOff	*program control*
Print	*program control*
PRT	*processing*
Public	*declaration*
PulseCount	*measurement*
PulseCountReset	*measurement*
PWR	*function*
Rainflow	*output processing*
Randomize	*function*
Read	*processing*
ReadIO	*measurement*
RealTime	*processing*
RectPolar	*processing*
RemoveOffset	*calibration*
ResetTable	*program control*
Restore	*processing*
RMSSpa	*processing*
RND	*function*
RunDldFile	*program control*
Sample	*output processing*
Scan	*program control*
SDMSpeed	*measurement*
SDMTrigger	*measurement*
sec	*=2, predefined constant*
Select	*program control*
SerialInput	*measurement*
SetDac	*calibration*
SGN	*function*
SIN	*function*
SIO4	*measurement*
SlotModules	*CSI testing*
SlowSequence	*program control*
SQR	*function*
StationName	*program control*
StdDev	*output processing*
StdDevSpa	*processing*
StrainCalc	*processing*
Sub	*declaration*
SubScan	*program control*
Table	*=5, predefined constant*
TAN	*function*
TCDiff	*measurement*
TCSe	*measurement*
Then	*program control*
TimerIO	*measurement*
Timer	*program control*
To	*program control*
Totalize	*output processing*
True	*=-1, predefined constant*
TypeB	*=4, predefined constant*
TypeE	*=1, predefined constant*
TypeJ	*=3, predefined constant*
TypeK	*=2, predefined constant*
TypeR	*=5, predefined constant*
TypeS	*=6, predefined constant*
TypeT	*=0, predefined constant*
Units	*declaration*
Until	*program control*
usec	*=0, predefined constant*
V10	*=7, predefined constant*
V2	*=10, predefined constant*
V2c	*=22, predefined constant*
V50	*=6, predefined constant*
VoltDiff	*measurement*
VoltSE	*measurement*
Vx105	*=9, predefined constant*
Vx15	*=8, predefined constant*
WaitDigTrig	*program control*
WaitLinkTrig	*program control*
WatchDogTrap	*CSI testing*
Wend	*program control*
While	*program control*
WorstCase	*output processing*
WriteIO	*measurement*
XOR	*operator*

CR9000 Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

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
sales@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
campbell.scientific@wanadoo.fr

Campbell Scientific Spain, S. L.
Psg. Font 14, local 8
08013 Barcelona
SPAIN
www.campbellsci.es
info@campbellsci.es

Please visit www.campbellsci.com to obtain contact information for your local US or International representative.