Warranty and Repair Policy Statement

General
Seller warrants that its products furnished hereunder will, at the time of delivery, be free from defects in material and workmanship and will conform to Seller's published specifications applicable at the time of sale. Seller's obligation or liability to Buyer for products which do not conform to above stated warranty shall be limited to Seller, at Seller's sole discretion, either repairing the product, replacing the product with a like or similar product, or refunding the purchase price of the nonconforming product, provided that written notice of said nonconformance is received by Seller within the time periods set forth below:

a. For all software products, including licensed programs, ninety (90) days from date of initial delivery to Buyer;
b. For all hardware products (excluding batteries), including complete systems, fifteen (15) months from date of initial delivery to Buyer, subject to the additional conditions of paragraph c) below;
c. In the event that Buyer's returned product is a Discontinued product and is not repairable for any reason, Seller may elect to replace it with like or similar product that is, in Seller's sole judgment, the closest equivalent to the returned product. Seller does not warrant that such replacement product will be an exact functional replacement of the returned product.

Further, all products warranted hereunder for which Seller has received timely notice of nonconformance must be returned FOB Seller's plant no later than thirty (30) days after the expiration of the warranty periods set forth above.

These warranties provided herein shall not apply to any products which Seller determines have been subjected, by Buyer or others, to operating and/or environmental conditions in excess of the limits established in Seller's published specifications or otherwise have been the subject of mishandling, misuse, neglect, improper testing, repair, alteration or damage. THESE WARRANTIES EXTEND TO BUYER ONLY AND NOT TO BUYER'S CUSTOMERS OR USERS OF BUYER'S PRODUCT AND ARE IN LIEU OF ALL OTHER WARRANTIES WHETHER EXPRESS, IMPLIED OR STATUTORY INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SELLER BE LIABLE FOR INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES. Seller's liability for any claim of any kind shall in no case exceed the obligation or liability specified in this Warranty clause.

Technical Assistance and Service
Seller's warranty as herein set forth shall not be enlarged, diminished or affected by, and no obligation or liability shall arise or grow out of, Seller's rendering of technical advice, facilities or service in connection with Buyer's order of the goods furnished hereunder. Products returned for warranty service, but which are found to be fully functional and in conformance with specifications may be subject to a nominal service charge and return freight charges. Periodic recalibration of products, if required, is the responsibility of Buyer and is not provided under this Warranty.

Online Support
Online support is available through technical support links on Intelligent Instrumentation's 24-hour World Wide Web site at http://www.EDASce.com. The site contains information on Intelligent Instrumentation's products, new developments, announcements, application notes, application examples, and other useful information. The site and support areas continue to grow as new products, updates, and features are added.

Email Support
Intelligent Instrumentation's technical support can be reached via email. When sending an email message, be sure to include complete contact information as well as a detailed description of the problem and the products being used to: support@edasce.com
Static Sensitivity

Seller ships all static-susceptible products in anti-static packages. Seller’s Warranty as herein set forth shall not cover warranty repair or replacement for products damaged by static due to Buyer's failure to use proper protective procedures when handling, storing, or installing products.

Trademarks

FactoryView®, DASport™, EDAS®, EDAS® CE, UDAS™, Intelligent Instrumentation® are trade names and/or trademarks of Intelligent Instrumentation®, Inc.

Other products or brand names are trademarks or registered trademarks of their respective companies.

Use of Equipment

Intelligent Instrumentation Inc., assumes no responsibility for any direct, indirect or consequential loss or damages resulting from misuse of the equipment or for improper or inadequate maintenance of the equipment or for any such damage or loss resulting from the use of other equipment, attachments, accessories, and repairs at any time made to or placed upon the equipment or any replacement thereof. Furthermore, Intelligent Instrumentation Inc., makes no representations or warranties, either expressed or implied, in connection with the use of the equipment in the event it is improperly used, repaired or maintained.

FCC Radio Frequency Interference Statement

This equipment generates and uses radio frequency energy, and may cause interference to radio or television reception.

Per FCC rules, Part 15, Subpart J, operation of this equipment is subject to the conditions that no harmful interference is caused and that interference must be accepted that may be caused by other incidental or restricted radiation devices, industrial, scientific or medical equipment, or from any authorized radio user.

The operator of a computing device may be required to stop operating his device upon a finding that the device is causing harmful interference and it is in the public interest to stop operation until the interference problem has been corrected.

The user of this equipment is responsible for any interference to radio or television reception caused by the equipment. It is the responsibility of the user to correct such interference.

Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>10-30-2006</td>
<td>Initial Release</td>
</tr>
<tr>
<td>1.1</td>
<td>8-21-2007</td>
<td>Added EDAS-2005M-2</td>
</tr>
</tbody>
</table>
Table of Contents

Chapter 1: Hardware and Installation ..................................................... 1
  1.1 Modules ....................................................................................... 1
    1.1.1 Base Unit ............................................................................... 1
    1.1.2 Power Supply ......................................................................... 6
    1.1.3 Digital I/O Module .................................................................. 12
    1.1.4 Analog Input Module ................................................................ 17
    1.1.5 Digital Input Module ................................................................ 19
    1.1.6 Digital Output Modules .......................................................... 21
    1.1.7 Analog Output Module ............................................................ 23
    1.1.8 Serial Module .......................................................................... 25
    1.1.9 Relay Output Module .............................................................. 27
    1.1.10 Digital Output (Triac) Module ................................................. 29
    1.1.11 Quadrature Module ............................................................... 31
  1.2 Installation .................................................................................... 31
    1.2.1 DIN rail Mounting .................................................................... 31
    1.2.2 Attaching a module to a DIN Rail .......................................... 31
    1.2.3 Removing Modules from a DIN Rail ....................................... 32

Chapter 2: Utilities ................................................................................. 34
  2.1 Summary of Utilities ...................................................................... 34
    2.1.1 MonitorCE .............................................................................. 35
    2.1.2 LCmdSet ................................................................................ 35
    2.1.3 Remote Manager ..................................................................... 35
    2.1.4 DebugLauncher ...................................................................... 35
    2.1.5 TimeSync ................................................................................ 36
    2.1.6 SerialSocket ........................................................................... 36
    2.1.7 CEFlush ................................................................................ 36
  2.2 MonitorCE ...................................................................................... 37
    2.2.1 Required Tools ....................................................................... 37
    2.2.2 Configuring the Communications Program .......................... 37
    2.2.3 Starting a MonitorCE Program Session ................................ 38
    2.2.4 MonitorCE Program Commands ......................................... 39
  2.3 LCmdSet ......................................................................................... 41
    2.3.1 Running LCmdSet ................................................................... 41
    2.3.2 Making a Telnet Connection using HyperTerminal .............. 41
    2.3.3 Issuing Commands .................................................................. 42
    2.3.4 Set Commands ........................................................................ 42
    2.3.5 Show Commands ..................................................................... 44
    2.3.6 Change Commands .................................................................. 46
    2.3.7 Operational Commands .......................................................... 46
  2.4 Remote Manager ............................................................................ 47
    2.4.1 WebDevice ............................................................................. 47
    2.4.2 Using the Remote Manager ................................................... 48
    2.4.3 Remote Manager Home Page ................................................ 49
    2.4.4 Application Manager ............................................................... 52
    2.4.5 File Manager .......................................................................... 53
    2.4.6 Module Manager ..................................................................... 54
    2.4.7 System Manager ..................................................................... 55
    2.4.8 Security Manager .................................................................... 55
    2.4.9 Update Manager ...................................................................... 58
    2.4.10 Developer’s Guide ................................................................. 58
  2.5 DebugLauncher ............................................................................... 59
    2.5.1 Setting Up DebugLauncher ................................................... 59
    2.5.2 debugworkstations.txt File Format .................................... 59
    2.5.3 Changing the contents of debugworkstations .................. 60
  2.6 TimeSync ....................................................................................... 61
    2.6.1 Running TimeSync .................................................................. 61
    2.6.2 Command Line Arguments ................................................... 61
  2.7 SerialSocket ................................................................................... 63
    2.7.1 Configuring the COM ports ................................................... 63
    2.7.2 Running the SerialSocket Utility .......................................... 63
This page intentionally left blank.
Chapter 1: Hardware and Installation

The EDAS CE is an open-architecture monitoring and control system based on the Windows CE operating system. The EDAS CE features a 32-bit processor with built-in 10/100BaseT Ethernet connectivity and a modular I/O system. This combination makes the EDAS CE suitable for a wide range of monitoring and control systems. The EDAS CE’s open architecture allows the user to develop a wide range of embedded control, machine and process monitoring applications using C/C++ or WebDevice development tools.

The built-in 10/100BaseT network, including a TCP/IP stack, enables applications running on the EDAS CE unit to communicate with other computers and other EDAS CE units on a network. The EDAS CE can operate as a stand-alone system or as part of a plant- or factory-wide monitoring/control system.

1.1 Modules

An EDAS CE system consists of a power module, a base unit and I/O modules as needed to meet the specific needs of the targeted application. All the modules are plug-and-play, requiring no system configuration. Up to 12 I/O modules may be used, allowing for systems of up to 192 I/O points.

1.1.1 Base Unit

Each system requires one EDAS-2000E base (processor) unit. The base unit includes the CPU, Ethernet connectivity, and an RS-232 port. The 32-bit
embedded processor runs the Windows CE 3.0 operating system, providing real-time and multi-tasking capabilities.

![EDAS-2000E Base Unit](image)

**FIGURE 1.2** EDAS-2000E Base Unit

The EDAS-2000E base unit includes the following features:
- 32 Bit embedded processor
- Windows CE Operating System
- 64 MB RAM
- 64 MB CompactFlash (1GB maximum)
- 1 RS-232 serial port
- Optional Non-volatile Battery RAM (128 kB)
- Network standards (TCP/IP, UDP, SNMP, DHCP)
- WebDevice (embedded Web server)

### 1.1.1.1 Application Development

Applications may be developed to execute on the EDAS CE using Microsoft’s eMbedded Visual Tools and the eMbedded Visual C/C++ programming language. Applications developed in C/C++ use the CE Link API for reading and writing I/O points on the optional modules. Network and serial port communications use the standard Windows 32 API. See Chapter 3: Developing Custom Programs.

The EDAS CE includes a program (suprcate.exe) that runs on the EDAS CE as a data server. A PC or other computer can open a connection and issue commands to read and write the EDAS CE I/O points. Applications which communicate with suprcate.exe use the Net Link API. The Net Link API supports Visual C/C++ and Visual Basic on the Windows (98/Me/NT/2000)
Platform and C on Unix platforms. See the Net Link Manual for more information.

### 1.1.1.2 EDAS-2000E Base Unit Specifications

All specifications are typical at 25° C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td></td>
<td>AMD Elan SC400</td>
</tr>
<tr>
<td>Operating System</td>
<td></td>
<td>Windows CE 3.0</td>
</tr>
<tr>
<td>DRAM Memory</td>
<td></td>
<td>64 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>72-pin EDO SIMM 60 ns</td>
</tr>
<tr>
<td>Compact Flash</td>
<td></td>
<td>64 MB (1GB maximum)</td>
</tr>
<tr>
<td>Ethernet</td>
<td></td>
<td>10/100BaseT (RJ-45)</td>
</tr>
<tr>
<td>Serial Port</td>
<td></td>
<td>1 RS-232 (up to 115 k Baud)</td>
</tr>
<tr>
<td>Power consumption</td>
<td>+5 VDC</td>
<td>1.5 A maximum</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 3.35W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 85W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 °C</td>
</tr>
</tbody>
</table>

**TABLE 1.1**  **EDAS-2000E-1 Base Unit Specifications**

### 1.1.1.3 Ethernet

The base unit features a 10/100BaseT port. The unit has 5 red LED’s used to display information about the Ethernet connection.

1. Link: Good link when on.
2. Collision: Collision detected when on.
3. 100BaseT: Indicates 100BaseT connection when on. When off, connection is 10BaseT or no connection exists.
4. TX: Transmitting.
5. RX: Receiving.

### 1.1.1.4 Serial Port

The unit has a single serial port, COM1. The serial port can be used to configure the unit’s network settings and real-time clock. See section 2.2 MonitorCE.

An application may read and write data to this port (COM1) using the standard Win32 API.
1.1.1.5 Real-Time Clock Battery

The base unit contains a 3.0 V lithium battery that powers the Real-Time Clock when external power is absent. Battery life is approximately two years when the unit is not powered. When the unit is on external power, battery life is longer since the battery does not have an appreciable power draw in this condition. To replace the battery, remove the right hand cover of the unit.

![Real-time Clock Battery and DRAM Access](image)

**FIGURE 1.3** Real-time Clock Battery and DRAM Access
1.1.1.6 DRAM

The EDAS-2000E can use up to 64 Mbytes of DRAM. The DRAM is industry standard 5 V, 72-pin EDO or FPM. To change the DRAM module, remove the right hand cover of the unit.

1.1.1.7 Compact Flash

The EDAS-2000E uses standard Compact Flash memory for non-volatile storage of the operating system and drivers. The remaining memory can be used by the applications for non-volatile application or data storage. The CompactFlash is accessed as part of the unit’s file system. The CompactFlash can be replaced with a larger card to providing additional storage capability.

1.1.1.8 Reset/Watchdog Timer

The unit has a watchdog timer that may be activated by an application. When the watchdog timer is activated, an application must "tickle" the timer every 2000 ms or the unit will reboot.

1.1.1.9 NV-RAM Option

The EDAS-2000E is available with an optional 128 kbytes of battery RAM. The NV-RAM provides high speed non-volatile storage. Applications such a fast control loops can use the NV-RAM to hold state variables, allowing an application to recover variables after a power loss.

The NV-RAM is memory mapped to 0xA0000 to 0xBFFFF.
1.1.2 Power Supply

The EDAS-2001E power supply module provides power for the base unit and attached modules. The EDAS-2001E can accept line voltages of 90 - 260 VAC @ 50 - 60 Hz.
1.1.1.10 EDAS-2001E Power Supply Specifications

The EDAS-2001E supplies 5 VDC power to the CPU module which in turn supplies power to the I/O modules. The -2 and -3 have quick a disconnect connector that can be used to pull 5VDC and 12 VDC (-2) or 24 VDC (-3) power from the supply to power other components in your system.

All specifications are typical at 25° C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Voltage</td>
<td></td>
<td>90 - 260 VAC</td>
</tr>
<tr>
<td>Input Frequency</td>
<td></td>
<td>50 - 60 Hz</td>
</tr>
<tr>
<td>Output Voltage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDAS-2001E-1</td>
<td></td>
<td>5.0 V +/- 5%</td>
</tr>
<tr>
<td>EDAS-2001E-2</td>
<td></td>
<td>5.0 V +/- 5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12.0 V +/- 5%</td>
</tr>
<tr>
<td>EDAS-2001E-3</td>
<td></td>
<td>5.0 V +/- 5%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>24.0 V +/- 5%</td>
</tr>
<tr>
<td>Output Current</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDAS-2001E-1</td>
<td>5 V</td>
<td>10.0 A max</td>
</tr>
<tr>
<td>EDAS-2001E-2</td>
<td>5 V</td>
<td>7.0 A max</td>
</tr>
<tr>
<td></td>
<td>12 V</td>
<td>2.5 A max</td>
</tr>
<tr>
<td>EDAS-2001E-3</td>
<td>5 V</td>
<td>7.0 A max</td>
</tr>
<tr>
<td></td>
<td>24 V</td>
<td>2.0 A max</td>
</tr>
<tr>
<td>Agency Approvals</td>
<td></td>
<td>Tested to UL 60950</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 3.35W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 85W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 ºC</td>
</tr>
</tbody>
</table>

TABLE 1.2 EDAS-2001E Power Supply Specifications

1.1.1.11 Power Connector

The EDAS-2001E has a standard IEC power connector. Due to the wide range of power sources and distance from the EDAS system a power cord is not included with this module.
1.1.3 Digital I/O Module

The EDAS-2002M Digital I/O Module has 8 digital inputs and 4 digital outputs. The digital inputs and outputs are designed for 24 VDC operation. All of the digital inputs and outputs provide 500 V channel to channel isolation with one return per channel. LEDs provide visual feedback on the channels’ current states. The digital inputs can be individually configured for normal (high/low), counter or latched operation. The digital outputs can individually configured for normal (open/closed), pulsed, delayed and square wave output.
1.1.1.12 Input Functions

The Digital I/O Module provides the following digital input functions. The inputs can be configured on a channel by channel basis.

- Normal: Reads the current states of the input (low/high).
- Counter: 24 bit up/down counter, 250 Hz maximum count rate.
- Latched: The input is latched on a low-to-high, high-to-low, or any change of state.
o High Speed Counter: Channel 0 only, 16-bit counter at 20 kHz maximum count rate.

1.1.1.13 Output Functions
The Digital I/O Module provides the following digital output functions. The outputs can be configured on a channel by channel basis.

o Normal: Set output to desired state (open/closed)
o Pulsed: Set output active for a specified amount of time.
o Delayed: Set output active after the specified time delay.
o Square wave: Generate a square wave with the specified period, 250 Hz maximum.

1.1.1.14 EDAS-2002M Digital I/O Module Specifications
All specifications are typical at 25° C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Input</td>
<td>8 inputs</td>
<td></td>
</tr>
<tr>
<td>Input Voltage Low</td>
<td>3.0 V max</td>
<td></td>
</tr>
<tr>
<td>Input Current Low</td>
<td>Vin = 0.5 V</td>
<td>&gt; 500 nA</td>
</tr>
<tr>
<td>Input Voltage High</td>
<td>9 V min, 30 V max</td>
<td></td>
</tr>
<tr>
<td>Input Current High</td>
<td>Vin = 24 V</td>
<td>5 mA max</td>
</tr>
<tr>
<td>Low Speed Counter</td>
<td>Ch 0 through 7</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>24 bit</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>250 Hz max</td>
<td></td>
</tr>
<tr>
<td>High Speed Counter</td>
<td>Channel 0 only</td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>16 bit</td>
<td></td>
</tr>
<tr>
<td>Frequency</td>
<td>20 kHz max</td>
<td></td>
</tr>
<tr>
<td>Digital Output</td>
<td>FET output</td>
<td>4 Outputs</td>
</tr>
<tr>
<td>On resistance</td>
<td>0.03 Ω</td>
<td></td>
</tr>
<tr>
<td>On current</td>
<td>0.5 A max</td>
<td></td>
</tr>
<tr>
<td>On Voltage</td>
<td>0.8 V max</td>
<td></td>
</tr>
<tr>
<td>Off Voltage</td>
<td>27 V max</td>
<td></td>
</tr>
<tr>
<td>Isolation</td>
<td>Inputs and Outputs</td>
<td></td>
</tr>
<tr>
<td>Channel - Channel</td>
<td>500 V max</td>
<td></td>
</tr>
<tr>
<td>Channel - Bus</td>
<td>1500 V max</td>
<td></td>
</tr>
<tr>
<td>Current Consumption</td>
<td>5 V</td>
<td>200 mA max</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 °C</td>
</tr>
<tr>
<td>TABLE 1.3</td>
<td>EDAS 2002M-1 Digital I/O Module Specifications</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

Hardware and Installation 11
1.1.4 Analog Input Module

The EDAS-2003M Analog Input Module can read voltage, current and thermocouple inputs. Voltage, current or thermocouple readings can be configured on a channel by channel basis, allowing one analog input module to read a combination of voltage, current or thermocouple inputs.

FIGURE 1.7  EDAS-2003M Analog Input module
The analog input stage consists of a 16 channel multiplexer (MUX), followed by a programmable gain amplifier (PGA), feeding a 12-bit analog to digital converter (ADC). The output of the ADC is isolated and presented to the system bus. The MUX is capable of providing 16 single-ended inputs, 8 differential inputs or a mixed combination to the PGA. The PGA has gains of 1, 10 and 100. The ADC supports 0-10 V and ±10 V ranges.

Current Readings: the module has 16 resistors (500 Ω) that may be switched between the individual input channels and ground allowing the unit to make current measurements in the range 0-20 mA.

Thermocouple readings: the module features a built-in Cold Junction Compensator (CJC), which may be switched in to channel zero. Additionally the module has 100 kΩ input bias return resistors that may be switched in to provide an input bias current path for the PGA

1.1.1.15 Configuring Inputs

Each Analog Input channel can be configured for voltage input, current input, or thermocouple input.

To configure a channel for voltage input:

1: Set the appropriate switches to disable current mode (SW2-1 through SW2-8 and SW3-1 through SW3-8). See the table below.
2: Set the appropriate switches to disable input ground return resistor (SW1-1 through SW1-8).
3: The gain and range are set through software control.

To configure a channel for 0-20 mA current input:

1: Set the appropriate switches to enable current mode (SW2-1 through SW2-8 and SW3-1 through SW3-8). See the table below. This configuration requires the channel to be read as a single-ended input.
2: Set the appropriate switches to disable ground return (SW1-1 through SW1-8).
3: In software set the PGA gain to 1 and the ADC Range to 0-10 V.

To configure an input for thermocouple input:

Note: Thermocouple can only be read on differential input channels 1 through 7. Channel 0 is used to read the CJC value.
1: Enable the differential ground return switches for the appropriate channels (SW1-1 through SW1-8). See table below.
2: Enable the CJC circuit (SW3-9). This will connect the CJC circuit to channel 0 to be read as a single-ended analog input. If SW3-10 is turned on with SW3-9, the CJC may be read in differential mode.

<table>
<thead>
<tr>
<th>Switch Number</th>
<th>Switch ON</th>
<th>Switch OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW1-1</td>
<td>GND return for differential Channel A0</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW1-2</td>
<td>GND return for differential Channel A1</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW1-3</td>
<td>GND return for differential Channel A2</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW1-4</td>
<td>GND return for differential Channel A3</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW1-5</td>
<td>GND return for differential Channel A4</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW1-6</td>
<td>GND return for differential Channel A5</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW1-7</td>
<td>GND return for differential Channel A6</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW1-8</td>
<td>GND return for differential Channel A7</td>
<td>No GND return</td>
</tr>
<tr>
<td>SW2-1</td>
<td>Current mode enable Channel 8</td>
<td>Voltage mode enable Channel 8</td>
</tr>
<tr>
<td>SW2-2</td>
<td>Current mode enable Channel 9</td>
<td>Voltage mode enable Channel 9</td>
</tr>
<tr>
<td>SW2-3</td>
<td>Current mode enable Channel 10</td>
<td>Voltage mode enable Channel 10</td>
</tr>
<tr>
<td>SW2-4</td>
<td>Current mode enable Channel 11</td>
<td>Voltage mode enable Channel 11</td>
</tr>
<tr>
<td>SW2-5</td>
<td>Current mode enable Channel 12</td>
<td>Voltage mode enable Channel 12</td>
</tr>
<tr>
<td>SW2-6</td>
<td>Current mode enable Channel 13</td>
<td>Voltage mode enable Channel 13</td>
</tr>
<tr>
<td>SW2-7</td>
<td>Current mode enable Channel 14</td>
<td>Voltage mode enable Channel 14</td>
</tr>
<tr>
<td>SW2-8</td>
<td>Current mode enable Channel 15</td>
<td>Voltage mode enable Channel 15</td>
</tr>
<tr>
<td>SW3-1*</td>
<td>Current mode enable Channel 0</td>
<td>Voltage mode enable Channel 0</td>
</tr>
<tr>
<td>SW3-2</td>
<td>Current mode enable Channel 1</td>
<td>Voltage mode enable Channel 1</td>
</tr>
<tr>
<td>SW3-3</td>
<td>Current mode enable Channel 2</td>
<td>Voltage mode enable Channel 2</td>
</tr>
<tr>
<td>SW3-4</td>
<td>Current mode enable Channel 3</td>
<td>Voltage mode enable Channel 3</td>
</tr>
<tr>
<td>SW3-5</td>
<td>Current mode enable Channel 4</td>
<td>Voltage mode enable Channel 4</td>
</tr>
<tr>
<td>SW3-6</td>
<td>Current mode enable Channel 5</td>
<td>Voltage mode enable Channel 5</td>
</tr>
<tr>
<td>SW3-7</td>
<td>Current mode enable Channel 6</td>
<td>Voltage mode enable Channel 6</td>
</tr>
<tr>
<td>SW3-8</td>
<td>Current mode enable Channel 7</td>
<td>Voltage mode enable Channel 7</td>
</tr>
<tr>
<td>SW3-9</td>
<td>CJC enable</td>
<td>CJC Disable</td>
</tr>
<tr>
<td>SW3-10</td>
<td>CJC differential mode</td>
<td>CJC single-ended mode</td>
</tr>
</tbody>
</table>

* must be OFF when CJC circuit is enabled  
Ground return resistors are 100 kΩ

**TABLE 1.4** Current and Ground Return Resistor Switch Settings

### 1.1.1.16 Hardware / Software Pacing

The Analog Input Module has provisions for pacing (starting) conversions by software or by one of two hardware start convert sources. These two hardware sources are an internal 24-bit Rate Generator and a digital input (EXT INT). The EXT INT input may also be used as a trigger signal.
to control when conversions are started and stopped. This allows for the support of Stop on Trigger, Stop on delay after Trigger, Start on Trigger and N Conversions after Trigger modes.

### 1.1.1.17 Channel Scanner

The Analog Input Module has a hardware channel scanner that scans channels 0 through N. The scanner increments the channel by one each time a conversion is started and wraps back to channel 0. The value of N is set by software.

### 1.1.1.18 Rate Generator

The Analog Input Module provides a 24-bit rate generator to pace (start) conversions. The base clock for this rate generator is 4 MHz providing a resolution of 250 nsec. This allows the rate generator to be set to an output range of 2 MHz to 0.238 Hz (4.2 s). The Rate Generator output is also available on the RGOUT terminal on the I/O connector.

### 1.1.1.19 EDAS 2003M Analog Input Specifications

All specifications are typical at 25 °C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Channels</td>
<td>Single-ended</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>Differential</td>
<td>8</td>
</tr>
<tr>
<td>Resolution</td>
<td></td>
<td>12 bits (1 part in 4096)</td>
</tr>
<tr>
<td>Voltage Ranges</td>
<td>Gain = 1</td>
<td>±10 V, 0 to 10 V</td>
</tr>
<tr>
<td></td>
<td>Gain = 10</td>
<td>±1 V, 0 to 1 V</td>
</tr>
<tr>
<td></td>
<td>Gain = 100</td>
<td>±0.1 V, 0 to 0.1 V</td>
</tr>
<tr>
<td>Current Ranges</td>
<td>Gain = 1</td>
<td>0 to 20 mA</td>
</tr>
<tr>
<td>Overvoltage Protection</td>
<td>Power on or off</td>
<td>-40 V to +55 V</td>
</tr>
<tr>
<td>Gain Accuracy</td>
<td>Gain = 1</td>
<td>±0.012%</td>
</tr>
<tr>
<td></td>
<td>Gain = 10</td>
<td>±0.08%</td>
</tr>
<tr>
<td></td>
<td>Gain = 100</td>
<td>±0.08%</td>
</tr>
<tr>
<td>Input Offset Voltage</td>
<td>Gain = 1</td>
<td>±1.2 mV</td>
</tr>
<tr>
<td></td>
<td>Gain = 10</td>
<td>±0.12 mV</td>
</tr>
<tr>
<td></td>
<td>Gain = 100</td>
<td>±0.012 mV</td>
</tr>
<tr>
<td>Input Bias Current</td>
<td></td>
<td>500 pA</td>
</tr>
<tr>
<td>Specification</td>
<td>Voltage Mode</td>
<td>Current Mode</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Input Impedance</td>
<td>10 GΩ</td>
<td>487 Ω</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common-Mode Range</td>
<td>±10 V</td>
<td></td>
</tr>
<tr>
<td>Common-Mode Rejection Ratio</td>
<td>Gain = 1</td>
<td>80 dB</td>
</tr>
<tr>
<td></td>
<td>Gain = 10</td>
<td>86 dB</td>
</tr>
<tr>
<td></td>
<td>Gain = 100</td>
<td>92 dB</td>
</tr>
<tr>
<td>Noise</td>
<td>RMS/p-p G=1</td>
<td>0.5 LSB/2 LSB</td>
</tr>
<tr>
<td>Monotonicity</td>
<td>No missing codes</td>
<td>12 bits</td>
</tr>
<tr>
<td>Linearity</td>
<td>±1 LSB</td>
<td></td>
</tr>
<tr>
<td>Rate Generator</td>
<td>Resolution</td>
<td>250 nS</td>
</tr>
<tr>
<td>Isolation</td>
<td>Output Frequency</td>
<td>0.238 Hz to 200 kHz</td>
</tr>
<tr>
<td></td>
<td>Input to Bus</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VAC rms for 60 s</td>
<td>1500 V</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>+5 V</td>
<td>400 mA</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 °C</td>
</tr>
</tbody>
</table>

**TABLE 1.5**   EDAS-2003M Analog Input Module Specifications
1.1.5 Digital Input Module

The EDAS-2004M Digital Input Module provides 16 channels of 5V, 24 V, 120 V or 240 V input. The digital inputs provide 500 V channel-to-channel isolation with one return per channel. The 16 channels are arranged as two 8 bit ports, Port 0 and Port 1. LEDs provide visual feedback on the channels’ present states.

**FIGURE 1.8** EDAS-2004M Digital Input module
### 1.1.1.20 EDAS-2004M Digital Input Module Specifications

All specifications are typical at 25° C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Inputs</td>
<td>Opto Isolators</td>
<td>16 inputs</td>
</tr>
<tr>
<td>EDAS-2004M-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td>0-3 V</td>
</tr>
<tr>
<td>High</td>
<td></td>
<td>9-30 V</td>
</tr>
<tr>
<td>Current</td>
<td>Vin = 24V</td>
<td>2 mA max</td>
</tr>
<tr>
<td>EDAS-2004M-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td>0-20 V</td>
</tr>
<tr>
<td>High</td>
<td></td>
<td>70-130 V</td>
</tr>
<tr>
<td>Current</td>
<td>Vin = 120V</td>
<td>2 mA max</td>
</tr>
<tr>
<td>EDAS-2004M-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td>0-40 V</td>
</tr>
<tr>
<td>High</td>
<td></td>
<td>140-250 V</td>
</tr>
<tr>
<td>Current</td>
<td>Vin = 240V</td>
<td>2 mA max</td>
</tr>
<tr>
<td>EDAS-2004M-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td></td>
<td>0-1 V</td>
</tr>
<tr>
<td>High</td>
<td></td>
<td>2-10 V</td>
</tr>
<tr>
<td>Current</td>
<td>Vin = 5V</td>
<td>3 mA max</td>
</tr>
<tr>
<td>Turn-on time</td>
<td></td>
<td>6 mS max</td>
</tr>
<tr>
<td>Turn-off time</td>
<td></td>
<td>35 mS max</td>
</tr>
<tr>
<td>Isolation</td>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Channel - Channel</td>
<td></td>
<td>500 V max</td>
</tr>
<tr>
<td>Bus - Channel</td>
<td></td>
<td>1500 V max</td>
</tr>
<tr>
<td>Current Consumption</td>
<td>5 V</td>
<td>500 mA max</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 °C</td>
</tr>
</tbody>
</table>

**TABLE 1.6  EDAS-2004M Digital Input Module Specifications**
1.1.6 Digital Output Modules

The EDAS-M Digital Output Modules provide 12 channels of open drain, 24 VDC (EDAS-2005M-1) or 360 VDC (EDAS-2005M-2) digital outputs. The digital outputs have 500 V channel-to-channel isolation with one return per channel. The 12 channels are arranged as two ports. Port 0 has 8 channels and Port 1 has 4 channels. This module supports read back allowing the software to determine the value last commanded for each output. Each channel has a hardware switch that determines the channel’s power-on state. When a switch is in the on position during the power-up reset, the corresponding channel will be turned on.

Figure 1.9 shows the EDAS-2005M Digital Output Modules.
1.1.1.21 EDAS-2005M Digital Output Modules Specifications

All specifications are typical at 25°C unless otherwise noted. Maximum voltage or current are dependent on power dissipation. Please contact support for particular current/voltage operating points.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Output</td>
<td>FET output</td>
<td>12 channels, 27VDC max</td>
<td>12 channels, 360VDC max</td>
</tr>
<tr>
<td></td>
<td>On resistance</td>
<td>0.03 Ω</td>
<td>5 Ω, typ</td>
</tr>
<tr>
<td></td>
<td>On current</td>
<td>0.5 A max</td>
<td>260 mA max</td>
</tr>
<tr>
<td></td>
<td>On Voltage</td>
<td>0.8 VDC max</td>
<td>2 VDC max</td>
</tr>
<tr>
<td></td>
<td>Off Voltage</td>
<td>27 VDC max</td>
<td>360 VDC max</td>
</tr>
<tr>
<td>Isolation</td>
<td>Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Channel - Channel</td>
<td></td>
<td>500 V max</td>
<td>500 V max</td>
</tr>
<tr>
<td>Channel - Bus</td>
<td></td>
<td>1500 V max</td>
<td>1500 V max</td>
</tr>
<tr>
<td>Current Consumption</td>
<td>5 V</td>
<td>160 mA max</td>
<td>160 mA max</td>
</tr>
<tr>
<td></td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 42W</td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 °C</td>
<td>0-60 °C</td>
</tr>
</tbody>
</table>
1.1.7 Analog Output Module

The EDAS-2006M Analog Output Module provides 8 channels of 0 to 10 V analog output. This module uses eight 12-bit digital-to-analog converters (DAC). The module has a single LED that will blink when an output is updated.
### 1.1.1.22 EDAS-2006M Analog Output Module Specifications

All specifications are typical at 25°C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Outputs</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>Resolution</td>
<td></td>
<td>12-bits (2.44 mV)</td>
</tr>
<tr>
<td>Output Range</td>
<td></td>
<td>0-10 V</td>
</tr>
<tr>
<td>Output current</td>
<td></td>
<td>10 mA max</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td>+/- 5.2 LSB (13 mV)</td>
</tr>
<tr>
<td>Isolation</td>
<td>Outputs</td>
<td>1500 V max</td>
</tr>
<tr>
<td>Bus - Channel</td>
<td></td>
<td>5V</td>
</tr>
<tr>
<td>Current Consumption</td>
<td>5V</td>
<td>450 mA max</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 °C</td>
</tr>
</tbody>
</table>

**TABLE 1.8** EDAS-2006M Analog Output Module Specifications
1.1.8 Serial Module

The EDAS-2008M Serial Module provides four RS-232 serial ports. The serial ports are interfaced through 9-pin D-subminiature connectors on the bottom of the unit. The EDAS-2000E base unit can support 6 serial modules for a total of 25 serial ports, one on the base plus 24 on additional serial modules.

![EDAS-2008M Serial Port Module](image)

**FIGURE 1.11** EDAS-2008M Serial Port Module

1.1.1.23 EDAS-2008M Serial Module Specifications

All specifications are typical at 25°C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Ports</td>
<td></td>
<td>4 Ports</td>
</tr>
<tr>
<td>Serial Outputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output Voltage Low</td>
<td>3 kΩ load</td>
<td>-5.0 V max</td>
</tr>
<tr>
<td>Output Voltage High</td>
<td>3 kΩ load</td>
<td>+5.0 V max</td>
</tr>
<tr>
<td>Serial Inputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Impedance</td>
<td>3 kΩ min 7 kΩ max</td>
<td></td>
</tr>
<tr>
<td>Input Threshold Low</td>
<td>0.8 V min</td>
<td></td>
</tr>
<tr>
<td>Isolation</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
</tbody>
</table>
1.1.1.24 COM port mappings

The COM ports on the serial module are mapped as COM2 through COM9 on the first two serial module, SER1 through SER8 on the next two and ASY1 through ASY8 on the last two.

<table>
<thead>
<tr>
<th>Port Mapping</th>
<th>Port</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>COM1</td>
<td>COM1</td>
<td>CPU</td>
</tr>
<tr>
<td>COM2</td>
<td>Port A</td>
<td>1</td>
</tr>
<tr>
<td>COM3</td>
<td>Port B</td>
<td>1</td>
</tr>
<tr>
<td>COM4</td>
<td>Port C</td>
<td>1</td>
</tr>
<tr>
<td>COM5</td>
<td>Port D</td>
<td>1</td>
</tr>
<tr>
<td>COM6</td>
<td>Port A</td>
<td>2</td>
</tr>
<tr>
<td>COM7</td>
<td>Port B</td>
<td>2</td>
</tr>
<tr>
<td>COM8</td>
<td>Port C</td>
<td>2</td>
</tr>
<tr>
<td>COM9</td>
<td>Port D</td>
<td>2</td>
</tr>
<tr>
<td>SER1</td>
<td>Port A</td>
<td>3</td>
</tr>
<tr>
<td>SER2</td>
<td>Port B</td>
<td>3</td>
</tr>
<tr>
<td>SER3</td>
<td>Port C</td>
<td>3</td>
</tr>
<tr>
<td>SER4</td>
<td>Port D</td>
<td>3</td>
</tr>
<tr>
<td>SER5</td>
<td>Port A</td>
<td>4</td>
</tr>
<tr>
<td>SER6</td>
<td>Port B</td>
<td>4</td>
</tr>
<tr>
<td>SER7</td>
<td>Port C</td>
<td>4</td>
</tr>
<tr>
<td>SER8</td>
<td>Port D</td>
<td>4</td>
</tr>
<tr>
<td>ASY1</td>
<td>Port A</td>
<td>5</td>
</tr>
<tr>
<td>ASY2</td>
<td>Port B</td>
<td>5</td>
</tr>
<tr>
<td>ASY3</td>
<td>Port C</td>
<td>5</td>
</tr>
<tr>
<td>ASY4</td>
<td>Port D</td>
<td>5</td>
</tr>
<tr>
<td>ASY5</td>
<td>Port A</td>
<td>6</td>
</tr>
<tr>
<td>ASY6</td>
<td>Port B</td>
<td>6</td>
</tr>
<tr>
<td>ASY7</td>
<td>Port C</td>
<td>6</td>
</tr>
<tr>
<td>ASY8</td>
<td>Port D</td>
<td>6</td>
</tr>
</tbody>
</table>
1.1.9 Relay Output Module

The EDAS-2010M Relay Output Module provides 12 channels of single-pole/single-throw normally open relay contacts capable of switching 2 A at either 250 VAC or 30 VDC. This module has power-up initialization hardware allowing the configuration of the power-up state of each output.

**FIGURE 1.12** EDAS-2010M Relay Output Module
### 1.1.1.25 EDAS-2010M Relay Output Module Specifications

All specifications are typical at 25°C unless otherwise noted.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relay Output</td>
<td>Contact closure</td>
<td>12 outputs</td>
</tr>
<tr>
<td>Closed resistance</td>
<td></td>
<td>0.03 Ω</td>
</tr>
<tr>
<td>Closed current</td>
<td></td>
<td>2.0 A max</td>
</tr>
<tr>
<td>Voltage</td>
<td></td>
<td>250 VAC max, 30 VDC max</td>
</tr>
<tr>
<td>Isolation</td>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Channel - Channel</td>
<td></td>
<td>500 V max</td>
</tr>
<tr>
<td>Bus - Channel</td>
<td></td>
<td>1500 V max</td>
</tr>
<tr>
<td>Current Consumption</td>
<td>5 V</td>
<td>450 mA max</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td>Mm</td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60 °C</td>
</tr>
</tbody>
</table>

**TABLE 1.11**  EDAS-2010M Relay Output Module Specifications
1.1.10 Digital Output (Triac) Module

The EDAS-2011M Digital Output Module provides 12 solid state (triac) outputs capable of switching 2 A at 120 VAC. This module has power-up initialization hardware allowing users to configure the power-up state of each output.

![Digital Output Module Diagram]

**FIGURE 1.13** EDAS-2011M Digital Output (Triac) module
1.1.1.26 EDAS-2011M Digital Output Module (Triac) Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triac Output</td>
<td>Contact closure</td>
<td>12 outputs</td>
</tr>
<tr>
<td>Closed resistance</td>
<td></td>
<td>0.03 Ω</td>
</tr>
<tr>
<td>Switching current</td>
<td>25°C 60°C</td>
<td>2.0 A max 1.0 A max</td>
</tr>
<tr>
<td>Voltage</td>
<td>Outputs</td>
<td>24 VAC to 240 VAC</td>
</tr>
<tr>
<td>Isolation</td>
<td>Channel – Channel</td>
<td>500 V max</td>
</tr>
<tr>
<td></td>
<td>Bus – Channel</td>
<td>1500 V max</td>
</tr>
<tr>
<td>Current Consumption</td>
<td>5 V</td>
<td>200 mA max</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches mm</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td></td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60°C</td>
</tr>
</tbody>
</table>

**TABLE 1.12** EDAS-2011M Digital Output Module Specifications
1.1.11 Quadrature Module

The EDAS-2015M Quadrature Module provides 2 independent quadrature decoder channels which can also be used as high-speed 32-bit counter channels. Each channel also provides two outputs which can be controlled manually or configured to output counter channel state information. Each channel also provides two inputs which can be used for counter channel control or for general-purpose digital input.

![Quadrature Module Diagram]

**FIGURE 1.14** EDAS-2015M Quadrature module
1.1.1.27 EDAS-2015M Quadrature Module Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Condition</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quadrature Inputs/Up-Down</td>
<td>24VDC compatible</td>
<td>2 count control per quadrature channel, 2 general purpose, 0-3V low, 9-30V high</td>
</tr>
<tr>
<td>Counters</td>
<td></td>
<td>2. Maximum input rate is 500kSteps/second, 2MHz for counting</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General-purpose Modes</td>
<td>Separately configured for each input</td>
<td>Latch, preload, enable, general-purpose</td>
</tr>
<tr>
<td>Outputs</td>
<td>24VDC compatible</td>
<td>2 per quadrature channel, 5-30VDC</td>
</tr>
<tr>
<td>Comparator Modes</td>
<td>Separately configured for each output</td>
<td>&lt;, &gt;, &lt;=, &gt;=, =, ≠, general purpose</td>
</tr>
<tr>
<td>Isolation</td>
<td>Channel – Channel</td>
<td>500 V max</td>
</tr>
<tr>
<td></td>
<td>Bus – Channel</td>
<td>1500 V max</td>
</tr>
<tr>
<td>Current Consumption</td>
<td>5 V</td>
<td>200 mA max</td>
</tr>
<tr>
<td>Dimensions</td>
<td>inches</td>
<td>4.55D x 5.9H x 1.74W</td>
</tr>
<tr>
<td></td>
<td>mm</td>
<td>116D x 150H x 42W</td>
</tr>
<tr>
<td>Temperature Range</td>
<td>Operating</td>
<td>0-60°C</td>
</tr>
</tbody>
</table>

**TABLE 1.13** EDAS-2015M Quadrature Module Specifications
1.2 Installation

1.1.12 DIN rail Mounting

The EDAS CE systems was designed to be mounted using 35 mm DIN-rails. 35 mm DIN-rails are available in 7.5 mm and 15 mm heights.

The EDAS CE modules are attached to the DIN-rail by first hooking the bottom of the DIN-rail clip on the DIN-rail and the rotating the EDAS CE module until it snaps on to the DIN-rail.

1.1.13 Attaching a module to a DIN Rail

The EDAS CE modules are installed one at a time on to the DIN rail and then slid together. Install the Power Supply module first, followed by the Base unit, followed by I/O modules.
1.1.14 Removing Modules from a DIN Rail

Using a screwdriver, lift the retaining spring until the module releases from the DIN rail. Note: Some modules have two retaining springs. To release the module from the DIN rail, lift both retaining springs.
Chapter 2: Utilities

2.1 Summary of Utilities

The EDAS CE unit is shipped with a number of pre-installed utilities which are summarized in the table shown below. These utilities simplify the development and deployment of applications on the EDAS CE. A brief description of each utility follows the table shown below.

The table below shows the utility programs and their cross-reference.

<table>
<thead>
<tr>
<th>Table 2.1 Utility Program Cross-reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure network settings</td>
</tr>
<tr>
<td>Upload / download files</td>
</tr>
<tr>
<td>Run an application</td>
</tr>
<tr>
<td>View a list of running applications</td>
</tr>
<tr>
<td>Terminate an application</td>
</tr>
<tr>
<td>Reboot the EDAS CE</td>
</tr>
<tr>
<td>Write an application</td>
</tr>
<tr>
<td>Set the unit’s real-time clock</td>
</tr>
<tr>
<td>Remotely communicate with serial ports</td>
</tr>
<tr>
<td>Save registry settings</td>
</tr>
</tbody>
</table>

* See Chapter 3: Developing Custom Programs for more information.
1.1.15 MonitorCE

The MonitorCE program allows a computer to connect to the EDAS CE in order to set the unit’s network parameters and real-time clock. With a null modem cable connected between COM1 on the EDAS CE and one of the computer’s serial ports, a communications application, such as HyperTerminal, may be used to configure many of the EDAS CE’s operating parameters.

1.1.16 LCmdSet

The LCmdSet application runs on the EDAS CE, allowing a remote client to open a Telnet session over the Ethernet network to the EDAS CE. Through this Telnet session, you can change the EDAS CE’s network configuration, serial port configurations, run programs and reboot the unit. The LCmdSet utility can only be used to change configuration settings after the initial configuration settings are made using the MonitorCE program and a functional network connection has been established.

If the EDAS CE will be running on a network accessible from the Internet, be sure to configure the security settings of LCmdSet.

1.1.17 Remote Manager

The Remote Manager is a Web-based management tool that allows a user to perform management functions on the EDAS CE unit using a Web browser. A user with the correct user name and password can download and upload files, run and terminate programs, reboot the unit, read and write the unit’s I/O points, etc.

1.1.18 DebugLauncher

The DebugLauncher application runs on the EDAS CE unit and enables the EDAS CE to make a connection to Microsoft eMbedded Visual Tools on a development computer. This allows remote downloading and debugging of custom programs.

1.1.19 TimeSync

The Time Synchronization application runs on the EDAS CE and can be configured to synchronize the EDAS CE’s real-time clock with an external time standard. This is useful for having the real-time clock maintain time in synchronization with other devices.
1.1.20 SerialSocket

The SerialSocket application runs on the EDAS CE, providing a bridge between the unit’s Ethernet and serial ports. When the SerialSocket program is running on the EDAS CE, a remote computer can open a TCP/IP socket to the EDAS CE and send and receive data to and from the EDAS CE’s serial ports. This is useful for remotely accessing serial devices connected to the EDAS CE’s COM ports.

1.1.21 CEFlush

The CEFlush program runs on the EDAS CE unit to write the current registry settings to the Compact Flash card. This application is typically used by developers that need to change registry settings.
2.2 MonitorCE

The MonitorCE program is used to configure the unit’s network setting and real-time clock. By connecting a null modem cable between the serial port on the EDAS-2000E and a serial port on a computer, a serial communications program such as HyperTerminal can be used to configure the EDAS CE. The EDAS CE’s network settings must be configured before any network communications can be established. The MonitorCE program automatically executes on the EDAS CE unit at start-up.

When the MonitorCE program is running, it monitors COM1 for commands. When another application running on the EDAS CE attempts to open COM1, the MonitorCE program closes, allowing the new program to take ownership of COM1. If another program closes MonitorCE, restart the EDAS CE unit to start it again.

1.1.22 Required Tools

To configure an EDAS CE unit, you need the following items:

- NULL modem serial cable to connect the EDAS CE to a computer.
- A computer with an RS-232 port and a serial communications program, such as HyperTerminal.

Connect the NULL modem serial cable to the EDAS CE base unit’s COM1 port and connect the other end of the cable to any available serial port on the computer.

1.1.23 Configuring the Communications Program

Run a serial communications program, such as HyperTerminal, on the computer and configure it as follows:

- Baud rate: 9600
- Parity: None
- Data bits: 8
- Stop bits: 1
- No flow control
- Line terminator: CRLF (carriage return/line feed)

Instructions for using HyperTerminal follow in the next section.
2.2.1.1 Using HyperTerminal

This section discusses the HyperTerminal serial communications program. However, other serial communications programs can be used.

To start HyperTerminal:

1. Open HyperTerminal using the Start menu on the computer. Click the Hypertrm.exe icon.
2. In the Connection Description dialog, in the name field, type a session name. Select an icon, and click the OK button.
3. In the Connect To dialog, verify the COMx port to which you connected the null modem cable is selected and click the OK button. The COMx Properties dialog opens.
4. In the Port Settings tabbed dialog, enter the Port Settings provided above, and click the OK button.
5. In the New Connection window, open the File menu and select Properties.
6. In the dialog named for the session that you named, select the Settings tab.
7. Click the ASCII Setup button.
8. In the ASCII Setup dialog, in the ASCII Sending section of the dialog, select the checkboxes for Send line ends with line feeds and Echo typed characters locally.
9. In the ASCII Receiving section of the dialog, select the checkbox for Append line feeds to incoming line ends and click the OK button.
10. In the dialog named for the session that you named, click the OK button.
11. When you close the New Connection window, a pop-up dialog asks if you want to save the new session that you named. Click the Yes, No, or Cancel buttons.

1.1.24 Starting a MonitorCE Program Session

With the serial communications program running on the computer and the EDAS CE powered up, start a MonitorCE program session. Note: From the time power is applied to the EDAS CE it will take approximately 30 seconds before the MonitorCE will be ready to accept a connection.

To start a MonitorCE program session:

1. In the serial communications program window, type the special string, @@ and press the Enter key.
   In response, the EDAS CE sends a message to the hyperterminal communications window that reads: MonitorCE Activated.
2. In the serial communication program window, to display help information, type *h* and press the *Enter* key.
3. The EDAS CE responds with the current command set.

```
<table>
<thead>
<tr>
<th></th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note:</strong> The backspace key is not supported in the MonitorCE program. If you use the backspace key the command or parameters will not be correctly processed. If inadvertently used the backspace, press enter and reissue the command without using the backspace key.</td>
<td></td>
</tr>
</tbody>
</table>
```

### 1.1.25 MonitorCE Program Commands

**TABLE 2.2** MonitorCE Program Commands contains the MonitorCE program command syntax to use when the serial communications program is running.

<table>
<thead>
<tr>
<th>Parameter Description</th>
<th>Command Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note:</strong> nnn.nnn.nnn.nnn represents the new parameter value for the IP address, Subnet mask, Gateway address, or DebugWorkstation address. For WINS and DNS: nnn.nnn.nnn.nnn is new first entry and yyy.yyy.yyy.yyy is new second entry (if required).</td>
<td></td>
</tr>
<tr>
<td><strong>IP Address:</strong></td>
<td>NRI</td>
</tr>
<tr>
<td>Viewing:</td>
<td>Nlnnn.nnn.nnn.nnn</td>
</tr>
<tr>
<td>Changing:</td>
<td>NGnnn.nnn.nnn.nnn</td>
</tr>
<tr>
<td><strong>Gateway Address:</strong></td>
<td>NRG</td>
</tr>
<tr>
<td>Viewing:</td>
<td>NGnnn.nnn.nnn.nnn</td>
</tr>
<tr>
<td>Changing:</td>
<td>NSnnn.nnn.nnn.nnn</td>
</tr>
<tr>
<td><strong>Subnet Mask Address:</strong></td>
<td>NRS</td>
</tr>
<tr>
<td>Viewing:</td>
<td>NSnnn.nnn.nnn.nnn</td>
</tr>
<tr>
<td>Changing:</td>
<td>NSnnn.nnn.nnn.nnn</td>
</tr>
<tr>
<td><strong>WINS Address:</strong></td>
<td>NRW</td>
</tr>
<tr>
<td>Viewing:</td>
<td>NWnnn.nnn.nnn.nnn yyy.yyy.yyy.yyy</td>
</tr>
<tr>
<td>Changing:</td>
<td>NWnnn.nnn.nnn.nnn yyy.yyy.yyy.yyy</td>
</tr>
<tr>
<td><strong>DHCP Enable Flag:</strong></td>
<td>NRP</td>
</tr>
<tr>
<td>Viewing:</td>
<td>NPn where n = 1 to enable, n = 0 to disable</td>
</tr>
<tr>
<td>Changing:</td>
<td>NPn where n = 1 to enable, n = 0 to disable</td>
</tr>
<tr>
<td><strong>DNS Address:</strong></td>
<td>NRD</td>
</tr>
<tr>
<td>Viewing:</td>
<td>NDnnn.nnn.nnn.nnn yyy.yyy.yyy.yyy</td>
</tr>
<tr>
<td>Changing:</td>
<td>NDnnn.nnn.nnn.nnn yyy.yyy.yyy.yyy</td>
</tr>
<tr>
<td><strong>Debugging Operations</strong></td>
<td><strong>IP Address of Host:</strong></td>
</tr>
<tr>
<td>Setting:</td>
<td>DBGnnn.nnn.nnn.nnn</td>
</tr>
<tr>
<td><strong>Ethernet Address:</strong></td>
<td>NRE</td>
</tr>
<tr>
<td>Viewing:</td>
<td>NRE</td>
</tr>
</tbody>
</table>
### Real Time Clock:

<table>
<thead>
<tr>
<th>Setting Month:</th>
<th>TIOx, where x = value from 1 - 12. Example: 3 = March</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting Day of the Month:</td>
<td>TIDx, where x = value from 1 and 31.</td>
</tr>
<tr>
<td>Setting Year:</td>
<td>TiYxxxx, where x = year using four digits.</td>
</tr>
<tr>
<td>Setting Hours:</td>
<td>TIHx, where x = value between 0 and 24.</td>
</tr>
<tr>
<td>Setting Minutes:</td>
<td>TIMx, where x = value between 0 and 60.</td>
</tr>
<tr>
<td>Setting Seconds:</td>
<td>TISx, where x = value between 0 and 60.</td>
</tr>
<tr>
<td>Set Real-time Clock:</td>
<td>TIR</td>
</tr>
<tr>
<td>View Real-time Clock Setting:</td>
<td>TIV</td>
</tr>
<tr>
<td>Daylight savings time:</td>
<td>DSTx, set x to 1 to enable; set x to 0 to disable.</td>
</tr>
<tr>
<td>Get current time zone:</td>
<td>GTZ</td>
</tr>
<tr>
<td>Setting current time zone:</td>
<td>STZxyyy zz, where x = + or -, y = number of minutes the desired time zone is from GMT, z = registry index of desired time zone.</td>
</tr>
</tbody>
</table>

**TABLE 2.2** MonitorCE Program Commands
2.3 LCmdSet

The LCmdSet application on the EDAS CE unit acts as a server for the Telnet protocol. This application facilitates an Ethernet connection between a EDAS CE unit running the LCmdSet utility and a computer running a Telnet client application.

When this program starts, the application begins listening for one TCP connection to the Telnet port. Only one active client at a time may be connected to the unit via this port. When one client disconnects, another client connection can be accepted.

1.1.26 Running LCmdSet

The EDAS CE unit ships with LCmdSet located in ```\Storage Card\LCmdSet.exe```. A shortcut is included in ```\Storage Card\Startup``` to launch LCmdSet at power-up.

1.1.27 Making a Telnet Connection using HyperTerminal

1. On a computer with TCP/IP connectivity to the EDAS CE, open the HyperTerminal folder using the Start menu on the computer. Click the ```Hypertrm.exe``` icon.
2. In the Connection Description dialog, enter a name for the connection, select an icon, and click the **OK** button.
3. In the Connect To dialog, select TCP/IP (Winsock), enter the EDAS CE’s IP address in the Host address field, leave the port number set to 23 and press OK.

1.1.28 Issuing Commands

At the **remote>** prompt you can issue commands to the EDAS CE. To issue a command, simply type the command followed by any command parameters and press **<enter>**.

2.3.1.1 Examples

To view the EDAS CE’s IP address: type

```show ipaddress<enter>```

The unit will respond with:

```IP address: nnn.nnn.nnn.nnn (adapter 0)```

where nnn.nnn.nnn.nnn is the IP address of the EDAS CE unit.

To set the EDAS CE IP address, you must be privileged. To become privileged, you must enter a password. Type
set privileged<enter>

When the EDAS CE responds with
    password>

    type

    system<enter>

You will now be privileged and will be allowed to change the unit’s IP address. To change the IP address, type
    change ipaddress nnn.nnn.nnn.nnn<enter>

where nnn.nnn.nnn.nnn is the new IP address for the unit.
The unit will respond with
    Changed IP address: nnn.nnn.nnn.nnn (adapter 0)

### 1.1.29 Set Commands

<table>
<thead>
<tr>
<th>Set Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET PRIVILEGED</td>
</tr>
<tr>
<td>Instructs the server to allow privileged commands. When the Set Privileged command is received, the client is prompted for the privileged password, which LCmdSet does not echo. If the password matches, the privileged mode is entered. The default password is <code>system</code>. It can be changed using the CHANGE PRIVPASS command.</td>
</tr>
<tr>
<td>SET NOPRIVILEGE</td>
</tr>
<tr>
<td>Instructs the server to return to the unprivileged mode.</td>
</tr>
<tr>
<td>SET ECHO ON</td>
</tr>
<tr>
<td>Overrides the echo setting negotiated between LCmdSet and the client terminal program with respect to echo. Set the echo <code>on</code> or <code>off</code>.</td>
</tr>
</tbody>
</table>

### 1.1.30 Show Commands

Notes for Show commands:

1. The IP address (ipaddr), subnet mask (mask), and gateway address (gateway) are entered and displayed in dotted decimal notation (127.0.0.1)
2. The optional ADAPTER parameter defaults to zero, which corresponds to the built-in LAN9000-compatible adapter.

3. The values displayed for IP addresses, the subnet mask, DHCP, and DNS and WINS lists may not be the operational settings. The settings are stored in the registry and will be applied at the next reboot.

<table>
<thead>
<tr>
<th>Show Command Function Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPADDRESS [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the IP address of the indicated network adapter.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SUBNET MASK [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the subnet mask for the indicated network adapter. The default is adapter 0, which is the built-in Ethernet.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>GATEWAY [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the gateway address for the indicated network adapter.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DHCP [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the status of the DHCP option for the indicated network adapter (ON or OFF).</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>DNSLIST [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the list of DNS addresses configured for the indicated network adapter. Items in the list are separated by semicolons.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>WINSLIST [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the list of WINS addresses configured for the indicated network adapter. Items in the list are separated by semicolons.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ADAPTERNAME [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the name of the network adapter which corresponds to the indicated network adapter number. This might be necessary to identify which network adapter to use when changing an option for an adapter other than the default.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>ASSIGNED [ADAPTER n]</td>
</tr>
<tr>
<td>Displays the IP address, subnet mask, gateway address, DNS server list, WINS server list, and DHCP server assigned to the unit by DHCP on the last restart of the unit. If the unit is configured for statically-assigned addresses, the command fails. The command will fail after changing the DHCP flag during the same session.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>SPEED [PORT portnum]</td>
</tr>
<tr>
<td>Displays the baud rate setting stored in the configuration of the indicated serial port.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>PARITY [PORT portnum]</td>
</tr>
<tr>
<td>Displays the parity setting stored in the configuration of the indicated serial port.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>FLOWCONTROL [PORT portnum]</td>
</tr>
<tr>
<td>Displays the flow control option (handshaking) in the stored configuration for the indicated serial port.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>STOPBITS [PORT portnum]</td>
</tr>
<tr>
<td>Displays the number of stop bits in the stored configuration for the indicated serial port. The default is port 1, COM1.</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CHARSIZE [PORT portnum]</td>
</tr>
<tr>
<td>Displays the character size (number of data bits) in the stored configuration for the indicated serial port.</td>
</tr>
</tbody>
</table>
TABLE 2.4  LCmdSet SHOW Commands

DATETIME
Displays the current local date and time of the unit. The date is displayed in mm/dd/yyyy (month/day/year).

TIMEZONE
Displays the time zone of the unit, including the offset from GMT. It also displays whether the unit’s time zone is presently observing Daylight Savings Time or Standard Time.

PROCESSLIST
Displays a list of the processes running on the unit. Displays the process ID, which can be used to terminate the process using the STOP command, and the exe file of the process.

DEBUGWORKSTATIONS
Displays the list of debugworkstations specified in the debugworkstations.txt file on the unit in the \Storage Card folder. Items in the list are separated by semicolons.

1.1.31 Change Commands

Notes for Change commands:

1. The commands in this section require the client be in the privileged mode. See section 1.1.29.

2. After issuing a change command it takes several seconds for the registry to write to persistent storage, wait for the "remote>" prompt before rebooting the unit.

3. The IP address (ipaddr), subnet mask (mask), and gateway address (gateway) are entered and displayed in dotted decimal notation (127.0.0.1)

4. The optional ADAPTER parameter defaults to zero, which corresponds to the built-in LAN9000-compatible adapter.

5. For any network settings to take effect the unit must be rebooted.

<table>
<thead>
<tr>
<th>Change Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP ADDRESS ipaddr [ADAPTER n]</td>
</tr>
<tr>
<td>Changes the IP address for a network adapter in the unit.</td>
</tr>
<tr>
<td>When this command is performed, the DHCP flag for the adapter is automatically set to OFF.</td>
</tr>
<tr>
<td>SUBNET MASK mask [ADAPTER n]</td>
</tr>
<tr>
<td>Sets the subnet mask for a network adapter in the unit.</td>
</tr>
<tr>
<td>When this command is performed, the DHCP flag for the adapter is automatically set to OFF.</td>
</tr>
<tr>
<td>GATEWAY gateway [ADAPTER n]</td>
</tr>
<tr>
<td>Sets the gateway address for a network adapter in the unit.</td>
</tr>
<tr>
<td>When this command is performed, the DHCP flag for the adapter is automatically set to OFF.</td>
</tr>
<tr>
<td>DHCP ON</td>
</tr>
<tr>
<td>Enables or disables DHCP control over the unit’s network configuration for the specified adapter.</td>
</tr>
</tbody>
</table>
When this option is set to ON, the IP address, subnet mask, and gateway address of the adapter are automatically set to 0.0.0.0.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DNSLIST</strong></td>
<td>Changes the DNS addresses for a network adapter in the unit. The list can contain multiple IP addresses, separated by semicolons.</td>
</tr>
<tr>
<td><strong>WINLIST</strong></td>
<td>Changes the WINS addresses for a network adapter in the unit. The list can contain multiple IP addresses, separated by semicolons.</td>
</tr>
<tr>
<td><strong>SPEED</strong></td>
<td>Sets the baud rate for the indicated serial port. Port 1 corresponds to COM1. The new configuration is effective the next time an application that uses the settings stored in the registry opens the port.</td>
</tr>
<tr>
<td><strong>PARITY</strong></td>
<td>Sets the parity for the indicated serial port. The new configuration is effective the next time an application that uses the settings stored in the registry opens the port.</td>
</tr>
<tr>
<td><strong>FLOW CONTROL</strong></td>
<td>Sets the flow control option (handshaking) for the indicated serial port. The new configuration is effective the next time an application that uses the settings stored in the registry opens the port.</td>
</tr>
<tr>
<td><strong>STOPBITS</strong></td>
<td>Sets the number of stop bits for the specified serial port (the default is 1, for COM1). The value 15 indicates 1.5 stop bits. The new configuration takes effect the next time an application that uses the settings stored in the registry opens the port.</td>
</tr>
<tr>
<td><strong>CHARSIZE</strong></td>
<td>Sets the character size for the specified serial port (the number of data bits). The new configuration takes effect the next time an application that uses the settings stored in the registry opens the port.</td>
</tr>
<tr>
<td><strong>DATETIME</strong></td>
<td>Sets the date and local time of the unit using the format month, day, year, hour, minute, second.</td>
</tr>
<tr>
<td><strong>TIMEZONE</strong></td>
<td>Sets the time zone of the unit. The gmtOffset specifies the offset of the unit’s time zone, in hours, from GMT. The offset may be fractional (such as 7.5), and/or negative (such as -7). If the DST flag is set to 1, the unit's time zone is presumed to be operating in Daylight Savings Time. If the flag is set to 0, Standard Time is assumed. If there are multiple time zones at the offset specified in the command, the server will display a list of matching time zones for selection.</td>
</tr>
<tr>
<td><strong>PRIVPASS</strong></td>
<td>Changes the privileged password for the unit. LCmdSet prompts for the new password and saves it to the registry.</td>
</tr>
</tbody>
</table>
DEBUGWORKSTATIONS wslist
Changes the contents of the debugworkstations.txt file on the Storage Card to include the specified list of workstation IP addresses and TCP/IP port numbers. The format is nnn.nnn.nnn.nnn[:port] where the port is optional and the default is 5000. The list can contain multiple items separated by semicolons.

REGISTRY [REMOVEACTIVE | REMOVEALL | FLUSH]
Used to delete the persistent registry files or to force the registry to be flushed.

### TABLE 2.5  LCmdSet CHANGE Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIT</td>
<td>Instructs LCmdSet to close the connection to the client and return to the idle state, waiting for a new client to connect.</td>
</tr>
<tr>
<td>REBOOT</td>
<td>Instructs the unit to be rebooted.</td>
</tr>
<tr>
<td>RUN program [commandline]</td>
<td>Instructs LCmdSet to execute the indicated program on the unit, passing the specified command line, if any.</td>
</tr>
<tr>
<td>STOP pid</td>
<td>Instructs the LCmdSet to terminate the indicated program on the unit. The pid is retrieved from the list of running processes via the SHOW PROCESSLIST command.</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>Instructs the LCmdSet utility to close.</td>
</tr>
</tbody>
</table>

### 1.1.32 Operational Commands

All of the commands in this section except QUIT require the client be in the privileged mode.

<table>
<thead>
<tr>
<th>Operational Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIT</td>
<td>Instructs LCmdSet to close the connection to the client and return to the idle state, waiting for a new client to connect.</td>
</tr>
<tr>
<td>REBOOT</td>
<td>Instructs the unit to be rebooted.</td>
</tr>
<tr>
<td>RUN program [commandline]</td>
<td>Instructs LCmdSet to execute the indicated program on the unit, passing the specified command line, if any.</td>
</tr>
<tr>
<td>STOP pid</td>
<td>Instructs the LCmdSet to terminate the indicated program on the unit. The pid is retrieved from the list of running processes via the SHOW PROCESSLIST command.</td>
</tr>
<tr>
<td>TERMINATE</td>
<td>Instructs the LCmdSet utility to close.</td>
</tr>
</tbody>
</table>
2.4 Remote Manager

The Remote Manager utility provides basic unit management functionality, including access to the file system, application management, reboot capability, and security settings. The Remote Manager is a Web-based application that uses a series of Web pages as the user interface. To access the Remote Manager, use a browser, such as Internet Explorer, and enter the EDAS CE unit’s IP address in the URL window.

A Remote Manager user can perform the following tasks:

- Assign the EDAS CE unit a name, description, and location.
- Perform file management on the unit: uploading, downloading, copying, and deleting files, creating new folders and deleting existing ones.
- Configure, read and write the analog, digital and serial I/O.
- Manage applications running on the unit, including viewing a list of applications currently executing, launching applications, and terminating applications.
- Reboot the EDAS CE unit.
- Control the security settings, user name and password, for the above capabilities.
- Update content on the unit’s Compact Flash card.
- Access the Developers Guide for information on using WebDevice to extend the embedded Web site to create custom applications.

Note: Most content is password protected. The default user name and password for all protected sections is:

  user name: admin
  password: admin

To change the passwords, refer to section 1.1.40 Security Manager.

1.1.33 WebDevice

WebDevice is an embedded Web server that comes preinstalled on the EDAS CE. In addition to serving standard HTML, GIF and JPG content, WebDevice can execute server side scripts to provide dynamic content. The Remote Manager’s functionality is implemented using WebDevice’s server side scripting capabilities.

In addition to enabling the features and functionality of the Remote Manager, WebDevice can be used to build other Web-based applications that run on the EDAS CE. Applications such as machine monitoring and remote data logging can be implemented using WebDevice’s scripting language. A user can extend the EDAS CE’s embedded Web site by uploading additional HTML pages, images and WebDevice script files. This allows the EDAS CE’s
Web interface to be fully customized to meet the application and user needs.

Information on WebDevice’s server side scripting language can be accessed via Remote Manager. Use the "Developers Guide" link on the Remote Manager home page.

The EDAS CE unit ships with WebDevice located on the Compact Flash card at \Storage Card\WebDevice\Wd_Edasce.exe. A shortcut is included in \Storage Card\Startup folder to cause WebDevice to start at power-up. If you do not want WebDevice to run on power-up, simply remove the shortcut.

If WebDevice is not running, it can be started remotely using LCmdSet. See section 2.3 LCmdSet.

1.1.34 Using the Remote Manager

To use the Remote Manager, run a Web browser on a computer and point the browser to the desired EDAS CE unit by entering the EDAS CE unit’s IP address in the browser’s address (URL) window.

Troubleshooting Errors

If a 404 page not found error occurs:

1. Enter http://IP in the address window, where IP is the EDAS CE unit’s IP address. Some browsers require the leading http:// as part of the URL.

2. Verify network connectivity between the computer and the EDAS CE by pinging the EDAS CE unit.
   a. On a Windows computer, open a Command line window.
   b. Type ping IP where IP is the IP address of the EDAS CE unit.
   If a time-out occurs, this indicates a network problem with the PC, the EDAS CE unit, or the network connecting them.

3. Verify the WebDevice program is running by using LCmdSet. See section 2.3 LCmdSet.

Password Protection

All sections of the Remote Manager are password protected with the exception of the Home page, Help and the Developers Guide. The default user name and password is:

   user name: admin
   password: admin

The user name and password can be changed using the Security Manager. See section 1.1.40 for more information.
Help Links

The Remote Manager includes context-sensitive help. To access help, use the Help link located in the upper right-hand corner of all pages.

Other Management Capabilities

For additional information on using other Remote Manager capabilities not included in this manual, refer to the Remote Manager Help pages.

1.1.35 Remote Manager Home Page

Access the Remote Manager home page by entering the EDAS CE unit’s IP address in the browser’s URL window.

![EDAS CE Remote Manager Home Page](image)

**FIGURE 2.1** EDAS CE Remote Manager Home Page
2.4.1.1 EDAS CE Information

The information section of the EDAS CE home page displays the unit’s:
- IP Address
- Name
- Location
- Description

To edit the name, location, and description information, click any of the links, Name, Location, or Description, to open the information page. Edit the information and click the Update link. To exit without making any changes click the Back link.

![Edit Terminal Information Page](image)

**FIGURE 2.2** Edit Terminal Information Page
2.4.1.2 Management

This section of the home page provides links to the six management sections of the Remote Manager Web site and to the Developers Guide section.

- **Application Manager** - View, terminate and run applications on the EDAS CE.
- **File Manager** - View folder listing, upload files, download files, run programs, etc.
- **Module Manager** - View attached modules; configure, read and write I/O points.
- **System Manager** - Reboot the EDAS CE.
- **Security Manager** - View and edit user names and passwords.
- **Update Manager** - Access and download software updates to the EDAS CE.
- **Developers Guide** - Access to WebDevice scripting documentation and samples.
1.1.36 Application Manager

The Application Manager enables you to view the programs that are currently running on the EDAS CE, terminate a program, run a program and create shortcuts to run programs in the startup folder.

**FIGURE 2.3  Application Manager**
1.1.37 File Manager

The File Manager allows a user to view files, download files, upload files, delete files, run programs and WebDevice script’s, create shortcuts to programs in the startup folder, create new folders, delete folders and copy files from one folder to another on the EDAS CE.

![File Manager](image)

**FIGURE 2.4**  File Manager
1.1.38 Module Manager

The EDAS CE module manager allows the user to view the attached modules, configure the I/O points on a module and read or write the I/O points.

FIGURE 2.5  EDAS CE Module Manager
1.1.39 System Manager

The System Management page allows the user to reboot the EDAS CE.

![Figure 2.6 Remotely Rebooting the EDAS CE](image)

1.1.40 Security Manager

The Security Manager allows a user to modify the access privileges for the EDAS CE through WebDevice. WebDevice uses a folder-based user/password system. A folder can be assigned a single user name and password to restrict access to the contents of the folder. The folder structure of the Remote Manager allows for different user/password access to the different management sections.

The Security Manager page provides a cross reference between the management function and the folder. For example to restrict access to the File Manager, you must assign a user name and password to the folder /file.
To add password protection to a folder, remove protection or edit existing user/password, click on the Edit/Remove link and alter the setting for the desired folder.

**FIGURE 2.7** Security Management main page
The WebDevice Server System Configuration page shown in FIGURE 2.8 WebDevice Server System Configuration page, displays the existing protected folders. Simply use the Edit or Remove link for the desired folder to change its setting. To add password protection to a folder that is not listed, use the Add New Protected Folder link.
1.1.41 Update Manager

The Update Manager allows the user to have the EDAS CE check the instrument.com Web site, or a user-created Web site on a corporate server, for new software updates. New updates may be downloaded by the EDAS CE directly from the instrument.com Web site or the corporate site.

![Update Manager](image)

**FIGURE 2.9** Update Manager

1.1.42 Developer’s Guide

The Developer’s Guide link provide links to the WebDevice help pages and to www.edasce.com Web site for example script samples.

The WebDevice help pages explain the server side scripting language and include a complete function reference.
2.5 DebugLauncher

In order to download and debug applications on the EDAS CE unit using eMbedded Visual Tools (eVT), a connection must be established between the EDAS CE and your development PC. The program on the EDAS CE that provides this connection is DebugLauncher.exe. When the DebugLauncher application runs, it reads the text file \Storage Card\debugworkstations.txt to obtain the IP address(s) of the development PC(s). It then enables communications between the EDAS CE and the development PC. If DebugLauncher does not find a debugworkstaions.txt file or the file is empty, DebugLauncher simply exits. The EDAS CE ships with a shortcut to DebugLauncher.exe included in \Storage Card\Startup folder to launch DebugLauncher at power-up.

1.1.43 Setting Up DebugLauncher

To set up the DebugLauncher:

1. Set the contents of debugworkstations.txt
2. Reboot the EDAS CE.

1.1.44 debugworkstations.txt File Format

ddebugworkstations.txt should contain zero or more IP addresses each on a separate line. Optionally, it can also include the TCP/IP port number on which the development PC is listening. The format of an entry in the debugworkstations.txt file is as follows:

<ip address>[:<tcp port>]

The <ip address> field is a dotted decimal representation of the IP address. 127.0.0.5, for example. The optional <tcp port> is the decimal port number on which the development PC is listening. The default tcp port is 5000. Two example lines are shown below. The first uses the default port address 5000, the second uses port 4500.

127.0.0.5
172.16.3.55:4500

**Note:** On Windows XP machines, port 5000 is typically in use, thus you must use a different port number. We recommend port 4500.
1.1.45 Changing the contents of debugworkstations

The debugworkstations.txt file can be changed using the MonitorCE or LCmdSet applications. See section 2.2 MonitorCE or section 2.3 LCmdSet for instructions.

After making changes to the debugworkstations.txt file, the EDAS CE unit must be rebooted for the changes to take effect.
2.6 TimeSync

The TimeSync program allows synchronization of the EDAS CE unit’s clock with an external time standard. The unit supports clock synchronization through the Internet Standard specification RFC 868, also known as the TCP TIME protocol.

The time standard obtained from the Web site www.cis.ohio-state.edu/htbin/rfc/rfc868.htm, for example, has a one-second accuracy and a useful lifetime to the year 2036.

1.1.46 Running TimeSync

The TimeSync program is located at \Storage Card\TimeSync.exe. The TimeSync program can be launched using the LCmdSet utility or the Remote Manager or via a link in the startup folder.

1.1.47 Command Line Arguments

Use one of the following command line formats.

    option argument

or

    option = argument

The option may be abbreviated as described in the table below. For example,

    e45

or

    e = 45

or

    every = 45

The host may be specified as

    host 198.182.119.56

or

    host = 198.182.119.56

Note: Spaces around the equal sign are ignored.
Command line arguments *once*, *every*, and *at* can not be used together.

<table>
<thead>
<tr>
<th>Command Line Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[t] &lt;time_string&gt;</td>
<td>Configures TimeSync to run every day at a specific time of day. The <em>time_string</em> field uses a 24-hour format (hh:mm:ss), where h = hour, m = minute, and s = second. Cannot be combined with e[very] or o[nce].</td>
</tr>
<tr>
<td>e[very] &lt;interval&gt;m</td>
<td>h</td>
</tr>
<tr>
<td>h[ost] &lt;hostname&gt;</td>
<td>Sets the time-server host. For <em>hostname</em>, use either a valid IP address or host name.</td>
</tr>
<tr>
<td>l[og] &lt;logfile_name&gt;</td>
<td>Each time the system clock is synchronized, TimeSync prints a line in the file <em>logfile_name</em>.</td>
</tr>
<tr>
<td>o[nce]</td>
<td>Synchronizes the system clock once and exits. Cannot be combined with a[t] or e[very].</td>
</tr>
<tr>
<td>p[rotocol] TCP</td>
<td>UDP</td>
</tr>
<tr>
<td>r[etry] &lt;interval&gt;s</td>
<td>m</td>
</tr>
<tr>
<td>t[imeout] &lt;seconds&gt;</td>
<td>Sets the UDP timeout to <em>seconds</em>.</td>
</tr>
</tbody>
</table>

**TABLE 2.7** TimeSync Command Line Arguments
2.7 SerialSocket

The SerialSocket application allows a remote computer to access the devices connected to the serial ports on the EDAS CE unit. When this application is executed, it listens on TCP/IP sockets corresponding to the EDAS CE serial ports. When a remote host connects to one of the sockets, the application opens the corresponding serial port and establishes a connection between the socket and the serial port. Any data sent to the socket is sent out the serial port. Data that is received through the serial port is sent to the host over the socket connection.

The SerialSocket utility creates one thread for each serial port on the EDAS CE unit. Each thread listens on the corresponding TCP/IP socket and handles the connection and opening of the serial port, processing data to and from the port/socket. Only one remote program can read/write a given serial port through SerialSocket. Sockets are numbered beginning at 3000, with the socket that corresponds to each COMx port found at 3000+x. For example, the socket number for COM2 is 3002.

1.1.48 Configuring the COM ports

The configuration settings for the ports are stored in the registry. They can be changed using the LCmdSet utility.

1.1.49 Running the SerialSocket Utility

The SerialSocket utility is located at \Storage Card\SerialSocket.exe. SerialSocket may be launched using LCmdSet or Remote Manager. To have SerialSocket run on power-up, use the Remote Manager to add a shortcut to SerialSocket.exe in the \Storage Card\Startup folder.

1.1.50 Testing SerialSocket with Hyperterminal

You can use Hyperterminal to open a socket connection to SerialSocket and send and receive data through one of the EDAS CE’s serial ports.

1. Use LCmdSet to configure the EDAS CE’s COM port settings to match the serial device connected to the port.
2. On a computer with TCP/IP connectivity to the EDAS CE, run Hypertrm.exe.
3. In the Connection Description dialog, enter a name for the connection, select an icon, and click the OK button.
4. In the Connect To dialog, select TCP/IP (Winsock), enter the EDAS CE’s IP address in the Host address field, set the port number to the
port number for the desired EDAS CE serial port (e.g., 3002 for COM2), and press OK.
2.8 CEFlush

The CEFlush application can be used to force the system registry to be written to non-volatile storage. It can be launched using LCmdSet or the Remote Manager.

1.1.51 Saving Registry Settings

The persistent registry settings on the EDAS CE unit are stored on the CompactFlash. The Windows CE operating system and the current registry settings run from DRAM. If a change is made to the registry while the unit is running, the changes are not persistent through a reboot of the unit unless the DRAM-based registry is saved to the CompactFlash card. Running CEFlush accomplishes this save. Refer to the programming section of this manual for more information on programmatically forcing the registry to be written to non-volatile storage.
This page intentionally left blank.
Chapter 3: Developing Custom Programs

Most applications developed for the EDAS CE are of a client-server nature. The EDAS CE typically has a server program running on it that allows a client program to connect and issue commands such as read, process and return data. Thus most solutions that use the EDAS CE consist of two programs, a client and a server. One of these runs on the EDAS CE and the other runs on PC, another EDAS CE or some other Ethernet connected device.

This chapter covers the development of C/C++ applications that are downloaded to and execute on the EDAS CE unit (the servers in the above discussion). Application development is done using Microsoft eMbedded Visual Tools 3.0 (eVT) and the eMbedded Visual C++ programming language. In addition to the eMbedded Visual Tools, you must have the EDAS CE SDK. The API for interfacing to the EDAS CE's I/O modules is called CE Link and is covered in Chapter 4: CE Link API for EDAS CE. Networking, serial and file I/O programming are handled through the Win32 API.

If the application calls for the EDAS CE to be the server and it only needs to provide the client with remote I/O, then you may not need to write a server program for the EDAS CE. The EDAS CE units ship with a program pre-loaded to allow remote computers to connect to and read/write the EDAS CE's I/O. The server program on the EDAS CE that provides this functionality is SuprCatE.exe. To use the EDAS CE as a remote data server, a developer need only write a client program. The API for developing client applications to connect to SuprCatE.exe is Net Link. The Net Link API is not covered in this manual. If you are interested in developing Net Link based client applications to interface to the SuprCatE.exe server, see the Net Link manual. The Net Link manual can be downloaded from the Web site at: www.edasce.com.

3.1 Setting up your Development Computer

Setting up the development environment is a four-step process:

1. Install Microsoft eMbedded Visual Tools 3.0 (eVT).
2. Install the EDAS CE Software Development Kit (SDK).
3. Configure the EDAS CE to communicate with eVT.
4. Configure eVT to communicate with the EDAS CE.
1.1.52 System Requirements

To install eMbedded Visual Tools, the development computer must be running one of the following operating systems.

- Windows 98 Second Edition (SE)
- Windows Me
- Windows NT 4.0 Service Pack 5 (SP5)
- Windows 2000
- Windows XP

1.1.53 Installing eMbedded Visual Tools

Insert Disk 1 of the eMbedded Visual Tools set into the CD ROM drive and follow the installation instructions.

If the installation program does not automatically start, open Windows Explorer and navigate to the CD-ROM drive icon, then right-click on the drive icon. In the pop-up menu, select AutoPlay.

We recommend using the default settings for installation.

1.1.54 Installing the EDAS CE SDK

Install the EDAS CE SDK (Software Development Kit) after completing the installation of the Microsoft eMbedded Visual Tools.

The EDAS CE SDK CD contains the folder sdk_vc, containing the SDK to use with eMbedded Visual C++.

Optionally, the SDK can be downloaded from our Web site at: www.edasce.com.

3.1.1.1 Installing the SDK Program Files

To install the EDAS CE SDK program files:

1. Insert the EDAS CE SDK CD in the development computer’s CD ROM drive.
2. Use Windows Explorer to view the folders included on the CD. Double-click on the sdk_vc folder.
3. Double-click the entry EDAS_CE10.exe to begin the installation.
4. Read the Software License Agreement and accept the license agreement to continue the installation.
5. In the Installation Directory dialog shown below, in the Specify the directory for the installation field, type the path to the directory where you want to install the EDAS CE SDK. We recommend using the default. Click the OK button to continue.
6. The installation will then continue, unpacking and installing the SDK program files.

### 3.1.1.2 Uninstalling the EDAS CE SDK

Follow the directions below to uninstall the EDAS CE SDK.

1. In \Start\Settings\Control Panel double-click the Add/Remove icon. In the Install/Uninstall tabbed dialog, select and delete the EDAS CE SDK.
2. In Windows Explorer, locate the directory containing the folder for the EDAS CE SDK you just uninstalled and manually delete the folder.

### 1.1.55 Setting up Communications to the EDAS CE

Before eVT can download or allow debugging of programs, you need to configure the EDAS CE and eMbedded Visual Tools to communicate.

### 3.1.1.3 Configuring the EDAS CE Unit

For the EDAS CE to communicate with the development PC, you must place the IP address of the development PC in the `debugworkstations.txt` file located in the `\Storage Card` folder on the EDAS CE unit and then reboot the EDAS CE. For more information, see section 2.5 DebugLauncher. The MonitorCE or LCmdSet programs can be used to set the IP address in the `debugworkstations.txt` file. See section 2.2 MonitorCE or section 2.3 LCmdSet.
3.1.1.4 Configuring eMbedded Visual Tools

To set up eMbedded Visual Tools to communicate with the EDAS CE, start eMbedded Visual C++ and follow the directions below:

1. In the Tools menu, select the Configure Platform Manager sub-menu item as shown below.

FIGURE 3.2 Configuring Platform Manager
2. In the Windows CE Platform Manager Configuration dialog, select EDAS_CE10 (Default Device) and click the Properties button.

![Platform Manager Configuration dialog](image)

**FIGURE 3.2** Platform Manager Configuration dialog

3. In the Device Properties dialog, select TCP/IP Transport for Windows CE and click the Configure button.

![Device Properties](image)

**FIGURE 3.2** Configuring TCP/IP Transport
4. In the TCP/IP Transport Configuration dialog shown below, check the Fixed Port checkbox. In the Port Number field, enter a value of 5000 for the port number, or use the port number set in the debugworkstations file on the EDAS CE unit. Click the OK button.

**Note:** On Windows XP machines port 5000 is typically is use, thus you must use a different port number. We recommend port 4500.

**FIGURE 3.2** TCP/IP Transport Configuration dialog
5. In the Device Properties dialog, click the Advanced button. In the next Device Properties dialog, select Manual Server and click the OK button.
6. In the Device Properties dialog, click the Test button.

![Device Properties Dialog]

7. In the Manual Server - Action dialog, click the OK button.

![Manual Server - Action dialog]
8. The development computer attempts to establish a connection with the EDAS CE unit as shown below.

![FIGURE 3.2 Establishing a Platform Manager Connection]

9. As soon as the connection to the device is established, the text in the dialog displays a confirmation message as shown below. Click the OK button to close the dialog.

![FIGURE 3.2 Confirming the Device Connection]
3.1.1.5 Troubleshooting a connection

If the eVT is does not establish communications with the EDAS CE, then use the following list to troubleshoot the problem:

1. Ping the EDAS CE from the PC to confirm that you have a working Ethernet connection. If the ping fails:
   A: Confirm that you have a good Ethernet connection by checking the link light on the EDAS CE, the PC and the hubs/switches between the EDAS CE and PC.
   B: Check the IP, subnet mask and gateway address in both the EDAS CE and the PC. When any of the network setting are changed the EDAS CE must be rebooted before the changes will take effect.

   If the ping is successful, check the debugworkstations IP address and port in the EDAS CE. When the debugworkstations settings are changed the EDAS CE must be rebooted before the changes will take effect.

2. Close eVC, reboot the EDAS CE and restart eVC.

   When a connection is lost between the eVT and the EDAS CE sometimes the only way to re-establish a connection is by closing eVC, resetting the EDAS CE.

3.2 Writing Custom Programs

After installing eMbedded Visual Tools and the EDAS CE SDK, and having established communications between the EDAS CE and eMbedded Visual Tools, you are ready to begin developing custom programs.

1.1.56 Generating an EDAS CE Application

The eMbedded Visual C++ environment includes several wizards that build the basic empty code modules for a variety of applications, DLLs, and so on. This is the recommended procedure for beginning a new programming project for the EDAS CE, as it simplifies the task.

To set up a new project for the EDAS CE, start eMbedded Visual C++ and follow the steps below:

1. In the File menu, select New.
2. In the New dialog, select the Projects tab and select the WCE Application.
3. In the Project Name field, enter a name for your project.
4. In the Location field, select the folder for the project files.
5. In the CPUs field, select the check box for the Win32 (WCE x86) processor.
6. Click the OK button.

**FIGURE 3.1** Using eVC Wizard to generate a new program
7. In the WCE Application - Step 1 of 1 dialog, select A simple Windows CE application, and click the Finish button.

![WCE Application dialog]

**FIGURE 3.2** WCE Application dialog

8. The New Project Information dialog displays the settings that you made in the previous dialogs. Click the OK button to continue.

![New Project Information dialog]

**FIGURE 3.2** New Project Information dialog
9. Once the project is created, eMbedded Visual Tools opens the project and displays the ClassView, ResourceView, and FileView tabbed dialogs in the project window.

10. From the Project menus, select Settings...

![FIGURE 3.2 Project Settings](image)

11. In the Projects Settings dialog, change the Settings For drop down to All Configurations.

12. Select the Link tab.
13. In the Object/library Modules field, remove `commctrl.lib` and add `edasapiE.lib`.

![FIGURE 3.2 Project Setting (Link tab) dialog](image)

14. Next select the Debug tab.

15. In the Download directory field, enter the folder on the EDAS CE where you want eVT to download this application.

The default Download directory is "\", the root folder of the EDAS CE’s file system. The root folder on the EDAS CE is part of the DRAM file system and is therefore volatile. Thus programs downloaded to root location are not retained through a power cycle. To have eVT download your application to the non-volatile file system (Compact Flash) change the Download directory to "\Storage card\" as shown below.
16. Press the OK button.

17. Select the FileView tabbed dialog.
18. Expand the Source Files tree and double clicking on your main .cpp program.

19. In the Select Active Configuration drop-down list, select \texttt{Win32 (WCE x86) Debug}.
20. In the **Select Default Device** drop-down list, select *EDAS_CE10 (Default Device)*.

![FIGURE 3.2 eMbedded Visual C++ Sample Program](image)

21. In the eMbedded Visual Tools window, select the **Build** menu item and select **Rebuild All**.

![FIGURE 3.2 Rebuilding the Program](image)

22. After a successful build, eVT will attempt to download the application. The **Connecting to the device** dialog will be displayed.
If eVT has not yet established a connection to the EDAS CE, the *Manual Server - Action* dialog will open, simply press the OK button.

![Connection Confirmation Message](image1)

**FIGURE 3.2** Connection Confirmation Message

23. The first time an application is executed on the EDAS CE you will get the *Find Local Module* dialog as shown below.

24. Uncheck Try to locate other DLLs and click Cancel.

![Find Local Module](image2)

**FIGURE 3.2** Find Local Module dialog

### 1.1.57 Sample Programs

A number of sample programs are located on Intelligent Instrumentation’s Web site at www.edasce.com. Intelligent Instrumentation grants you permission to copy and modify the sample programs for your EDAS CE application.
3.3 Loading Custom Programs

There are two methods that you can use to transfer programs and other files from the development computer to the EDAS CE unit.

1. Using remote file viewer in Microsoft eMbedded Visual Tools.
2. Using Remote Manager and a browser, see section 1.1.37

1.1.58 Remote File Viewer

You can use the Tools menu in eMbedded Visual Tools to transfer files using the Remote File Viewer.

To transfer files to the EDAS CE unit with the eMbedded Visual Tools:

2. In the Connection menu item, select the Add Connection sub-menu item. The Select a WindowsCE Device dialog opens.
4. In the Manual Server-action dialog, click the OK button. The Connecting to device dialog opens and attempts to connect to a device.

The Remote File Viewer will display the directory structure and files found on the EDAS CE unit. To transfer files to or from the EDAS CE, highlight the target file to transfer from the EDAS CE and choose Import File, or choose the Export File item and you will be prompted for the PC file to be transferred to the EDAS CE.
Chapter 4: CE Link API for EDAS CE

The CE Link API provides an interface between programs running on the EDAS CE and the EDAS CE’s I/O modules. The API includes function calls for initializing, configuring and reading/writing the I/O.

4.1 Overview

This section provides an overview of the CE Link API and how to use it in your application. Section 4.2 provides descriptions and usage for the individual API functions.

1.1.59 Interfacing to the I/O system

For a program on the EDAS CE to interface with the I/O modules it must implement the following sequence:

1. Initialize the hardware and software.
   Initialization is done with a single call to the function nsHWInit(). This function must be called before any other function calls in the CE Link API. See 4.2.1.1 Initialization.

2. Open a session to the I/O system.
   The CE Link API uses a session based methodology for accessing the I/O system. This allows for multiple applications to access the I/O without conflict. The function nsOpenSession(), is used to obtain a session handle. See 4.2.1.2 Opening an I/O Session.

3. Configure the I/O channels for the desired operation.
   To configure the individual I/O channels a program must create a structure to hold the configuration information, populate the structure with the desired settings and then call the appropriate configuration call for the I/O type.

4. Read/write the I/O as needed.
   To read inputs:
   A: Create and populate a structure that defines what modules and inputs to read.
   B: Create a structure with pointers to read data.
   C: Call the appropriate read function.
   D: Access the read data.
   To write outputs:
   A: Create and populate a structure that defines what modules and outputs to write and the data to be written.
B: Call the appropriate write function.

5. Free any dynamically allocated memory.

The read function calls dynamically allocate memory to hold the data read from the inputs. Once your application has made a read call and is done with the data, it must call the i3free() function to release the memory. Failure to do so will create a memory leak in your application. See 4.2.1.38 Memory De-allocation.

Typically steps 4 and 5 are implemented in a loop along with appropriate code to implement a control or monitoring application.

6. Close the session.

Once the application is done interfacing with the I/O, call the nsCloseSession() to close the session. See 4.2.1.3 Closing the I/O Session.

7. Deinitialize the software.

The final step is to deinitialize the CE Link API software using nsSWDeinit(). See 4.2.1.4 Deinitialization.

1.1.60 Function Calls

Most of the CE Link functions have the form:

   error = nsFunction (caller, input-structure-pointer, output-structure-pointer).

The caller parameter is the session handle. The function nsOpenSession() is used to obtain the caller value used for subsequent calls to CE Link functions.

The input-structure-pointer is a pointer to a structure containing all parameters to be passed to the function. All fields in the input-structure must be set by the application before making the function call.

The output-structure-pointer is a pointer to a structure that contains information and pointers to data that are set by the function.

The input-structure-pointer and output-structure-pointer may be specified as NULL (0) if no input or output structure is required.

All the CE Link functions return an error code. When an API function executes successfully, it returns a value of zero. A return value greater than zero indicates an error. A return value less than zero is a warning. Warnings indicate that the API function executed, but that there may be problems you should be aware of. An explanation of the error and warning return values can be found in the list of error codes.
Most debugging of non-functional CE Link applications consists of tracking down where an error is occurring. To simplify this process, check the return values from all CE Link calls, writing a debug string or otherwise recording the failure. The function `OutputDebugString()` is available as part of the eMbedded Tools library.

It is recommended that you retain error code checking in a finished program to detect inadvertent system changes that may affect your application.

The file, `errcodes.txt`, is provided with the CE Link library. `errcodes.txt` contains the most recent list and description of error codes generated by the library. Although the `errcodes.txt` file does not need to be included in your development project, it is a useful tool for diagnosing problems with your code. In addition to the error codes, this file also includes probable causes of various problems and solutions.
4.1.1.1 Data Arrays Returned from Functions

API functions which acquire and return data determine internally how many data values will be read and allocate a corresponding block of memory for the returned data. If optional time-stamping is requested for these functions, the memory block also contains space for the time-stamp values, one per data reading. The pointers to the allocated arrays are returned in the return data structure for the API function. When the application is done with the memory block, it must be deallocated by calling the function i3free(), passing the pointer to the data array. If the optional time-stamping information is returned, that array will be automatically deallocated along with the data memory.

Optional Time Stamping

API functions that acquire data can optionally perform time stamping of that data. If time stamping is requested (by setting a flag in the input data structure for the function), the system clock will be read immediately after each data reading. The function will return two arrays, an array of I/O data and a corresponding array of timestamp data. The data type for the timestamp data is FILETIME, which is defined by the eMbedded Tools header files. This is a 64-bit value which gives time in hundreds of nanoseconds since January 1, 1601. The Windows CE API function FileTimeToSystemTime() provides a conversion to familiar date and time fields. Note that the time returned by CE Link API calls is Universal Time, not local time on the device, so you may need to use the SystemTimeToLocalTime() API function to convert.

4.1.1.2 Memory Allocation

Be sure to allocate enough memory for variable-length structures. Some structure elements in CE Link are declared as arrays with a nominal number of elements. These arrays may be prototypes for longer real arrays where the amount of data in the array depends on some parameter of the process being performed.

For example, the nsAIConfigureData structure is defined as:

typedef
struct nsAIConfigureData
{
    unsigned short count;
    nsAIList list[2];
} nsAIConfigureData;
typedef
struct nsAIList
{
    unsigned short module;
    unsigned short channel;
    unsigned short gain;
    unsigned short range;
    unsigned short diff;
    unsigned short zero;
} nsAIList;

The last array (list[2]) in the nsAIConfigureData structure is defined with a length of 2. If one or two channels are being configured, you can allocate an instance of the structure and use it as it is declared. However, if you are writing more than two channels, you must allocate enough memory to hold configuration information for all the channels.

The following example will allocate the space needed to hold configuration data for 16 analog input channels:

    int count = 16;
    nsAIConfigureData *cf;

    err = i3malloc(sizeof(short) + count*sizeof(nsAIList),(void **) &cf);

Once done with the configuration data, use i3free() to release the memory:

    i3free(cf);

See 4.2.1.5 Configuring the Channels to be Read for the complete sample.

Save time by double-checking your code to eliminate common memory allocation errors. The most common errors include:

- Not allowing for enough channels, especially when using ALL_MODULES_S, ALL_PORTS_S, ALL_PORT_BITS_S, or ALL_CHANNELS_S to specify a configuration or I/O operation.

- Failing to allow for the size of the actual array elements. You can use the sizeof operator to specify the number of bytes of data to allocate.
Failing to free memory blocks allocated by the API functions for returned data.

### 4.1.1.3 Specifying I/O Channels

API functions that address specific I/O channels do so by specifying a module number, a channel or port number, and in some cases a bit number within the port.

Modules are numbered starting with module 0, next to the CPU module. Channels or ports are numbered from 0 on each module. Digital modules which support bit functions number the bits from 0 within each port.

Some API functions can access multiple channels or ports. Special values are defined which facilitate reference to multiple channels. `ALL_MODULES_S` refers to all modules which are supported by the API function referencing this module value. `ALL_CHANNELS_S` refers to all channels on the specified module or modules supported by the API function. `ALL_PORTS_S` refers to all digital ports on the specified module or modules supported by the API function. `ALL_PORT_BITS_S` refers to all digital bits within the specified digital port(s).

### 1.1.61 Header Files

Every program that will interact with the CE Link library requires the C header file, `edasapi.h`. The `edasapi.h` file includes the declaration files for the various function call types, such as analog input, digital I/O, and so forth. Most applications written for the CE Link library also require the `edascons.h` and `sysutil.h` files. `edascons.h` declares various constant values used in the I/O function calls. `sysutil.h` declares various constant values used in the time and memory allocation/deallocation calls.

### 1.1.62 DLLs on the EDAS CE

The actual functions in the CE Link library are located in the DLL file `edasapiE.dll`. The `edasapiE.lib` library resolves function calls into references in the `edasapiE.dll`. Other DLLs, such as `anlg_inE.dll` and `dig_ioE.dll`, provide specific hardware capabilities for individual module types.

The DLLs for CE Link are pre-loaded on the EDAS CE as shipped from the factory. The `edasapiE.dll`, along with I/O specific DLLs (`anlg_inE.dll`, `dig_inE.dll`, etc) are located in the `\Storage Card\hardware` folder on the EDAS CE.

The DLLs are used by multiple programs such as Net Link Server (`suprcatE.exe`), WebDevice (wd_edasce.exe), in addition to the programs
you develop. It is therefore recommended that they be left in this location.

1.1.63 Registry Entries on the EDAS CE

The registry on the EDAS CE contains several entries associated with CE Link.

The EDASpath entry specifies the location of the CE Link DLLs for the various modules that might be attached to the system. If this entry is missing or empty, the directory where the edasapiE.dll file is located is used.

The EDASid entry specifies the hardware ID that will be returned by the nsSYSInquire() function. See 4.2.1.31 Obtaining System Information. This will not normally affect a CE Link application, but an appropriate value may be required by remote Net Link applications designed to be compatible with both the EDAS CE and EDAS products. The default value for EDASid is 0xED40.

The EDASprior entry is used by the Net Link Server (suprcatE.exe), application to optionally accelerate threads belonging to it.

The CE Link registry entries can be found in HKEY_LOCAL_MACHINE\SOFTWARE\Intelligent Instrumentation\EDAS API.
4.2 CE Link API Function Descriptions

1.1.64 Initialization and De-initialization Calls

There is one mandatory initialization call, nsHWInit(), that must be called before any of the other CE Link function calls. Before exiting any application that includes the initialization calls, the program should call the deinitialization function nsSWDeinit().

4.2.1.1 Initialization

This function initializes the hardware, determines the I/O modules that are present, establishes any interrupts they may require, assigns I/O locations, and so forth. It also initializes the software operating environment for CE Link. It loads support files for various modules present in the system, assigns channels, and so forth.

Function call in C++

long nsHWInit(void);

Parameters
None

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

STARTUP_ERR (-5)
nsSWDeinit() was called before nsHWInit().

SED_MOD_FAIL_ERR (-6)
Module failure. Contact the factory.

SED_GEOG_FAIL_ERR (-7)
Module mapping failure. Contact the factory.
4.2.1.2 Opening an I/O Session

This function establishes communication with the local I/O hardware. It returns an owner identification value in the owner field of the `nsOpenSessionReturn` structure. This identification value must be passed to each subsequent call to CE Link.

A single application may open more than one I/O session, for example, if it creates multiple threads which operate independently. Processes started and I/O channels locked by one session will not be accessible to another session.

**Function call in C++**

```c++
long nsOpenSession(void *res, nsOpenSessionReturn *ret);
```

typedef
struct nsOpenSessionReturn
{
    unsigned long owner;
} nsOpenSessionReturn;

**Parameters**

- `res` Reserved parameter. Set to NULL (0).
- `ret` Specifies pointer to return information from the call. The `nsOpenSessionReturn` structure contains a single field, `owner`, whose value is to be passed to all subsequent CE Link calls.

**Return Value Error Codes**

- Zero value: Indicates the function completed successfully.
- Non-zero value: Indicates failure.
4.2.1.3 Closing the I/O Session

This function closes the session established with the nsOpenSession() call. Use this function call to close any open sessions prior to deinitialization of the CE Link drivers.

**Function call in C++**

```c
long nsCloseSession(unsigned int caller, void *res1, void *res2);
```

**Parameters**

- **caller**  
  Indicates the session to be closed. Pass the value returned by the call to nsOpenSession().
- **res1**  
  Reserved parameter. Set to NULL (0).
- **res2**  
  Reserved parameter. Set to NULL (0).

**Return Value Error Codes**

- Zero value: Indicates the function completed successfully.
- Non-zero value: Indicates failure.

4.2.1.4 Deinitialization

This function deinitializes the software operating environment for CE Link. It frees any memory allocations, unloads support files for I/O modules, and so forth. Use this deinitialization function as the final call in any program that calls nsHWInit().

**Function call in C++**

```c
long nsSWDeinit (void);
```

**Parameters**

None

**Return Value Error Codes**

- Zero value: Indicates the function completed successfully.
- Non-zero value: Indicates failure.

1.1.65 Unpaced Analog Input Calls

Analog input functions in CE Link are accomplished through two types of data acquisition, paced and unpaced. Paced acquisitions are performed
using a hardware signal to start the analog-to-digital conversions. Unpaced acquisitions are performed using software start commands to begin the conversions. For information on paced acquisitions, see section 1.1.66 Paced Analog Input Calls.

### 4.2.1.5 Configuring the Channels to be Read

This function specifies the configuration for each channel to be read using the `nsAIRead()` function. Use this call to configure all channels before they are read.

**Function call in C++**

```c
long nsAIConfigureList(unsigned int caller, nsAIConfigureData *data, void *res);

typedef
struct nsAIConfigureData
{
    unsigned short count;
    nsAIList list[2];
}nsAIConfigureData;

typedef
struct nsAIList
{
    unsigned short module;
    unsigned short channel;
    unsigned short gain;
    unsigned short range;
    unsigned short diff;
    unsigned short zero;
} nsAIList;
```

**Parameters**

- **caller** Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data** Specifies an `nsAIConfigureData` input structure, described below.
- **res** This parameter is reserved and should be set to NULL (0).
**Fields**

nsAIConfigureData:
- **count**: Specifies the number of entries in the *list* to be configured.
- **list**: Declared as a two-element array of channel configuration structures. Be sure to allocate enough space for `count` list entries.

nsAIList:
- **module**: Module number, or ALL_MODULES_S.
- **channel**: Channel number, or ALL_CHANNELS_S.
- **gain**: Gain value. Supported gains are dependent on hardware.
- **range**: Range value. Supported ranges are dependent on hardware. When using the ZEROto20ma (0-20mA) range, differential mode may not be selected. If using ZEROto20ma or UNIPOLAR_10, the returned data should be declared as `unsigned short`.
- **diff**: Differential mode flag. A value of 1 specifies differential mode; 0 specifies single-ended mode. This parameter should be set to properly match the hardware configuration (which may be set through switches on the module).
- **zero**: Auto-zero correction. Specifies internal zero-reference signal (BF_HW_ZERO), an external channel number on the same module to which a zero-reference signal is connected, or no auto-zero correction (BF_NO_ZERO).

**Return Value Error Codes**

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

**Example**

```c
{
    long err;
    int count = 16;
    nsAIConfigureData *cf;

    err = i3malloc(sizeof(short) + count*sizeof(nsAIList),(void **)&cf);

    cf->count = count;
    for(i = 0; i < count; i++)
    {
        cf->list[i].module =1;
        cf->list[i].channel = i;
        cf->list[i].gain = 1;
```
```
cf->list[i].range = BIPOLAR_20;  // ±10 V range
cf->list[i].diff = 0;  // single-ended
    cf->list[i].zero = AI_NO_ZERO;  // no auto-zero correction
```

```c
err = nsAIConfigureList (sess, cf);
if(err)
{
    // Handle error.
}
    i3free(cf);
}
```

**Hardware Specifics**

**EDAS-2003M**

- In single-ended mode, the module supports 16 channels.
- In differential mode, the module supports 8 channels. Note that differential mode connects two single-ended channels (e.g. 0 and 8, 1 and 9), using the higher-numbered channel as the differential return. Only channels 0 through 7 can be configured as differential.
- If a differential channel is configured, its return connection can not be configured as a single-ended channel.
- When CJC (cold-junction compensation) is connected, it uses differential channel 0.
- Supported gains are 1, 10, and 100.
- Supported ranges are `BIPOLAR_20` (±10 V), `UNIPOLAR_10` (0-10 V) and `ZEROto20ma` (0-20 mA). When using `ZEROto20ma` range, differential mode may not be selected.
4.2.1.6 Reading a Previously Configured Analog Input Channel

The `nsAIRead()` function reads a previously-configured analog input channel. Prior to making this function call, configure the channel using the `nsAIConfigureList()` call.

**Function call in C++**

```c
long nsAIRead(unsigned int caller, nsAIReadData *data, nsAIReadReturn *ret);
```

typedef

```c
struct nsAIReadData
{
    unsigned short module;
    unsigned short channel;
    unsigned long count;
    unsigned short tflag;
} nsAIReadData;
```

typedef

```c
struct nsAIReadReturn
{
    unsigned long length;
    short *data;
    FILETIME *ft;
} nsAIReadReturn;
```

**Parameters**

- **caller** Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data** Specifies an `nsAIReadData` input structure, described below.
- **ret** Specifies an `nsAIReadDataReturn` returned data structure, described below.
Fields

nsAIReadData:
module: Module number, or ALL_MODULES_S.
channel: Channel number, or ALL_CHANNELS_S.
count: Number of samples to be read from each channel.
tflag: Time stamp flag. Set to 1 to return time stamp information for each sample.

nsAIReadReturn:
length: The number of samples returned in the resultant data array.
data: Pointer to the memory block allocated by the function to contain the returned data. When the data and ft pointers are no longer needed, the caller must use the i3free() function to free the data buffer allocated by this call.
ft: Optional timestamp data, returned as the system time in FILETIME format.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.
AI_CHAN_CONFIG_ERR (-2000)
This warning code indicates that a channel was not configured before its first read. In this case, the channel is read at 0-10 volts, gain of 1, single-ended. After the first read, the channel is treated as if the user configured it with these parameters.

Returned Data

The nsAIReadReturn structure contains a pointer to the returned data from the specified channel or channels. The function allocates enough space to hold the returned data. If multiple readings are specified by the count parameter, all readings of a channel are taken before moving on to the next channel (if any).

The function will also allocate memory to hold the timestamp information if it is requested. The data and ft fields are set to point to the memory blocks by the function before returning to the caller. When the data and ft pointers are no longer needed, the caller must use the i3free() function to free the data buffer allocated by this call. The data block is passed to i3free() to free the buffer; the ft block will be automatically freed along with the data block.

If the range codes ZEROto20ma or UNIPOLAR_10 are used, the returned data should be treated as unsigned short data. The nsAIReadReturn structure is defined with a short* data element. If the application uses a unipolar range, declare the pointer variable you use to access the data as unsigned short, and cast the pointer to short* when you assign it to the value returned in the structure. If the data pointer is not declared as unsigned short, when
you apply the conversion shown below, the data will not be interpreted correctly.

**Current-Input Data**

The current input mode **ZEROto20ma** returns an unsigned short value with a range of 0 to 65535, with zero corresponding to 0 mA and 65535 corresponding to 20 mA minus one LSB, or about 19.99969 mA.

To convert the data to current in mA:

```
unsigned short *pnt;
double current;
nsAIReadData inpt;
nsAIReadReturn outpt;
...
err = nsAIRead (session, &inpt, &outpt);
pnt = (unsigned short *)outpt->data;
...
current = *pnt * 20.0 / 65536.0;
```

**Hardware Specifics**

**EDAS-2003M**

- If **ALL_CHANNELS_S** is specified, the returned data array will contain 16 entries for a module. If a channel can not be read because it is a differential return for a channel configured in differential mode, the value returned will be zero.

**Example**

```
{
    int err;
    nsAIReadData dat;
    nsAIReadReturn ret;
    int count = 100;  // 100 samples from the channel.

    dat.module = 1;
    dat.channel = 0;
    dat.count = count;  // Read count samples from the channel.
    dat.tflag = 1;  // Read timestamp information for each sample.
```
ret.length = 0;
ret.data = 0;
ret.ft = 0;

err = nsAIRead (sess, &dat, &ret);
if (err)
{
    // Handle error.
}
else
{
    // Process data.
}
// Note that ft is not freed. Freeing ret.dat also frees the
// allocation for ft.
i3free (ret.data);

1.1.66 Paced Analog Input Calls

Paced acquisitions are performed using a hardware signal to start the analog-to-digital conversions. Unpaced acquisitions are triggered by a software command and are not synchronized with any particular event. For information on unpaced acquisitions, see section 1.1.65 Unpaced Analog Input Calls.

Paced acquisitions may occur at higher rates than unpaced conversions. Since they are controlled by hardware signals, these acquisitions have repeatable sample intervals that are very important when capturing waveforms.

You can configure more than one paced acquisition at once, but there can be only one acquisition at a time for each analog input module in your system. If you attempt to configure a second paced acquisition for an analog input module, it will replace the first acquisition.

You can run more than one paced acquisition at once. However, the demand this places on system resources will limit the rate at which the acquisitions can be run, as will other tasks being performed during the acquisition, including other applications which may be running.
4.2.1.7 Configuring a Paced Analog Input Process

The `nsAIHSConfigureList()` function establishes the configuration used for a paced analog input process. The configuration describes the following attributes of the process:

- Event that starts the process.
- Event that stops the process.
- Signal that initiates analog input conversions.
- Signal that acts as a trigger.
- Configuration information for each channel to be read by the process.
- Amount of data to be acquired by the process.

**Function call in C++**

```c
long nsAIHSConfigureList(unsigned int caller, nsAIHSConfigureData *data, nsAIHSConfigureReturn *ret);
```

```c
typedef struct nsAIHSConfigureData
{
    short start;
    short stop;
    short pacer;
    short trigger;
    short count;
    nsAIList list[2];
} nsAIHSConfigureData;
```

```c
typedef struct nsAIList
{
    unsigned short module;
    unsigned short channel;
    unsigned short gain;
    unsigned short range;
    unsigned short diff;
```
unsigned short zero;
} nsAIList;

typedef
struct nsAIHSConfigureReturn
{
    short clustersize;
    short handle;
} nsAIHSConfigureReturn;

**Parameters**

*caller*  Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.

*data*  Specifies the `nsAIHSConfigureData` input structure, described below.

*ret*  Specifies the `nsAIHSConfigureReturn` returned data structure, described below.
**Fields**

**nsAIHSConfigureData:**

**start:** Indicates the event that initiates the paced analog input process. Legal values are:
- `HS_START_COMMAND`. Using this option, the process starts immediately when the nsAIHSEnable() function is called with the appropriate parameters.
- `HS_START_TRIGGER`. Using this option, the process is enabled by the nsAIHSEnable() function. It begins after the trigger source becomes active. Between the time the process is enabled and the trigger occurs, the pacer is enabled, but conversions do not occur.

**stop:** Indicates the event that terminates the analog input process. Legal values are:
- `HS_STOP_COMMAND`. The process continues until it is stopped with a call to the nsAIHSEnable() function having appropriate parameters.
- `HS_STOP_TRIGGER`. The process runs until the specified trigger signal is detected and a specified number of post-trigger samples are collected (trigger delay).
- `HS_STOP_TC`. The process collects a specified number of samples from each channel used. When this specified number is reached, the process stops.

**pacer:** Indicates the source of the pacer signal used by the hardware to start analog to digital conversions. Legal values are:
- `SYNC_RG_OUT` or `RG_TYPE`. Uses the on-board rate generator.
- `SYNC_EXT_INT`. Uses the external interrupt input to the module.

**trigger:** Indicates the source of an optional trigger signal for the process. The process can be configured to start on the occurrence of a trigger or terminate some number of samples after the occurrence of a trigger. Legal values are:
- `SYNC_EXT_INT` or `SYNC_EXT_LOTOHI`. Use the external interrupt input and trigger on a low-to-high transition of the input.
- `SYNC_EXT_HITOLO`. Use the external interrupt input and trigger on a high-to-low transition of the input.
- `SYNC_NONE`. No trigger.

**count:** Number of elements of the `list` array.

**list:** Array of nsAIList structures describing the channels to be sampled.

**nsAIList:**

- **module:** Module number.
- **channel:** Channel number, or `ALL_CHANNELS_S`. If `ALL_CHANNELS_S` is not used, the successive channel numbers must be in order starting from 0.
- **gain:** Gain value. Supported gains are dependent on hardware. Gains must be the same for all entries in the channel list.
- **range:** Range value. Supported ranges are dependent on hardware. The range `ZEROto20ma` is not supported, and gains must be the same for all entries in the channel list.
- **diff:** Differential mode flag. A value of 1 specifies differential mode; 0 specifies single-ended mode. Values must be the same for all entries in the channel list.
- **zero:** Auto-zero correction. Must specify `BF_NO_ZERO`. 
nsAIHSConfigureReturn:
clustersize:
    Number of bytes required to store one data cluster.
handle:    A paced acquisition handle to be used in further high speed calls.

Return Value Error Codes
Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Hardware Specifics
EDAS-2003M
- In single-ended mode, the module supports 16 channels.
- In differential mode, the module supports 8 channels.
- Supported gains are 1, 10, and 100.
- Supported ranges are BIPOLAR_20 and UNIPOLAR_10 (current input is not supported with high-speed acquisition).
- The stop mode HS_STOP_TRIGGER is not supported.

4.2.1.8 Enabling or Disabling a Paced Analog Input Acquisition
This function enables (starts) or disables (stops) a paced analog input acquisition process. It can also free all resources associated with a disabled process.

Function call in C++

long nsAIHSEnable(unsigned int caller, nsAIHSEnableData *data, void *res);
typedef
struct nsAIHSEnableData
{
    short handle;
    short enable;
    unsigned long count;
    unsigned long delay;
    short clustersize;
    short free;
} nsAIHSEnableData;
Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies the nsAIHSEnableData input structure, described below.
res This parameter is reserved and should be set to NULL (0).

Fields

nsAIHSEnableData:

handle: The process to be enabled or disabled. Pass the handle value returned from nsAIHSConfigureList() function.

enable: Flag indicating whether the specified process should be enabled or disabled. For start-on-command processes, setting enable to 1 causes the process to begin; for start-on-trigger processes, the trigger is enabled. For any type of process, setting enable to 0 stops the process.

count: The maximum number of samples from each channel (clusters) to be acquired. For processes configured to stop on terminal count, this field indicates the number of clusters to be acquired before stopping the process. For other types of acquisitions, this field indicates the total acquisition buffer size in clusters.

delay: For stop-on-trigger processes, this field indicates the number of samples per channel (clusters) to be acquired after the occurrence of a trigger before the process is stopped automatically.

clustersize:
Set this field to the clustersize value returned by the nsAIHSConfigureList() function.

free: When the enable field is set to zero, this flag indicates whether the system resources for the process should be released. When a process completes, you typically disable the process but leave the free field set to zero to retain the data, read the data of interest from the buffers, then call nsAIHSEnable() again with the free field set to 1 to free the resources. If you want to re-run the acquisition without making changes to the configuration parameters, you do not need to free the resources.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

4.2.1.9 Reading High Speed Analog Input Data

The nsAIHSSRead() function returns to the application program the paced analog input data which has been acquired using the nsAIHSEnable() function.

Function call in C++

long nsAIHSSRead(unsigned int caller, nsAIHSSReadData *data, nsAIHSSReadReturn *ret);
typedef
struct nsAIHSReadData
{
    short handle;
    unsigned long offset;
    unsigned long samples;
    short clustersize;
    short commonformat;
    short wait;
    unsigned short *data;
} nsAIHSReadData;

typedef
struct nsAIHSReadReturn  {
    unsigned short status;
    unsigned long xsamples;
    BufInfo info;
} nsAIHSReadReturn;

typedef struct BufInfo  {
    short wrap;
    short running;
    short bytespercluster;
    short framespercluster;
    long size;
    long count;
    long currentcluster;
    long triggercluster;
    long nextcluster;
    unsigned long DMAword;
    unsigned long DMAaddr;
    unsigned long command;
    short unlatchedwrap;
    char reserved[132];
} BufInfo;
**Parameters**

`caller` Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.

`data` Specifies the `nsAIHSReadData` input structure, described below.

`ret` Specifies the `nsAIHSReadReturn` returned data structure, described below.

**Fields**

`nsAIHSReadData`:

- **handle**: Paced data acquisition handle returned by `nsAIHSConfigureList()`.
- **offset**: The buffer offset at which you want to start reading. The offset is the cluster number from the beginning of the buffer, or -1. Specify -1 if you want to continue reading from the position where the last read terminated. If you are reading from a buffer during an active acquisition which may cause the buffer to wrap, only use the -1 offset parameter. Gaps in your data may exist if the buffer write process passes the point at which you last read. If the acquisition is still running, this parameter is ignored and data will be returned as if -1 were specified.
- **samples**: Number of clusters to read.
- **clustersize**: The clustersize value returned by `nsAIHSConfigureList()`.
- **commonformat**: Converts data to common format. This parameter is ignored: data is automatically read in common format.
- **wait**: Flag commanding the function to wait until data is available to read it and return. If the acquisition has terminated, this parameter will be ignored.
- **data**: Pointer to a data buffer where the process data will be returned to the calling program. The calling program must allocate this data space. The buffer must be at least `samples * clustersize` bytes long.
nsAIHSReadReturn:

status: Flags indicating the status of the paced acquisition. Status bits that may be returned are:
- HS_PROCESS_RUNNING: The paced acquisition process is still active.
- HS_TRIGGER_OCCURRED: The paced acquisition process trigger has occurred.
- HS_BUFFER_WRAPPED: The paced acquisition process' circular buffer has wrapped.
- HS_FIFO_OVERRUN: A write to a full FIFO during the paced acquisition process was detected. This results in a loss of data.

xsamples: 
Actual number of samples read.

info: A BufInfo structure describing buffer characteristics. This field is provided for compatibility with previous software. Users are not encouraged to use this information to access the acquisition buffer directly.

BufInfo:

wrap: Buffer wrap flag. A 1 value indicates that the buffer acquisition pointer has wrapped past the end of the internal data buffer and restarted at the buffer's beginning. Same as status bit HS_BUFFER_WRAPPED.

running: Running flag. A 1 value indicates that the data acquisition has started and has not stopped. Same as status bit HS_PROCESS_RUNNING.

bytespercluster: 
Same as the clustersize value returned by nsAIHSConfigureList().

framespercluster:
Not used; set to 1.

size: Buffer size in clusters, normally determined by the count parameter for nsAIHSEnable().

count: Number of clusters in the buffer that are available for decoding. In the case of a triggered acquisition, this number will be zero (although the running flag is set) until the trigger occurs and samples begin to be acquired.

currentcluster:
The next cluster to be read, if nsAIHSRead() is called with offset equal to -1.

triggercluster:
Cluster number (from the beginning of the buffer) corresponding to the occurrence of the trigger, if any.

nextcluster:
Reserved

DMAword:
Reserved.

DMAaddr:
Reserved.

command:
Reserved.

unlatchedwrap:
Reserved.
Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Returned Data

After the function call, the user-allocated buffer will contain $\text{xsamples} \ast \text{clustersize}$ bytes of data. Each channel read returns two bytes of data, and the cluster size is determined by the number of channels configured. The channels are read in order beginning with Channel 0, with one channel read at each pacing event. When all channels have been read, the cycle begins again with Channel 0.

The total number of clusters available at the end of an acquisition process is normally determined by the count parameter of $\text{nsAIHSEnable()}$. If a stop-on-terminal-count process is terminated by software before all the data has been taken, there will be less data available. Also, if a stop-on-trigger process detects a trigger too early, the buffer may not be filled by the time the trigger delay elapses. If a trigger event or stop command stops the process in the middle of a cluster, the partial cluster will be discarded.

If the range code UNIPOLAR_10 is used, the returned data should be treated as unsigned short. The $\text{nsAIHSReadData}$ structure is defined with a short* data element. If the application uses a unipolar range, declare the pointer variable you use to access the data as unsigned short, and cast the pointer to short* when you assign it to the value returned in the structure. If the data pointer is not declared as unsigned short, when you apply the conversion shown above, the data will not be interpreted correctly.

This function is normally used to return data after a data acquisition has completed. However, you can use it to read data from the buffer as the acquisition proceeds. If you configure the acquisition to start on command and stop on command, for example, many buffers of data may be acquired, with new data overwriting older data in the buffer. By calling $\text{nsAIHSRead()}$ during the acquisition, you might transfer all this data to your application. However, there is no guarantee that you will always be able to read the data from the buffer before it is overwritten, even for quite slow acquisition rates. If you begin to read the data during the data acquisition and the acquired data subsequently wraps around and overwrites past the last data you read back, the status bit HS_BUFFER_WRAPPED will be set.
4.2.1.10 Status of High Speed Data

Provides the status of high speed data.

**Function call in C++**

```c
long nsAIHSStatus(unsigned int caller, nsAIHSStatusData *data, nsAIHSStatusReturn *ret);
```

```c
typedef
struct nsAIHSStatusData
{
    short handle;
}nsAIHSStatusData;
```

```c
typedef
struct nsAIHSStatusReturn
{
    BufInfo info;
    unsigned short status;
}nsAIHSStatusReturn;
```

**Parameters**

- **caller** Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data** Specifies the `nsAIHSStatusData` input structure, described below.
- **ret** Specifies the `nsAIHSConfigureReturn` returned data structure, described below.

**Fields**

`nsAIHSStatusData`:

- **handle**: Paced data acquisition handle returned by `nsAIHSConfigureList()`.

`nsAIHSStatusReturn`:

- **info**: A `BufInfo` structure describing buffer characteristics. This field is provided for compatibility with previous software. Users are not encouraged to use this information to access the acquisition buffer directly. The fields of this structure are described with the function `nsAIHSSRead()`.
- **status**: Flags indicating the status of the high-speed acquisition. Status bits that may be returned are described with `nsAIHSSRead()` function.

**Return Value Error Codes**

- **Zero value**: Indicates the function completed successfully.
- **Non-zero value**: Indicates failure.
1.1.67 Analog Output

Analog output support in CE Link consists of a configuration function and an output function.

4.2.1.11 Configuring the Channel to be Written

This function specifies the configuration for each channel to be written using the nsAOWrite() function. Use this call to configure all channels before they are written.

Function call in C++

long nsAOConfigureList (unsigned int caller, nsAOConfigureData *data, void *res);

typedef
struct nsAOConfigureData
{
    unsigned short count;
    nsAOList list[2];
} nsAOConfigureData;

typedef
struct nsAOList
{
    unsigned short module;
    unsigned short channel;
    unsigned short range;
} nsAOList;
Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies an nsAOConfigureData input structure, described below.
res This parameter is reserved and should be set to NULL (0).

Fields

nsAOConfigureData:
count: Number of entries in the list.
list: Defined as a two-element array of nsAOList structures. Be sure to allocate enough space for the count list entries, if you are configuring more than two channels.

nsAOList:
module: Module number, or ALL_MODULES_S.
channel: Channel number, or ALL_CHANNELS_S.
rangle: Range value. Supported ranges are dependent on hardware.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Hardware Specifics

EDAS-2006M

- The module supports 8 analog output channels.
- The only range code supported is UNIPOLAR_10 (0-10 V). The configuration function may be called but is not required.

4.2.1.12 Writing a Previously-Configured Analog Output Channel

The nsAOWrite() function reads a previously-configured analog output channel. Prior to making this function call, configure the channel using the nsAOConfigureList() call.

Function call in C++

nsAOWrite (unsigned int caller, nsAOWriteData *data, void *res);
typedef
struct nsAOWriteData
{
    unsigned short module;
}
unsigned short channel;
unsigned short data[2];
} nsAOWriteData;

Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies an nsAOWriteData input structure, described below.
res This parameter is reserved and should be set to NULL (0).

Fields

nsAOWriteData:
module: Module number of the channel(s) to write, or ALL_MODULES_S.
channel: Channel number to write, or ALL_CHANNELS_S.
data: Contains the data value(s) to write to the analog output channel(s).

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Output Data

If ALL_MODULES_S is specified, the same data will be written to all analog output modules in the system. If ALL_CHANNELS_S is specified, the length of the data buffer passed in this field must be long enough to contain data for all channels on the modules.

Hardware Specifics

EDAS-2006M

- The module supports 8 analog output channels.
- The configuration function may be called but is not required.

1.1.68 Digital Input and Output (Port or Byte)

Digital input and output functions in CE Link are accomplished through one of two types of data acquisition: port- or byte-wide operations, or individual channel operations (bit-wide operations).

Port-wide operations read and write the current states of all of the channels (up to 8) in a port. Port operations are limited to reading or writing the current state, 0 or 1, of each of the channels in a port, while individual channel operations can perform other operations,
depending on the capabilities of the hardware. Bit-wide operations are described in section 1.1.69 Digital Input and Output (Individual Bit).

4.2.1.13 Specifying the Configuration for Each Digital I/O Port

This function specifies the configuration for each digital I/O port. Although the configuration function allows for both input and output configurations, the ports may be fixed as input only or output only by the hardware design.

Function Call in C++

```c
long nsDIOConfigureList (unsigned int caller, nsDIOConfigureData *data, void *res);
```

typedef
```c
struct nsDIOConfigureData  
    unsigned short count;
    nsDIOList list[2];
} nsDIOConfigureData;
```

typedef
```c
struct nsDIOList  {
    unsigned short module;
    unsigned short port;
    short input;
    short handshake;
    unsigned short data;
} nsDIOList;
```

Parameters

- **caller** Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data** Specifies an nsDIOConfigureData input structure, described below.
- **res** This parameter is reserved and should be set to NULL (0).
**Fields**

nsDIOConfigureData:
- **count**: The number of entries in the *list* structure array.
- **list**: This is defined as a two-element array of channel configuration structures. To use the call to configure more than two channels, allocate enough memory for the number of ports to configure, and cast the pointer to memory to `nsDIOConfigureData*` to access the fields.

nsDIOList:
- **module**: Module number where the port to be configured is located, or ALL_MODULES_S.
- **port**: Port number to be configured, or ALL_PORTS_S.
- **input**: To configure the port as an input, set this flag to 1. To configure the port as an output, set this flag to zero.
- **handshake**: Set this value to zero.
- **data**: Contains the initial state values for ports configured as outputs.

**Return Value Error Codes**

- **Zero value**: Indicates the function completed successfully.
- **Non-zero value**: Indicates failure.

**Hardware Specifics**

**EDAS-2002M**
- The module supports two ports.
- Port 0 is configured in hardware as an input port, with 8 bits.
- Port 1 is configured in hardware as an output port with 4 bits.
A port must be configured as an input port before reading or as an output port before writing.

**EDAS-2004M**
- The module supports two ports with 8 bits each, both configured in hardware as inputs.
- The configuration function may be called but is not required.

**EDAS-2005M, EDAS-2010M, EDAS-2011M**
- The module supports two ports, both configured in hardware as outputs.
- Port 0 has 8 bits and port 1 has 4 bits.
- The configuration function may be called but is not required.
4.2.1.14 Reading a Digital Port

The `nsDIORead()` function reads a digital input port. Specify the module number and port on the module to be read by using a data structure passed as a parameter. If a digital port configured as an output has read back capability, this function will read the last data written to the port.

**Function Call in C++**

```cpp
long nsDIORead (unsigned int caller, nsDIOReadData *data, nsDIOReadReturn *ret);
```

```cpp
typedef
struct nsDIOReadData {
    unsigned short module;
    unsigned short port;
    short tflag;
} nsDIOReadData;
```

```cpp
typedef
struct nsDIOReadReturn {
    unsigned long length;
    unsigned short *data;
    FILETIME *ft;
} nsDIOReadReturn;
```

**Parameters**

- **caller** Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data** Specifies an `nsDIOReadData` input structure, described below.
- **ret** Specifies the `nsDIOReadReturn` returned data structure, described below.

**Fields**

**nsDIOReadData:**
- module: Module number, or ALL_MODULES_S.
- port: Port number, or ALL_PORTS_S.
- tflag: Time stamp flag. Set to 1 to return time stamp information for each sample.

**nsDIOReadReturn:**
- length: Number of ports read by the function call.
- data: Pointer to the memory block allocated by the function to contain the returned data. When the `data` and `ft` pointers are no longer needed, the caller must use the `i3free()` function to free the data buffer allocated by this call.
ft: Optional timestamp data, returned as the system time in FILETIME format.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Return Data

The nsDIOReadReturn structure contains a pointer to the returned data from the specified port or ports. The function allocates enough space to hold the returned data.

If ALL_PORTS_S is specified, the function will allocate enough space for all ports on the module, even if some ports are configured as outputs. A data value of zero will be returned for an output port. If ALL_MODULES_S is specified, data will be returned from all modules in the system having digital port input capability. Note that some output-only modules have read back capability and will return data to this function.

The function will also allocate memory to hold the time stamp of the information if it is requested. The data and ft fields are set to point to the memory blocks by the function before returning to the caller. When the data and ft pointers are no longer needed, the caller must use the i3free() function to free the data buffer allocated by this call. The data block is passed to i3free() to free the buffer; the ft block will be automatically freed along with the data block.

Hardware Specifics

EDAS-2002M

o One port, port 0 with 8 data bits, is supported for input.
  o The port must be configured as a digital input port before being read.

EDAS-2004M

o The module supports two ports with 8 bits each, both configured in hardware as inputs.
  o The configuration function may be called but is not required.

EDAS-2005M, EDAS-2010M, EDAS-2011M

o These output modules have ports that can be read back. The function will return the current output data.
  o The module supports two ports, both configured in hardware as outputs.
Port 0 has 8 bits and port 1 has 4 bits.

The configuration function may be called but is not required.

### 4.2.1.15 Writing to a Digital I/O Port

This function writes a digital output port. The module number and port number to be written, as well as the data to write, is specified through the data structure passed as a parameter.

#### Function Call in C++

```c++
long nsDIOWrite(unsigned int caller, nsDIOWriteData *data, void *res);
```

typedef

```c++
struct nsDIOWriteData
{
    unsigned short module;
    unsigned short port;
    unsigned short data[2];
} nsDIOWriteData;
```

#### Parameters

- **caller**: Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data**: Specifies an `nsDIOWriteData` input structure, described below.
- **res**: This parameter is reserved and should be set to NULL (0).

#### Fields

- **module**: Module number, or `ALL_MODULES_S`.
- **port**: Port number, or `ALL_PORTS_S`.
- **data**: Array containing the output value(s) for ports configured as outputs. If you are writing more than two ports, you must allocate a suitable amount of memory to hold the necessary data for each output port.

#### Return Value Error Codes

- **Zero value**: Indicates the function completed successfully.
- **Non-zero value**: Indicates failure.

#### Output Data

If `ALL_MODULES_S` is specified, the same data will be written to all modules in the system. If `ALL_PORTS_S` is specified, the function will expect enough data for all ports on the module, even if some ports are
configured as inputs. Data corresponding to input channels will be ignored.

**Hardware Specifics**

**EDAS-2002M**

- The module supports one port for output. Port 1 is the output port, and contains 4 bits.
- The port must be configured as an output port before being written.

**EDAS-2005M, EDAS-2010M, EDAS-2011M**

- The module supports two ports for output.
- Port 0 contains 8 bits and port 1 contains 4 bits.
- The configuration function may be called but is not required.

### 1.1.69 Digital Input and Output (Individual Bit)

Digital input and output functions in CE Link are accomplished through one of two types of data acquisition: port- or byte-wide operations, or individual channel operations (bit-wide operations).

Bit-wide operations read or write the data of a single channel. Depending on the hardware capabilities, digital input bit channels may be programmed as counters or latches. Port-wide operations are described in section 1.1.68 Digital Input and Output (Port or Byte).

**Portbit Masks**

The digital input bit functions require specification of a portbit parameter in addition to the module and port numbers. The portbit may be a number from 0 to 7, or a portbit mask. A portbit mask has a low byte of 0 and a high byte in which bits within the port are designated by 1 bits. For example, the portbit value 0x0500 designates bits 0 (0x0100) and 2 (0x0400).

### 4.2.1.16 Specifying the Configuration for Each Digital I/O Bit

The `nsDIOBITConfigureList()` function specifies the configuration for a list of digital I/O channels to be used in bit-wide operational modes. Several channels can be configured with one call to the function.

**Function Call in C++**

```cpp
long nsDIOBITConfigureList (unsigned int caller, nsDIOBITConfigureData *data, void *res);
```

typedef
struct nsDIOBITConfigureData
{
    unsigned short count;
    nsDIOBITList list[2];
} nsDIOBITConfigureData;

typedef
struct nsDIOBITList
{
    unsigned short module;
    unsigned short port;
    unsigned short portbit;
    unsigned short mode;
    unsigned long data;
} nsDIOBITList;

**Parameters**

**caller** Indicates the session making the call. Pass the value returned by the call to nsOpenSession().

**data** Specifies an nsDIOBITConfigureData input structure, described below.

**res** This parameter is reserved and should be set to NULL (0).
Fields

nsDIOBITConfigureData:
- **count**: Number of elements in the *list* array.
- **list**: This field is declared as a two-element array of nsDIOBITList structures. To use the call to configure more than two channels, you need to allocate memory for a list long enough to accommodate the number of bits you want to configure. To access the fields, cast the pointer to the allocated memory to nsDIOConfigureData*. Elements of this array specify the configuration for the channels to be acquired.

nsDIOBITList:
- **module**: Module number, or ALL_MODULES_S.
- **port**: Port number, or ALL_PORTS_S.
- **portbit**: Bit number of the channel being configured (0-7), or a portbit mask, or ALL_PORT_BITS_S.
- **mode**: Indicates the operational mode for which the channel is to be configured. Legal values of the mode field for digital I/O channels are:
  - DIOBIT_INPUT_NORMAL: Uses the channel in the same manner as if using port-wide operational mode. Channel has two states: ON (1) and OFF (0). Reading the channel returns a value of either 1 or 0.
  - DIOBIT_INPUT_LATCH_LOW: Traps a high-to-low transition on the input. Until the transition occurs, a zero value is returned when the channel is read, regardless of the input state. Once the transition is detected, the channel's value is 1 until the channel is reset. To reset the latch, call the nsDIOBITRead() function and set the reset field of the read data structure to 1. The channel must be enabled by calling the nsDIOBITEnable() function after it is configured and before any triggers will be detected.
  - DIOBIT_INPUT_LATCH_HIGH: Traps a low-to-high transition on the input. This is the only difference from the DIOBIT_INPUT_LATCH_LOW mode.
  - DIOBIT_INPUT_LATCH_STATE_CHANGE: This mode is a combination of the two previous modes, DIOBIT_INPUT_LATCH_LOW and DIOBIT_INPUT_LATCH_HIGH, trapping either a low-to-high or a high-to-low transition.
  - DIOBIT_INPUT_COUNT_UP: Counts the number of low-to-high transitions on the channel input. When the channel is read, the number of events counted since the last counter reset is returned. To reset the counter, call the nsDIOBITRead() function and set the reset field of the read data structure to 1. To enable the counter after configuring it and before it will count input transitions, you need to call the nsDIOBITEnable() function. Maximum value returned by the counter is \((2^{24} - 1)\).
  - DIOBIT_INPUT_COUNT_DOWN: This mode counts down from the initial count that you specify with the data value passed for the channel. Once enabled, the counter counts down from the specified value, stopping at zero and setting the overflow status bit. To reset the counter, call the nsDIOBITRead() function and set the reset field of the read data structure to 1. Maximum value returned by the counter is \((2^{24} - 1)\).
  - DIOBIT_INPUT_COUNT_EVENT: This mode works very similarly to the DIOBIT_INPUT_COUNT_UP mode, but is capable of counting much higher-speed transitions on the input. Before it can begin counting, it must be enabled by calling...
nsDIOBITEnable(). The counter is able to count up to a value of 65535, or \(2^{16} - 1\). On the next input transition after reaching this value, the counter rolls over to a value of zero and sets the overflow status bit. To reset the counter, call the nsDIOBITRead() function and set the reset field of the read data structure to 1.

DIOBIT_OUTPUT_NORMAL. This mode behaves as if the channel configured was being used as part of a port-wide operation. The channel has two values, 0 or 1, that can be set through the nsDIOBITWrite() function. The configuration data is ignored.

DIOBIT_OUTPUT_PULSE_LOW. Use this mode when a momentary output is needed and the normal state of the output is high (it pulses low). The configuration data value passed for the channel is the number of milliseconds the output will pulse low when it is triggered with a call to nsDIOBITEnable() with the enable parameter set to 1. To reset the output, call nsDIOBITEnable() with the enable parameter set to 0.

DIOBIT_OUTPUT_PULSE_HIGH. Use this mode when a momentary output is needed and the normal state of the output is low (it pulses high). The configuration data value passed for the channel is the number of milliseconds the output will pulse high when it is triggered with a call to nsDIOBITEnable() with the enable parameter set to 1. To reset the output, call nsDIOBITEnable() with the enable parameter set to 0.

DIOBIT_OUTPUT_DELAY_LOW. This mode causes the output to be in an initially high state. After a trigger from the nsDIOBITEnable() call with the enable parameter set to 1, the output transitions to the low state after a specified period of time. The delay time in milliseconds is specified by the configuration data value for the channel. To reset the output, call nsDIOBITEnable() with the enable parameter set to 0.

DIOBIT_OUTPUT_DELAY_HIGH. This mode works in a very similar manner to DELAY_LOW, but it is initially low and transitions to high after the trigger and delay period. To reset the output, call nsDIOBITEnable() with the enable parameter set to 0.

DIOBIT_OUTPUT_SQUARE_WAVE. Generates a square wave output waveform from the channel. The period of the waveform is specified in milliseconds by the configuration data parameter. The waveform is started and stopped by calling nsDIOBITEnable() with the enable parameter set appropriately.

The additional data is the initial data (normal output mode), pulse or delay duration in milliseconds (pulse and delay modes), or the pulsewidth duration in milliseconds (square wave). For pulse and delay outputs, configure and disable functions set the initial level of the output. Pulse Low and Delay Low are initially high, Pulse High and Delay High are initially low.

**Return Value Error Codes**

- **Zero value:** Indicates the function completed successfully.
- **Non-zero value:** Indicates failure.

**Hardware Specifics**

EDAS-2002M

- Port 0 is the input port on the module.
o Any of the 8 bits in the input port can be configured for normal input, count up, count down, or latch modes.
o Only bit 0 can be configured for event counter mode.
o Port 1 is the output port on the module.
o Any of the 4 bits in the output port can be configured for any of the supported output modes.

4.2.1.17 Enabling or Disabling a Digital Channel or Group of Channels

The nsDIOBITEnable() function is used to enable or disable a digital channel or group of channels. The channel(s) to be enabled or disabled is specified in the data structure passed to the function. The enable function acts as a trigger for pulsed and delayed output modes.

Function Call in C++

long nsDIOBITEnable(unsigned int caller, nsDIOBITEnableData *data, void *res);

typedef
struct nsDIOBITEnableData
{
  unsigned short module;
  unsigned short port;
  unsigned short portbit;
  short enable;
} nsDIOBITEnableData;

Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies an nsDIOBITEnableData input structure, described below.
res This parameter is reserved and should be set to NULL (0).
Fields

nsDIOBITEnableData:
module: Module number, or ALL_MODULES_S.
port: Port number, or ALL_PORTS_S.
portbit: Bit number, or a portbit mask, or ALL_PORT_BITS_S.
enable: Flag, set to 0 to disable the channel, or 1 to enable the channel. For multiple bits, see the discussion below.

Return Value Error Codes
Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Specifying Bit Enable Flags
If multiple bits are specified using a portbit mask, the enable flag is also specified as a mask. A 1 bit in the enable mask low order byte corresponds to an enable flag of 1 for that bit. A 0 bit in the enable mask corresponds to an enable flag of 0 for that bit. The enable bit is ignored for bits that are not selected by the portbit parameter. For example, if portbit is set to 0x0f00 and enable is set to 0xc0, bits 0 and 1 of the port would be disabled and bits 2 and 3 would be enabled. Bits 4 through 7 would not be affected.

Operating Modes
For normal input and normal output modes, this function is not required; these bits are enabled automatically when they are configured.

For latch modes, the enable state determines whether the change of state of the input is latched or ignored. Reset the latch by calling nsDIOBITRead() with the reset-on-read-flag set. The latch modes are:

- DIOBIT_INPUT_LATCH_LOW
- DIOBIT_INPUT_LATCH_HIGH
- DIOBIT_INPUT_LATCH_STATE_CHANGE

For counter input modes, the enable state of the channel determines whether incoming events are counted or ignored. The counter input modes are:

- DIOBIT_INPUT_COUNT_UP
- DIOBIT_INPUT_COUNT_DOWN
- DIOBIT_INPUT_COUNT_EVENT
For pulse and delay output modes, enabling the channel triggers or retriggers the pulsed or delayed behavior (driving the pulse-low output low for the configured period, for example). When a pulsed or delayed output is retriggered, its output is initially reset. To reset the initial output level prior to retriggering, disable the channel first. 

The output pulse and delay modes are:

- DIOBIT_OUTPUT_PULSE_LOW
- DIOBIT_OUTPUT_PULSE_HIGH
- DIOBIT_OUTPUT_DELAY_LOW
- DIOBIT_OUTPUT_DELAY_HIGH

For the square wave mode, the enable state of the channel determines whether the output is active or not.

- DIOBIT_OUTPUT_SQUARE_WAVE

4.2.1.18 Reading Data from a Digital Input Channel

The nsDIOBITRead() function reads the current data state from a digital input channel. For example, a digital input channel configured for a counter mode can be read to return the number of events which have been counted since the last reset.

**Function Call in C++**

```c++
long nsDIOBITRead(unsigned int caller, nsDIOBITReadData *data, nsDIOBITReadReturn *ret);
```

typedef
struct nsDIOBITReadData
{
    unsigned short module;
    unsigned short port;
    unsigned short portbit;
    short reset;
    short tflag;
} nsDIOBITReadData;

typedef
struct nsDIOBITReadReturn
{
    unsigned long length;
```
unsigned long *data;
unsigned short *status;
FILETIME *ft;
} nsDIOBITReadReturn;

**Parameters**

- **caller**: Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data**: Specifies an `nsDIOBITReadData` input structure, described below.
- **ret**: Specifies an `nsDIOBITReadReturn` returned data structure, described below.

**Fields**

**nsDIOBITReadData:**
- **module**: Module number, or `ALL_MODULES_S`.
- **port**: Port number, or `ALL_PORTS_S`.
- **portbit**: Bit number, or portbit mask, or `ALL_PORT_BITS_S`.
- **reset**: Reset flag. To reset a latch mode or counter mode channel after reading, set this field to 1. For **latch mode** channels, the latch trigger indication is reset and the channel awaits a new trigger. For **counter mode** channels, the count value is reset to the initial value. When the reset field contains a value of zero, the channel is not reset. For multiple bits, see the discussion below.
- **tflag**: Timestamp flag. Set to 1 to return timestamp information for each sample.

**nsDIOBITReadReturn:**
- **length**: Number of data elements returned by the function.
- **data**: Pointer to an array, allocated by the function, in which the data values are returned. When the **data**, **status**, and **ft** arrays are no longer needed, they must be deallocated by passing the **data** pointer to the `i3free()` function (this call automatically frees the **status** and **ft** arrays).
- **status**: Pointer to an array where the status of each channel read by the function is returned. The **status** array is allocated by the function before it returns.
- **ft**: Optional timestamp data, returned as the system time in `FILETIME` format.

**Return Value Error Codes**

- **Zero value**: Indicates the function completed successfully.
- **Non-zero value**: Indicates failure.
Returned Data

If the module contains both input and output ports, specify only input ports. If ALL_PORT_BITS_S is specified, memory will be allocated in the returned data array for all bits in the port. If only some bits have been configured, those bits will be read and the others ignored.

The reset field indicates whether a given channel in latch mode or counter mode should be reset after the read is complete. For latch mode inputs, resetting the channel clears the latch, readying the channel to detect another triggering transition. For counter mode inputs, resetting the channel resets the counter to the preload value. When ALL_PORT_BITS_S is passed as the portbit number, the reset parameter is treated as a bit field. Bits which have 1’s in the reset value in the bit position corresponding to their bit number are reset and those which have 0’s in the reset value are not. If ALL_PORTS_S is passed as the port number, the same reset value is applied to all ports.

4.2.1.19 Setting the Output State of Digital Output Bits

The nsDIOBITWrite() function is used to set the output state of digital output bits configured for DIOBIT_OUTPUT_NORMAL mode.

Function Call in C++

```c
long nsDIOBITWrite(unsigned int caller, nsDIOBITWriteData *data, void *res);
```

typedef

```c
struct nsDIOBITWriteData
{
    unsigned short module;
    unsigned short port;
    unsigned short portbit;
    unsigned short data[2];
} nsDIOBITWriteData;
```

Parameters

- **caller** Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
- **data** Specifies an nsDIOBITWriteData input structure, described below.
- **res** This parameter is reserved and should be set to NULL (0).
Fields

nsDIOBITWriteData:
module: Module number, or ALL_MODULES_S.
port: Port number, or ALL_PORTS_S.
portbit: Bit number, or portbit mask, or ALL_PORT_BITS_S.
data: This array contains the data, one element per channel, which is to be written to the
specified channel(s). To write data to more than two channels, you must allocate enough
memory for the other structure fields and the data for all of the channels specified for
writing.

Output Data

If a portbit mask or ALL_CHANNELS_S is specified, the function will
expect enough data for all bits in the port, even if some bits are not
written. Data destined for a particular channel must occupy the
corresponding element of the array. If ALL_MODULES_S is specified, the
same data will be written to all modules in the system.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Example

The example below shows how the function might be called to write all of
the bits in a single port (a 4-channel port):

```c
int siz = sizeof(nsDIOBITWriteData) + sizeof(unsigned short) * 8;
nsDIOBITWriteData *data;
int err;

err = i3malloc (siz, (void **)&data);
// Set up the module, port number, and portbit.
data->module = 0;
data->port = 0;
data->portbit = ALL_PORT_BITS;

// Fill the data array for all eight bits.
data->data[0] = 0;
data->data[1] = 1;
```
data->data[2] = 0;
data->data[3] = 1;

err = nsDIOBITWrite (caller, data, 0);
if (err)
{
  // Handle error.
}
else
{
  // Process data.
}
i3free (data);

1.1.70 Rate Generator Functions

CE Link provides functions to configure rate generator operation and enable and disable the rate generator, independent of paced acquisition operation.

4.2.1.20 Configuring the Rate Generator Frequency

The function nsRGConfigure() establishes the output frequency of a rate generator. Rate generators may be used to provide a timing base for paced analog acquisitions.

Function Call in C++

long nsRGConfigure (unsigned caller, nsRGConfigureData *data, void *res);
typedef
struct nsRGConfigureData
{
  unsigned short module;
  unsigned short channel;
  unsigned long data;
} nsRGConfigureData;
Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().

data Specifies an nsRGConfigureData input structure, described below.

res This parameter is reserved and should be set to NULL (0).

Fields

nsRGConfigureData:
module: Module number, or ALL_MODULES_S.
channel: Channel number, or ALL_CHANNELS_S.
data: Count data for programming the rate generator.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Hardware Specifics

EDAS-2003M

- The Rate Generator is a 24-bit clock divider.
- The output pulse is a 250 ns high pulse every N clocks of a 4 MHz base frequency. The count data must be in the range of 2 to \(2^{24} - 1\).
- The Rate Generator must be configured before being enabled, or before a paced acquisition using it is enabled.

4.2.1.21 Enabling and Disabling the Rate Generator

The function nsRGEnable() is provided for enabling and disabling the Rate Generator. The Rate Generator’s most common use will be as a pacer for a paced analog acquisition, in which case it will be enabled along with the acquisition. For other applications you will not need to enable/disable the Rate Generator directly.

Function Call in C++

```c
long nsRGEnable (unsigned caller, nsRGEnableData *data, void *res);
typedef
struct nsRGEnableData
{
    unsigned short module;
    unsigned short channel;
    unsigned short enable;
};
```
Parameters

**caller**
Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.

**data**
Specifies an `nsRGEnableData` input structure, described below.

**res**
This parameter is reserved and should be set to NULL (0).

Fields

`nsRGEnableData`:

- **module**: Module number, or `ALL_MODULES_S`.
- **channel**: Channel number, or `ALL_CHANNELS_S`.
- **enable**: Enable flag. Set to 1 to enable the Rate Generator; set to 0 to disable it.

Return Value Error Codes

- **Zero value**: Indicates the function completed successfully.
- **Non-zero value**: Indicates failure.

Hardware Specifics

EDAS-2003M

- The Rate Generator must be configured before being enabled, or before a paced acquisition using it is enabled.

1.1.71 Quadrature Functions

Quadrature and high-speed counter operations using the EDAS-2015M module are accomplished through a set of function calls designed for the purpose.

4.2.1.22 Specifying the Configuration for A Quadrature Channel

This function specifies the configuration for a single quadrature/counter channel. The channel’s mode, as well as the functions of the associated digital input and output bits, is set through this function call.

Function Call in C++

```cpp
long nsQUADConfigure (unsigned int caller, nsQUADConfigureData *data, void *res);
```

```cpp
typedef
struct nsQUADConfigureData 
{
    USHORT module;
    USHORT channel;
};
```
USHORT mode;
USHORT input0Mode;
USHORT input1Mode;

USHORT output0Mode;
USHORT output0InitialState;
USHORT output1Mode;
USHORT output1InitialState;

USHORT reserved1;

ULONG compareData;
ULONG preloadData;

} nsQUADConfigureData;

Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies an nsQUADConfigureData input structure, described below.
res This parameter is reserved and should be set to NULL (0).

Fields

nsQUADConfigureData:
module: Module number where the channel to be configured is located, or ALL_MODULES_S.
channel: Channel number to be configured, or ALL_CHANNELS_S.
mode: Operating mode for the channel. Can be QUADRATURE_INPUT to read quadrature data using the A and B inputs of the channel, or COUNTER_INPUT to use A as the counter clock and B as the up/down control input.

input0mode:
  The digital input bit mode for input 0 on the channel. This field can have the following values:
  DIOBIT_INPUT_NORMAL — General-purpose I/O. Read with nsDIOBITRead().
  DIOBIT_INPUT_PRELOAD_CONTROL — Preload control. When the input goes high, the counter value will be reloaded with the preload value set during the nsQUADConfigure() function call.
  DIOBIT_INPUT_LATCH_CONTROL — Latch control. When the input goes high, the count value will be latched into the internal latch registers.
  DIOBIT_INPUT_ENABLE_CONTROL — Enable control. When selected, the counter will
count only when the input is high. When the input is low, clocks signals presented to the A and B inputs will be ignored by the channel.

input1mode:
The digital input bit mode for input 1 on the channel. See input0mode for mode list.

output0mode:
The digital output bit mode for output bit 0 on the channel. The field can have the following values:
- DIOBIT_OUTPUT_NORMAL – General-purpose I/O. Write with nsDIOBITWrite().
- DIOBIT_OUTPUT_GE_COMPARE_RESULT – Comparator output of greater-than-or-equal-to comparison. The current count value is compared to the compare value configured with the nsQUADConfigure() call and, if the count value is >= the comparator value, the output is high. Otherwise, the output will be low.
- DIOBIT_OUTPUT_LT_COMPARE_RESULT – Comparator output of less-than comparison. The current count value is compared to the compare value configured with the nsQUADConfigure() call and, if the count value is < the comparator value, the output is high. Otherwise, the output will be low.
- DIOBIT_OUTPUT_NE_COMPARE_RESULT – Comparator output of not-equal comparison. The current count value is compared to the compare value configured with the nsQUADConfigure() call and, if the count value is <> the comparator value, the output is high. Otherwise, the output will be low.
- DIOBIT_OUTPUT_EQ_COMPARE_RESULT – Comparator output of equal comparison. The current count value is compared to the compare value configured with the nsQUADConfigure() call and, if the count value is = the comparator value, the output is high. Otherwise, the output will be low.
- DIOBIT_OUTPUT_GT_COMPARE_RESULT – Comparator output of greater-than comparison. The current count value is compared to the compare value configured with the nsQUADConfigure() call and, if the count value is > the comparator value, the output is high. Otherwise, the output will be low.
- DIOBIT_OUTPUT_LE_COMPARE_RESULT – Comparator output of less-than-or-equal-to comparison. The current count value is compared to the compare value configured with the nsQUADConfigure() call and, if the count value is <= the comparator value, the output is high. Otherwise, the output will be low.

output0InitialState:
For general-purpose output mode, the initial state of the output is set by nsQUADConfigure() to this value, 0 for low, 1 for high.

output1mode:
The digital output bit mode for output bit 0 on the channel. The field can have the same values as output0mode, above.

output1InitialState:
For general-purpose output mode, the initial state of the output is set by nsQUADConfigure() to this value, 0 for low, 1 for high.
reserved1:
  This field is reserved. It should be set to zero.

compareData:
  The comparator value for the channel. All comparisons are performed between the
  current count value and this value. The comparator output can drive the digital output
  bits for the channel and can generate an interrupt.

preloadData:
  The preload value for the channel. This value will be loaded into the channel counter
  when a preload signal is received or when reset-on-read is triggered during a read of the
  channel's value.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

4.2.1.23 Enabling and Disabling the Quadrature Channel

The function nsQUADEnable() is provided for enabling and disabling the
quadrature/counter channel. This function must be called to allow the
channel to begin counting events, unless one of the digital input bits
associated with the channel is configured for
DIOBIT_INPUT_ENABLE_CONTROL.

Function Call in C++

long nsQUADEnable (unsigned caller, nsQUADEnableData *data);
typedef
struct nsQUADEnableData
{
  unsigned short module;
  unsigned short channel;
  unsigned short enable;
}nsQUADEnableData;

Parameters

caller  Indicates the session making the call. Pass the value returned by the call to
  nsOpenSession().

data  Specifies an nsQUADEnableData input structure, described below.
Fields
nsQUADEnableData:
module: Module number, or ALL_MODULES_S.
channel: Channel number, or ALL_CHANNELS_S.
enable: Enable flag. Set to 1 to enable the channel; set to 0 to disable it.

Return Value Error Codes
Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Hardware Specifics
EDAS-2015M
- Channels are not necessarily enabled at precisely the same moment, so you must take external steps to assure that no clock signals occur which might be handled by one channel, but not by the other, because the enable operations are not simultaneous.

4.2.1.24 Reading a Quadrature Channel
The nsQUADRead() function reads a quadrature/counter channel. Specify the module number and channel on the module to be read by using a data structure passed as a parameter.

Function Call in C++

```c
long nsQUADRead (unsigned int caller, nsQUADReadData *data, nsQUADReadReturn *ret);
```

typedef
```c
struct nsQUADReadData {
    unsigned short module;
    unsigned short channel;
    short reset;
    short tflag;
} nsQUADReadData;
```

typedef
```c
struct nsQUADReadReturn {
    unsigned long length;
    unsigned long *data;
    unsigned long *status;
    FILETIME *ft;
```
Parameters

caller  Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data  Specifies an nsQUADReadData input structure, described below.
ret  Specifies the nsQUADReadReturn returned data structure, described below.

Fields

nsQUADReadData:
module:  Module number, or ALL_MODULES_S.
channel:  Channel number, or ALL_CHANNELS_S.
reset:  Reset flag. If this flag is set, the counter channel, after the read is completed, will be reloaded with the configured preload value, set in nsQUADConfigure().
tflag:  Time stamp flag. Set to 1 to return time stamp information for each sample.

nsQUADReadReturn:
length:  Number of channels read by the function call.
data:  Pointer to the memory block allocated by the function to contain the returned data. When the data, status and ft pointers are no longer needed, the caller must use the i3free() function to free the data buffer allocated by this call.
status:  Pointer to status return data for each channel read.
ft:  Optional timestamp data, returned as the system time in FILETIME format.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Return Data

The nsQUADReadReturn structure contains a pointer to the returned data from the specified channel or channels. The function allocates enough space to hold the returned data.

If ALL_CHANNELS_S is specified, the function will allocate enough space for all channels on the module. If ALL_MODULES_S is specified, data will be returned from all modules in the system having digital port input capability (subject to hardware support). The returned data consists of the count/position value for the indicated channel or channels. It is a 32-bit value ranging from 0 to 0xFFFFFFFF.

The status return array contains one status value for each channel read by the function. The status value is a 32-bit value consisting of 32 ‘flags’ which are defined by the following constants:
QUAD_INTR_PRELOAD - If this bit is set, a hardware preload signal has been sent to the channel since the last read.

QUAD_INTR_LATCH - If this bit is set, a hardware latch signal has been sent to the channel since the last read.

QUAD_INTR_IN0 - If this bit is set, the interrupt for general-purpose input bit 0 is active.

QUAD_INTR_GT - If this bit is set, the current count value is greater than the comparator value configured for the channel.

QUAD_INTR_EQ - If this bit is set, the current count value is equal to the comparator value configured for the channel.

QUAD_INTR_LT - If this bit is set, the current count value is less than the comparator value configured for the channel.

QUAD_INTR_UNDERFLOW - If this bit is set, counter has been given a count-down signal while its count value was zero. This results in the count value 0xFFFFFFFF being loaded.

QUAD_INTR_OVERFLOW - If this bit is set, the counter has been given a count-up signal while its count value was 0xFFFFFFFF. This results in the count value zero being loaded.

The memory for the status return array is allocated in the same memory block with the data array. When the data, status, and ft arrays are no longer needed, the data pointer should be passed to the i3free() function to release the memory for all three buffers.

The function will also allocate memory to hold the time stamp of the information if it is requested. The data, status and ft fields are set to point to the memory blocks by the function before returning to the caller. When the data, status and ft pointers are no longer needed, the caller must use the i3free() function to free the data buffer allocated by this call. The data block is passed to i3free() to free the buffer; the ft and status blocks will be automatically freed along with the data block.

**Hardware Specifics**

EDAS-2015M

- The use of ALL_MODULES_S is not supported by this module.

### 4.2.1.25 Configuring Quadrature Interrupts

This function configures the interrupt functionality of a quadrature/counter channel. Channels can generate interrupts on the occurrence of a variety of events, including comparison events. Being able to capture such events in an interrupt handler can allow you to
react more quickly to them. Note, however, that interrupt handlers have many restrictions on what operations they can perform and are much more difficult to debug than ordinary program code. You should not use an interrupt handler if a simple call to nsQUADRead() from the main program code will suffice.

Function Call in C++

```c
long nsQUADInterrupt(unsigned int caller, nsQUADInterruptData *data, void *res);
```

```c
typedef
struct nsQUADInterruptData
{
    USHORT module;
    USHORT channel;

    USHORT interruptMask;
    USHORT reserved;
    ULONG reservedX0;

    LONG (WINAPI *callback)( ULONG param, void *chn,
                              int chnNum, ULONG baseAddr );
    ULONG callbackParam;
} nsQUADInterruptData;
```

**Parameters**

- **caller**: Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data**: Specifies an `nsQUADInterruptData` input structure, described below.
- **res**: This parameter is reserved and should be set to NULL (0).

**Fields**

- **nsQUADInterruptData**:  
  - **module**: Module number. Note that ALL_MODULES_S is not supported.
  - **channel**: Channel number. Note that ALL_CHANNELS_S is not supported.

- **interruptMask**:  
  Bit field indicating which possible interrupt events should generate the interrupt for the channel. If set to 0, the interrupt is disabled. This should be done prior to closing any program that uses quadrature/counter interrupts to assure that the callback will not be made after the program has exited. Values for this mask include:  
  - `QUAD_INTR_PRELOAD` – interrupt generated when the preload hardware signal,
provided by one of the general-purpose inputs programmed for DIOBIT_INPUT_PRELOAD_CONTROL, is triggered.

QUAD_INTR_LATCH – interrupt generated when the latch hardware signal, provided by one of the general-purpose inputs programmed for DIOBIT_INPUT_LATCH_CONTROL, is triggered.

QUAD_INTR_IN0 – interrupt generated when general-purpose input bit 0 of the quadrature/counter channel transitions from low to high.

QUAD_INTR_GT – interrupt generated when the count value in the channel transitions from being less-than-or-equal to the comparator value, to being greater-than the comparator value.

QUAD_INTR_EQ – interrupt generated when the count value in the channel transitions from being not-equal to the comparator value, to being equal to the comparator value.

QUAD_INTR_LT – interrupt generated when the count value in the channel transitions from being greater-than-or-equal to the comparator value, to being less-than the comparator value.

QUAD_INTR_UNDERFLOW – interrupt generated when the counter counts in the down direction from a count value of zero, transitioning to 0xFFFFFFFF.

QUAD_INTR_OVERFLOW – interrupt generated when the counter counts in the up direction from a count value of 0xFFFFFFFF, transitioning to zero.

reserved:
   This field is reserved. It should be set to zero.

reservedX0:
   This field is reserved. It should be set to zero.

callback: A pointer to the callback function, located in your application code, which will be invoked by the module interrupt handler when the channel’s interrupt is fired.

callbackParam:
   A 32-bit value which will be passed by the module interrupt handler to the callback function, when it is called. You might pass a pointer to some data which the interrupt handler will need, or any other 32-bit value.

callback Parameters:
param: The value of the callbackParam passed when the interrupt was configured through nsQUADInterrupt().

chn: This parameter is for internal use only. Do not change it or the memory to which it points.
chnNum: The channel number of the quadrature/counter channel which generated the interrupt.

baseAddr:  
The base address of the module on which the channel is located. This value, in combination with the chnNum parameter, can be used to create a pointer which can be used to access the control registers of the module to read the counter value, set outputs, etc. CE Link API function calls may not be invoked from the interrupt handler callback, so all module I/O to be performed during interrupt handling must access the module registers directly.

**Return Value Error Codes**

- Zero value: Indicates the function completed successfully.
- Non-zero value: Indicates failure.

### 1.1.72 Alarm Functions

CE Link provides for asynchronous events, known as alarms, which may occur in the system. You can configure an alarm to be triggered by one of several types of events, such as a transition of a digital input bit. When the alarm is triggered, a local response may occur. A local response is an I/O operation that occurs on another channel on the EDAS device, such as a momentary output on a digital output bit. Additionally, the event may be logged directly to the EDAS CE, or by transmitting data to another device.

This section describes the functions used to configure and control alarms in CE Link.

### 4.2.1.26 Configuring an Alarm Process

The `nsALARMConfigure()` function configures an alarm process.

**Function Call in C++**

```c++
long nsALARMConfigure(unsigned int caller, nsALARMConfigureData *data, nsALARMConfigureReturn *ret);
```

typedef

```c++
struct nsALARMTypen
{
    unsigned short iotype;
    unsigned short module;
    union {
        struct
        {
            unsigned short channel;
        }
    }
}
```
unsigned short gain;
unsigned short range;
unsigned short differential;
unsigned short port;
unsigned short portbit;
unsigned short mode;
unsigned long data1;
unsigned long data2;
};
struct {
    unsigned short enable;
    unsigned short clustersize;
    unsigned short free;
    unsigned short ra, rb;
    unsigned short hndl;
    unsigned long count;
    unsigned long delay;
};

} nsALARMType;

typedef struct nsALARMConfigureData
{
    unsigned short type;
    unsigned short mode;
    nsALARMType input;
    nsALARMType output;
} nsALARMConfigureData;

typedef struct nsALARMConfigureReturn
{
    unsigned short alarm;
} nsALARMConfigureReturn;
Parameters

caller Indicates the session making the call. Pass the value returned by the call to
nsOpenSession().

data Specifies an nsALARMConfigureData input structure, described below.

ret Specifies an nsALARMConfigureReturn returned data structure, described below.
Fields

nsALARMConfigureData:
The input and output fields each reference an nsALARMType structure.

- **type:** Alarm type code. This field describes the type of action that will cause an alarm to occur. Supported types are:
  - ALARM_DIOBIT_LATCH_LOW: Digital input bit high-to-low latch trigger.
  - ALARM_DIOBIT_LATCH_HIGH: Digital input bit low-to-high latch trigger.
  - ALARM_DIOBIT_LATCH_STATE_CHANGE: Digital input bit low-to-high latch trigger.
  - ALARM_DIOBIT_TERMINAL_COUNT: Digital input bit down-counter reaching terminal count.
  - ALARM_DIOBIT_OVERFLOW_COUNT_SW: Digital input bit software counter overflow.
  - ALARM_DIOBIT_OVERFLOW_COUNT_HW: Digital input bit hardware event counter overflow.

- **mode:** Alarm mode code. The alarm may be configured as ALARM_CONTINUOUS. In continuous mode, the software will re-arm the alarm and the input channel after alarm processing is complete. In a single-shot mode, ALARM_SINGLE_SHOT, the alarm will be disabled after a single alarm occurrence and must be reenabled to detect another alarm condition.

- **input:** The contents of this field describe the alarm channel that will trigger the alarm. This field is an nsALARMType structure.

- **output:** Description of an output process that the software will perform in response to an input. This field is an nsALARMType structure.

nsALARMType:

- **iotype:** For alarm trigger channels, specifies the data type of the channel. For a digital input bit trigger for the alarm, this field would be set to DIOBIT_TYPE. For alarm response output channels, specifies the output data type. Supported output types are DIO_TYPE (digital port output), DIOBIT_TYPE (any digital bit output), AI_TYPE (paced analog acquisition), and AO_TYPE (analog output).

- **module:** Specifies the module location of the alarm trigger source or the alarm response channel.

- **channel, gain, range, differential:** These parameters describe an analog input or analog output channel.

- **port, portbit, mode:** These parameters describe a digital bit or byte, input or output.

- **data1, data2:** These parameters depend on the I/O type of the channel being described.
  - Digital input bit terminal count: $data1$ is the terminal count value.
  - Digital output bit: $data1$ is the value to be output (1 or 0).
  - Digital output byte: $data1$ is the value to be output to the port.
Analog output: data1 is the value to be output to the channel.

enable, clustersize, free, hndl, count, delay:
These parameters describe a high-speed analog input process that will be enabled (or
disabled) when the alarm occurs. See the description of nsAIHSEnable() in section
4.2.1.8 Enabling or Disabling a Paced Analog Input Acquisition for a more detailed
explanation of the parameters.

ra, rb: These parameters are reserved and should be set to zero.

nsAlarmConfigureReturn:
alarm: Returned handle. Use this value to identify the alarm in subsequent alarm API calls.

Return Value Error Codes
Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

4.2.1.27 Enabling or Disabling an Alarm
This function enables or disables an alarm. Call this function after
nsALARMConfigure() with the enable field of the data structure set to 1 to
place the alarm in a state of readiness to respond to its trigger event.
An alarm configured in single-shot mode can be re-armed by calling this
function after one alarm has been triggered. An alarm does not respond
to its trigger unless it is enabled.

Function Call in C++
long nsALARMEnable(unsigned int caller, nsALARMEnableData *data, void *res);
typedef
struct nsALARMEnableData
{
    unsigned short alarm;
    unsigned short enable;
} nsALARMEnableData;

Parameters

caller: Indicates the session making the call. Pass the value returned by the call to
nsOpenSession().
data: Specifies an nsALARMEnableData input structure, described below.
res: This parameter is reserved and should be set to NULL (0).
Fields

nsALARMEnableData:
alarm: Alarm handle returned by nsALARMConfigure, of the alarm to be enabled or disabled.
enable: Enable flag: 1 to enable. 0 to disable.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

4.2.1.28 Alarm Status

This function checks and returns the status of a specified alarm, active or not. The state of the alarm is returned separately from any output action or transfer option that might be set; it returns the current alarm state.

Function Call in C++

long nsALARMStatus(unsigned int caller, nsALARMStatusData *data, nsALARMStatusReturn *ret);

typedef
struct nsALARMStatusData
{
    unsigned short alarm;
} nsALARMStatusData;

typedef
struct nsALARMStatusReturn
{
    unsigned short status;
} nsALARMStatusReturn;

Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies an nsALARMStatusData input structure, described below.
ret Specifies an nsALARMStatusReturn returned data structure, described below.
Fields

nsALARMStatusData:
alarm: The alarm handle returned by nsALARMConfigure().

nsALARMStatusReturn:
status: Return status bits describing the alarm:
   ALARM_LATCHED: The alarm condition has occurred since the last time the alarm was enabled.
   ALARM_ACTIVE: The alarm condition has occurred since the last time the status was checked.
   ALARM_ERROR: An error has occurred in the alarm thread. The error returned reflects the thread error.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

4.2.1.29 Alarm Transfer Options

The alarm transfer options specify special operations which can be performed each time an alarm is triggered. Reports can be sent as a message over the network or logged to a local file, for example.

Function Call in C++

long nsALARMTransferOptions(unsigned int caller, xALARMTransferData *data, void *res, void *fcn);

typedef
struct nsALARMTransferData
{
    unsigned short alarm;
    char remote[80];
    unsigned short xfermode;
    unsigned short xferport;
    unsigned short xferdata;
    unsigned short nbytes;
    char message[1];
} nsALARMTransferData;
Parameters

caller  Indicates the session making the call. Pass the value returned by the call to nsOpenSession().

data    Specifies an nsALARMTransferData input structure.

res     This parameter is reserved and should be set to NULL (0).

fcn     Pointer to a network data transfer function if network transfer is desired. Sample code is provided.
**Fields**

`nsALARMTransferData`:

- **alarm**: Handle of the alarm which is having its options set.
- **remote**: For network data transfer, this field contains the IP address or host name of the remote device to which the alarm process will send the information, specified below, when the alarm is triggered. If the transfer mode is set to XFER_FILE_LOG, however, this field specifies the log file name to which the alarm data will be sent.
- **xfermode**: This field indicates the protocol class used for the communications with the remote device. The legal values are:
  - `XFER_SYNC` or `XFER_NONE`: Use this value to disable data transfer for the alarm.
  - `XFER_ASYNC_DATAGRAM`: Use this value to indicate that when the alarm is triggered, a single UDP datagram (non-connection packet) will be sent to the specified port on the specified host.
  - `XFER_ASYNC_STREAM_OPEN`: Use this value to indicate when the alarm is triggered, a TCP connection will be made to the indicated port on the indicated host, if it does not already exist, allowing the alarm packet to be sent over the connection. After the data transmission, the connection remains open until the remote device closes it or the alarm is freed.
  - `XFER_ASYNC_STREAM_CLOSE`: This option indicates operations identical to `XFER_ASYNC_STREAM_OPEN` with the exception that after the data is transferred, the connection to the remote device is closed and must be reopened if the alarm recurs.
  - `XFER_FILE_LOG`: This option indicates the alarm will write alarm message to a log file instead of sending a packet.
- **xferport**: For network data transfer, this field indicates the TCP or UDP port number on the remote host to which the datagram will be sent or to which the EDAS will connect before sending the alarm data. For serial data transfer, this field indicates which serial port is used for the transfer (0 = COM1, 1 = COM2, and so forth).
- **xferdata**: This field indicates what, if any, I/O data from the EDAS will be transmitted. The legal values for this flag are:
  - `XFER_HEADER`: Transfer fixed-length header before the data. Header format is defined by `nALARMHeaderType`, described below.
  - `XFER_DIGITAL_DATA`: Transfer all digital input data (read all bits and bytes from the EDAS unit).
  - `XFER_ANALOG_DATA`: Transfer readings from all analog input channels on the EDAS unit.
  - `XFER_TIME_ASCII`: Transfer the ASCII string representation of the current date and time, at the time of the alarm trigger. The time is Universal Time.
  - `XFER_TIME_BINARY`: Transfer a binary timestamp. The timestamp format is `FILETIME`, described in the timestamping discussion. The time is Universal Time.
- **nbytes**: This is the length of string message (not including terminating NULL). Indicates the number of bytes of explicit data (a string, for example), contained in the message field,
which will be transmitted when the alarm is triggered. You might use this message to identify the device from which the alarm is being sent.

**Message:**

An arbitrary data string to be included in the alarm packet. This field specifies any explicit data to be sent when the alarm is triggered. Note: The actual size of the structure allocated must allow for the length of the data in the message array.

**Res:** Not used, should be set to NULL (0).

**Fcn:** Pointer to a network data transfer function if network transfer is desired. Sample code is provided.

### Alarm Packets

This is a description of the format of the alarm packet. Fields are included if specified in `xferdata`.

<table>
<thead>
<tr>
<th>Field</th>
<th>Length (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header: alarm</td>
<td>2</td>
<td>Alarm handle</td>
</tr>
<tr>
<td>Header: length</td>
<td>2</td>
<td>Length of message excluding header</td>
</tr>
<tr>
<td>Header: status</td>
<td>2</td>
<td>Alarm status</td>
</tr>
<tr>
<td>Header: reserved</td>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>Data: message</td>
<td>nbytes</td>
<td>User-supplied message</td>
</tr>
<tr>
<td>Data: ASCII time</td>
<td>22</td>
<td>ASCII timestamp representation</td>
</tr>
<tr>
<td>Data: binary time</td>
<td>8</td>
<td>FILETIME timestamp representation</td>
</tr>
<tr>
<td>Data: digital bit</td>
<td>4 * num of bits</td>
<td>Digital bit data from all digital bit channels</td>
</tr>
<tr>
<td>Data: digital port</td>
<td>2 * num of ports</td>
<td>Digital byte data from all digital ports</td>
</tr>
<tr>
<td>Data: analog</td>
<td>2 * num of chan</td>
<td>Analog data from all analog channels</td>
</tr>
</tbody>
</table>

If the data options are specified, a large amount of data may be transferred, depending on the system configuration. You may want to define a structure based on your EDAS CE system configuration to aid in interpreting the data.

Digital bit data will be allocated for all digital I/O channels that can be configured as bits. Digital bits that are presently configured as outputs (or that are set in hardware as outputs) will return data values of zero, but space will be allocated for them. If the system contains digital modules that can only be read as bytes, no digital bit data space will be allocated for them, but an error code will be set which can be read by calling `nsALARMStatus()`.

Digital byte data will be allocated for all digital I/O ports and input ports. Ports that are presently configured as outputs (or that are set in hardware as outputs), may return meaningless data.

Analog data will be allocated for all analog channels. Channels that are unavailable because of other differential channel configurations will not...
be read. If a channel has not been previously configured, a warning code will be returned the first time it is read. Data will be properly interpreted as unsigned short if the channels are configured in a unipolar mode.

Multiple alarms may transfer data to the same TCP port. However, if the mode XFER_ASYNC_STREAM_OPEN is used, the port must be capable of accepting multiple simultaneous connections.

**Return Value Error Codes**

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

4.2.1.30 Alarm Resources: Disabling and Releasing

This function disables and releases all resources associated with the indicated alarm. If transfer options have been enabled, the TCP port (if any) is closed and the local file (if any) is closed.

**Function Call in C++**

```c
long nsALARMFree(unsigned int caller, nsALARMFreeData *data, void *res);
```

```c
typedef
struct nsALARMFreeData {
    unsigned short alarm;
} nsALARMFreeData;
```

**Parameters**

- **caller** Indicates the session making the call. Pass the value returned by the call to `nsOpenSession()`.
- **data** Specifies an nsALARMFreeData input structure, described below.
- **res** This parameter is reserved and should be set to NULL (0).

**Fields**

nsAlarmFreeData:
alarm Identifies the alarm that is being freed.

**Return Value Error Codes**

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.
1.1.73 System and Utility Functions

CE Link provides a number of system and utility functions for performing certain operations such as querying system configuration information.

I/O locking can also be used to prevent a secondary process on the unit from performing operations on a given channel. For example, this enables a CE Link program to ensure that the analog input configuration remains in effect and cannot be changed by another program running on the unit or by a remote program communicating with the unit.

4.2.1.31 Obtaining System Information

The `nsSYSInquire()` function returns information about the EDAS CE system including the ID, which identifies the type of the unit.

Function Call in C++

```c
long nsSYSInquire(void *res, nsSYSInfoReturn *info);
```

typedef

```c
struct nsSYSInfoReturn {
    unsigned short id;
    unsigned short resetcount;
    unsigned short length;
    unsigned long *data;
} nsSYSInfoReturn;
```

Parameters

- `res` This parameter is reserved and should be set to NULL (0).
- `info` Specifies an nsSYSInfoReturn structure, described below.

Fields

nsSYSInfoReturn:

- `id` Unit ID is normally 0xED40 but can be changed in the registry.
- `resetcount` Not used with the EDAS CE unit.
- `length` Returns a zero value.
- `data` Not used, remains NULL.

Return Value Error Codes

- Zero value: Indicates the function completed successfully.
- Non-zero value: Indicates failure.
4.2.1.32 Returning System Version Information

The `nsVERInquire` function returns information about the system, including the version information for the system and module ID information for all installed I/O modules, allowing the calling program to detect the system configuration.

**Function Call in C++**

```c
long nsVERInquire(void *res, nsVERInfoReturn *info);
```

typedef

```c
struct nsVERInfoReturn
{
    unsigned long EXECver;
    unsigned long PICver;
    unsigned short moduleID[16];
    unsigned long moduleVER[16];
    FILETIME systime;
    char reserved[16];
} nsVERInfoReturn;
```

**Parameters**

- `res`  This parameter is reserved and should be set to NULL (0).
- `info`  Specifies an `nsVERInfoReturn` structure, described below.
Fields

nsVERInfoReturn:
EXECver:
   ROM version (EDAS) or software version (EDAS CE)
PICver: PIC version (EDAS)
moduleID[16]:
   Returns module ID information for all installed I/O modules. Module positions which are unoccupied will contain hex values of 0xffff.
moduleVER[16]:
   Module returns version information to the caller.
systime: Time stamp indicates the unit’s clock setting at the time the function was called.
reserved[16]:
   Field not used

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

4.2.1.33 Locking an I/O Subsystem

An I/O subsystem in the EDAS unit can be locked to prevent another process from reconfiguring or using the I/O. I/O locking provides a means of assigning control over I/O to a specific owner. This function is useful in situations where network clients and/or local applications might be trying to use the I/O on the unit, each having a different configuration.

Function Call in C++

long nsIOLock(unsigned int caller, nsIOLockData *data, void *res);
typedef
struct nsIOLockData
{
   unsigned short iotype;
   unsigned short module;
   unsigned short channel;
   unsigned short lock;
} nsIOLockData;
Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies an nsIOLockData input structure, described below.
res This parameter is reserved and should be set to NULL (0).

Fields

nsIOLockData:
iotype: Indicates the type of I/O to be locked. Legal values depend on the I/O available on the selected module, including:
  DIO_TYPE: Digital I/O (port-wide operation). A single port on the module may be locked.
  DIOBIT_TYPE: Digital I/O (individual operation). All the bits in one port will be locked.
  AI_TYPE: Analog input. All channels on the module will be locked.
  AO_TYPE: Analog output. A single channel on the module may be locked.
  RG_TYPE: Rate generator.
module: The module number, dependent on the system configuration.
channel: The channel number, dependent on the system configuration.
lock: Set to 1 to lock the indicated subsystem; set to zero to unlock a previously-locked subsystem.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.
IO_LOCK_ERR (30)
IO_UNLOCK_ERR (31)
Indicates that at least one channel or port was locked by another owner and could not be locked or unlocked by the caller. If the call specified multiple channels and/or ports to be locked or unlocked, the change in lock status for some channels might have been successful.
**1.1.74 System Calibration**

Although the EDAS unit is calibrated at the factory, utility functions are provided to permit access to the calibration system. An EDAS CE module may have non-volatile (EE) storage for calibration values which can be read and written, and programmable DACs which are set from stored data in the EE storage.

**4.2.1.34 Reading Calibration Data**

The `nsEERead()` function reads calibration data stored in the EE. The address range is 0 to 31.

**Function Call in C++**

```c++
long nsEERead (unsigned int caller, nsEEReadData *data, nsEEReadReturn *ret);
```

```c++
typedef
struct nsEEReadData
{
    unsigned short module;
    unsigned short count;
    unsigned short address;
} nsEEReadData;
```

```c++
typedef
struct nsEEReadReturn
{
    unsigned short length;
    unsigned short *data;
} nsEEReadReturn;
```
Parameters

caller Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data Specifies an nsEEReadData input structure, described below.
ret Specifies an naEEReadReturn structure, described below.

Fields

nsEEReadData:
module: Module number of the module containing the configuration data.
count: Number of data values to read.
address: Starting address relative to the bottom of the EE storage from which to read data.

nsEEReadReturn:
length: Number of values read from the EE storage.
data: Returned data array, allocated by the function. When the data is no longer needed, the caller must use the i3free() function to free the memory.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Hardware Specifics

EDAS-2003M

There are 32 address locations in the EE storage, numbered 0 through 31. Locations 0 through 15 are used for data to be written into DACs 0 through 15. Location 30 is used for data to be written to the trim pot.

EDAS-2006M

There are 32 address locations in the EE storage, numbered 0 through 31. Locations 0 through 7 are used for data to be written into DACs 0 through 7. Locations 8 through 15 are used for data to be written into pots 0 through 8. Location 16 is used for data to be written to the trim pot.

4.2.1.35 Writing Calibration Data

The nsEEWrite() function writes calibration data stored in the EE. The address range is 0 through 31.

Function Call in C++

long nsEEWrite (unsigned int caller, nsEEWriteData *data, void *res);
typedef
struct nsEEWriteData
{
    unsigned short module;
    unsigned short count;
    unsigned short address;
    unsigned short data[2];
} nsEEWriteData;

Parameters

caller Indicates the session making the call. Pass the value returned by the call to
nsOpenSession().
data Specifies an nsEEWriteData input structure, described below.
res This parameter is reserved and should be set to NULL (0).

Fields

nsEEWriteData:
module: Module number of the module to which the configuration data is to be written.
count: Number of data values to write.
address: Starting address relative to the bottom of the EE storage to which to write data.
data: Data array to be written to the module. Be sure to allocate enough memory for the
number of values to be written.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.
Hardware Specifics

EDAS-2003M

There are 32 address locations in the EE storage, numbered 0 through 31. Locations 0 through 15 are used for data to be written into DACs 0 through 15. Location 30 is used for data to be written to the trim pot.

EDAS-2006M

There are 32 address locations in the EE storage, numbered 0 through 31. Locations 0 through 7 are used for data to be written into DACs 0 through 7. Locations 8 through 15 are used for data to be written into pots 0 through 8. Location 16 is used for data to be written to the trim pot.

4.2.1.36 Writing Data to the Calibration System

The nsDACWrite() function writes data directly to the module calibration system. If you want to change the contents of the EE storage and see the results without restarting the unit, you can use this function to update the calibration system.

Function Call in C++

long nsDACWrite (unsigned int caller, nsDACWriteData *data, void *res);

typedef
struct nsDACWriteData
{
    unsigned short module;
    unsigned short count;
    unsigned short address;
    unsigned short data[2];
} nsDACWriteData;
Parameters

caller  Indicates the session making the call. Pass the value returned by the call to nsOpenSession().
data  Specifies an nsDACWriteData input structure, described below.
res  This parameter is reserved and should be set to NULL (0).

Fields

nsDACWriteData:
module: Module number of the module to which the configuration data is to be written.
count:  Number of data values to write.
address:  Starting address relative to the bottom of the EE storage corresponding to the data to be written.
data:  Data array to be written to the module. Be sure to allocate enough memory for the number of values to be written.

Return Value Error Codes

Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure.

Hardware Specifics

EDAS-2003M

There are 32 address locations in the EE storage, numbered 0 through 31. Locations 0 through 15 are used for data to be written into DACs 0 through 15. Location 30 is used for data to be written to the trim pot.

EDAS-2006M

There are 32 address locations in the EE storage, numbered 0 through 31. Locations 0 through 7 are used for data to be written into DACs 0 through 7. Locations 8 through 15 are used for data to be written into pots 0 through 8. Location 16 is used for data to be written to the trim pot. Data values for locations 8 through 15 are written in pairs and must be supplied as a block. If only a single value is supplied, it will not be written.

1.1.75 Memory Management

CE Link functions which read data may allocate memory for returned data buffers. The application program is responsible for freeing this memory when the buffers are no longer needed. If this is not done, memory errors will result. In order to insure that the memory is returned to the heap from which it was allocated, the memory free utility should be called. The memory allocation utility used internally in the CE Link
library is provided for your convenience. If your application will allocate large amounts of memory, you should use the standard memory allocation functions provided by the run-time library (malloc()/free(), new/delete, etc.)

### 4.2.1.37 Memory Allocation

The i3malloc() function allocates memory from the default heap belonging to the current process.

**Function Call in C++**

long i3malloc (unsigned long nbytes, void **pntr);

**Parameters**

- **nbytes**: Number of bytes to allocate. Use the `sizeof` operator to determine the amount of memory to allocate for objects.
- **pntr**: Location to which the allocated pointer is to be returned. Since `pntr` is declared as `void**`, you will have to cast the address of your actual data pointer to this type.

**Return Value Error Codes**

- Zero value: Indicates the function completed successfully.
- Non-zero value: Indicates failure. Failure of memory allocation is most likely caused by not freeing memory blocks that are no longer used, or by trying to allocate very large amounts of memory.

### 4.2.1.38 Memory De-allocation

The function i3free() must be used to free memory blocks allocated by CE Link API functions, as well as memory blocks allocated explicitly using the function i3malloc(). Note that some CE Link functions allocate a single memory block which may be divided into multiple arrays whose pointers are returned to the caller. In this case only the beginning block (usually the data array), need be freed.

**Function Call in C++**

long i3free (void *pntr)

**Parameters**

- **pntr**: Pointer to the memory block to free. The block must have been allocated using the function i3malloc().

**Return Value Error Codes**
Zero value: Indicates the function completed successfully.
Non-zero value: Indicates failure. Failure is most likely due to trying to free a pointer that does not point to the beginning of a memory block allocated by `i3malloc()`.
4.3 Return Error Codes for CE Link and EDAS CE API Functions

This section contains the error codes and their descriptions for the CE Link API functions and by the Net Link API functions.

No Errors: Function Successfully Completed

0 NO_EDAS_ERROR: The function completed successfully without any errors.

System Entry Error Codes: 1 to 99

1 DRIVERS_NOT_INIT_ERR: Indicates the Net Link software drivers have not been initialized. Usually indicates the nsSWInit() function was not called.

20 MAX_CLIENTS_EXCEEDED_ERR: Multiple Net Link client support error. The maximum number of supported clients has been exceeded.

21 CLIENT_NOT_REGISTERED_ERR: Multiple Net Link client support error. You must register your application instance with the RegisterClient() function before making any driver function calls. Note: When your application instance is through using the driver functions, you should unregister the application instance by using the UnregisterClient() function.

22 CLIENT_DATA_ALLOC_ERR: This is a Net Link multiple client support error. The client data could not be allocated usually because the global memory pool has been exhausted.

23 CLIENT_DATA_FREE_ERR: Indicates an internal Net Link error. The client data could not be freed.

30 IO_LOCK_ERR: This is an I/O lock error. The specified I/O resource has been locked by another client and is unavailable until that client unlocks the resource.

31 IO_UNLOCK_ERR: This is an I/O lock error. An attempt was made to unlock an I/O resource by a client that did not originally lock the resource.

32 IO_LOCK_TYPE_NOT_SUPPORTED_ERR: This is an I/O lock error. Resource locking is not supported for the specified I/O type.

50 SED_NULL_POINTER_ERR: Indicates an internal error.

51 SED_INTERRUPT_ERR: Indicates an EDAS CE interrupt error. A required interrupt could not be initialized for the module, or there is an interrupt bus error or interrupt timeout. This error
may be returned after the initialization process returned
SED_MOD_FAIL_ERR.

SED_THREAD_ERR: Indicates an EDAS CE thread creation error.

**System Entry Warning Error Codes: -1 to -99**

-5 STARTUP_ERR: This is a CE Link start up error. The nsSWDeinit() function was called before calling nsHWInit(). Initialization functions are always called first.

-6 SED_MOD_FAIL_ERR: Indicates an EDAS CE module failure. Hardware initialization of one or more modules has failed. This error is returned from the first Net Link API call that accesses the EDAS CE, and from the CE Link nsHWInit() call.

-7 SED_GEOG_FAIL_ERR: Indicates an EDAS CE geographical mapping failure for CE Link functions. Data structures are automatically released. The application program may repeat the call to nsHWInit() to see if a second call works.

**Lookup Function Error Codes: 200 to 299**

202 SED_IOTYPE_NOT_SUPP_ERR: This is an internal error, indicating that a software module does not support an expected I/O type.

203 SED_HWRID_NOT_SUPP_ERR: This is an internal error. A software module does not support an expected hardware module ID.

204 SED_INT_OPERATION_ERR: This is an internal error. A software module does not support an expected internal operation.

205 SED_INT_FUNCTION_ERR: This is an internal error. A software module does not support an expected API function.

208 SED_MOD_CHANNEL_ERR: This is an EDAS CE channel lookup failure. The module and channel number specified do not exist.

209 SED_MOD_FUNCTION_ERR: This is an EDAS CE function lookup failure. The API function is not supported for the hardware module.

210 SED_NO_CHANNEL_ERR: This is an EDAS CE illegal channel. The channel specified does not exist.

**Lookup Function Warning Error Codes: -200 to -299**

-203 EDAS_ID_ERR: This is an invalid EDAS ID. An Ethernet DAS not supported by this software has been installed.

-204 EDAS_SHUTDOWN: Shutdown command returned by the CE Link function nsCloseSession(). Your application should terminate in order to unload the EDAS CE DLLs.
Memory Allocation Error Codes: 300 to 399

300 MEMORY_ALLOC_ERR: This is a memory allocation error. Insufficient memory is available for the system data.

301 MEMORY_FREE_ERR: This is an internal memory deallocation error. An attempt was made to de-allocate memory outside of the system data block.

302 MALLOC_HOST_ERR: This is a Net Link host language memory allocation error. This error might occur if there is not enough system memory to satisfy a buffer allocation request. If possible, use a smaller buffer.

303 MALLOC_HOST_FREE_ERR: This is a Net Link host language memory free error.

320 MALLOC_WINDOWS_LOCK_ERR: This is a Windows GlobalLock error in Net Link.

321 MALLOC_WINDOWS_HANDLE_ERR: This is a Windows GlobalHandle error in Net Link.

322 MALLOC_WINDOWS_FREE_ERR: This is a Windows GlobalFree error in Net Link.

323 MALLOC_WINDOWS_REALLOC_ERR: This is a Windows GlobalReAlloc error in Net Link.

325 MALLOC_WINDOWS_PAGE_LOCK_ERR: This is a Windows GlobalPageLock error in Net Link.

330 GLBALLOC_ALLOC_ERR: There is insufficient memory for allocation in the shared segment.

331 GLBFREE_HEADER_ERR: The memory allocation header is invalid (memory overwritten).

332 GLBFREE_LINK_ERR: Link is not in the list.

333 FILE_ALLOC_ERR: Unable to map a file for buffer allocation.

334 MAP_ALLOC_ERR: Unable to view a file for buffer allocation.

335 FILE_REMAP_ERR: Unable to remap a file allocation.

336 FILE_NULL_ERR: Unable to free a null pointer.

337 FILEFREE_LINK_ERR: File buffer pointer was not found.

File Errors Codes: 400 to 499

410 WIN_FILE_OPEN_ERR: This is a Windows file open error in Net Link. This error might occur if the drivers do not find a thermocouple table.

411 WIN_FILE_CLOSE_ERR: This is a Windows file close error in Net Link.
WIN_FILE_READ_ERR: This is a Windows file read error in Net Link.

UNIX_FILE_OPEN_ERR: This is a UNIX file open error in Net Link. This error might occur if the drivers do not find a thermocouple table.

UNIX_FILE_CLOSE_ERR: This is a UNIX file close error in Net Link.

UNIX_FILE_READ_ERR: This is a UNIX file read error in Net Link.

SED_FILE_ERR: Indicates an EDAS CE error opening a DLL module file or an input or output data file.

Lock Error Codes: 550 to 599

MAX_SEGMENT_LOCK_FUNCTION_ERR: This is an internal error in Net Link. The maximum number of driver code segments that can be locked has been exceeded. Contact the factory.

Network Error Codes: 800 to 899

SOCKETS_VERSION_ERR: The WINSOCK.DLL installed on the host computer is not compatible with the Net Link drivers.

SOCKETS_CONFLICT_ERR: There is a conflict in the error code assignment between the drivers and the NetLink host TCP/IP implementation. Contact the factory.

SESSION_CLOSED_ERR: The EDAS server has closed the communication link, or socket, with the Net Link client.

COMMAND_PENDING_ERR: The EDAS Net Link client attempted to issue another command to the EDAS, on the same socket, before the previous command completed.

LISTEN_PENDING_ERR: The EDAS attempted to initiate a new listen process before the previous NetLink listen process completed.

INVALID_COMMAND_ERR: The EDAS received an invalid, or unsupported, Net Link command. Contact the factory.

SOCKETS_GETLASTERROR_ERR: A critical error occurred in the sockets GetLastError() function. Contact the factory.

SESSION_HANDLE_ERR: The specified Net Link session handle is invalid.

INVALID_SERVER_STATE_ERR: The EDAS CE Net Link server (suprcatE.exe) has entered an invalid command state. Contact the factory.

INVALID_CLIENT_STATE_ERR: The EDAS CE Net Link client has entered an invalid command state. Contact the factory.
CLIENT_RX_BUFFER_SIZE_ERR: The EDAS CE Net Link client's receive buffer is too small to accommodate the data returned by the EDAS CE. Contact the factory.

CLIENT_RX_BUFFER_NULL_ERR: The EDAS CE Net Link client's receive buffer is NULL and therefore, the client can not receive the data returned by the EDAS CE. Contact the factory.

SEND_LENGTH_ZERO_ERR: Indicates an internal error. The EDAS CE Net Link client/server attempted to send a packet but specified a send length of zero.

SEND_BUFFER_NULL_ERR: Indicates an internal error. The EDAS Net Link client/server attempted to send a packet but specified a NULL send buffer pointer.

SEND_SELECT_ERR: An unhandled error was returned by the sockets select() function call while attempting to send a Net Link packet. Contact the factory.

RECV_LENGTH_ZERO_ERR: This is an internal error. The EDAS Net Link client/server attempted to receive a packet but specified a receive length of zero.

RECV_BUFFER_NULL_ERR: This is an internal error. The EDAS Net Link client/server attempted to receive a packet but specified a NULL receive buffer pointer.

RECV_SELECT_ERR: An unhandled error was returned by the sockets select() function call while attempting to receive a Net Link packet. Contact the factory.

LISTEN_SELECT_ERR: An unhandled error was returned by the sockets select() function call while listening for a new Net Link client. Contact the factory.

EXCEPT_SELECT_ERR: An unhandled error was returned by the sockets select() function call while checking for exceptions on the Net Link socket. Contact the factory.

Network Warning Error Codes: -800 to -849

-800 LISTEN_PENDING_WARN: The current listen command is still pending (i.e. the EDAS CE NetLink server is still waiting for a client to call). You must wait for the current listen command to complete before listening again on the same socket.

-801 COMMAND_PENDING_WARN: The current EDAS Net Link command is still pending. You must wait for the current command to complete before issuing another command on the same socket.

-802 SEND_PENDING_WARN: The current Net Link send request is still pending. You must wait for the current send request to complete before sending another packet on the same socket.
-803 RECV_PENDING_WARN: The current Net Link receive request is still pending. You must wait for the current receive request to complete before receiving another packet on the same socket.

**Analog Input Error Codes: 2000 to 2999**

2005 AI_CHANNEL_ERR: The specified analog input channel is invalid.
2006 AI_DIFF_CHANNEL_ERR: The specified analog input channel is invalid or can not be configured for differential input.
2007 AI_SE_CHANNEL_ERR: The specified analog input channel is invalid or can not be configured for single-ended input.
2010 AI_GAIN_ERR: The specified gain value for an analog input channel is invalid.
2011 AI_RANGE_ERR: The specified range value for an analog input channel is invalid.
2014 AI_ZEROCHANNEL_ERR: This is an illegal auto-zero reference channel.
2015 AI_DATA_LENGTH_ERR: Unexpected returned data length, from old-style Net Link call to EDAS CE. Use new-style calls for multiple modules.
2503 AI_INDEX_ERR: An EDAS CE EE or DAC number error.

**Analog Input Warning Error Codes: -2000 to -2999**

-2000 AI_CHAN_CONFIG_ERR: The specified analog input channel has not been configured, therefore, default parameters were used for the reading. The default configuration is single-ended, 0-10 V, no auto-zero correction.

**Analog Output Error Codes: 3000 to 3999**

3001 AO_CHANNEL_ERR: The specified analog output channel is invalid.
3002 AO_RANGE_ERR: The specified range value for an analog output channel is invalid.
3003 AO_NOT_CONFIGURED_ERR: The specified analog output channel must be configured before writing to the channel.
3006 AO_INDEX_ERR: An EDAS CE EE or DAC number error.

**Digital I/O Error Codes: 7000 to 7999**

7000 DIO_NOT_INITIALIZED_ERR: The specified digital I/O port was never successfully initialized or does not exist.
7001 DIO_PORT_ERR: The specified digital I/O port is invalid.
7002 DIO_PORT_BIT_ERR: The specified digital I/O port bit is invalid. Valid values are 0 - 7 (0 = LSB, 7 = MSB).
7003  DIO_NO_WRITE_ERR: Writing to digital I/O port configured for input is not permitted.
7004  DIO_NO_HANDSHAKE_ERR: The specified digital I/O port does not have handshake capability.
7009  DIO_INPUT_ONLY_ERR: The specified digital I/O port can only be configured for input.
7010  DIO_OUTPUT_ONLY_ERR: The specified digital I/O port can only be configured for output.
7012  DIO_DATA_LENGTH_ERR: For the EDAS CE API: Unexpected returned data length, from an old-style call. Use new-style functions for multiple modules.
7013  DIO_STATUS_ERR: For EDAS CE: In-progress bit error (hardware error).
7110  DIO_PORT_BIT_MODE_ERR: The specified digital I/O port bit mode is invalid.
7111  DIO_PORT_BIT_DATA_ERR: The specified digital I/O port bit configuration data is invalid.
7113  DIO_PORT_BIT_CONF_ERR: The specified digital I/O port bit has not been configured for the specified function.
7114  DIO_PORT_BIT_ENAB_ERR: The specified digital I/O port bit has not been enabled.

Rate Generator Error Codes: 8000 to 8999

8002  RG_MODE_ERR: The specified rate generator mode is invalid.
8003  RG_COUNT_ERR: Invalid count values cause this error. The count value must be greater than 1 and less than or equal to $2^{24}-1$.
8004  RG_NOT_CONFIGURED_ERR: The specified rate generator has not been configured.
8006  RG_NOT_DISABLED_ERR: The rate generator must be disabled before being reconfigured.

Thermocouple Error codes 11000 to 11999

11000 TC_NO_DATA_ERR: Internal Net Link error. There is no data for the specified thermocouple type.
11001 TC_HANDLE_ERR: Internal Net Link error. The thermocouple dictionary has been assigned an invalid handle type.
11002 TC_TABLE_NOT_LOADED_ERR: Internal Net Link error. The thermocouple table for the specified type has not been loaded.
11003 TC_TABLE_LOADED_ERR: Internal Net Link error. The specified thermocouple table has already been loaded.
11004 TC_TABHDR_READ_ERR: (Net Link) Thermocouple table header read error. The thermocouple table files may have been corrupted. Recopy the files from the original disks.

11005 TC_TABLE_READ_ERR: (Net Link) Thermocouple table read error. The thermocouple table files may have been corrupted. Recopy the files from the original disks.

11006 TC_DICTIONARY_FULL_ERR: Internal Net Link error. The thermocouple dictionary is full.

11007 TC_VOLTAGE_OUT_OF_RANGE_ERR: (Net Link) The thermocouple voltage is out of range.

11008 TC_CJCTEMP_OUT_OF_RANGE_ERR: (Net Link) The thermocouple CJC temperature is out of range.

11100 TC_5B_AD_NOT_SE_ERR: (Net Link) The non-linearized 5B thermocouple modules must be used with an A/D that has single-ended inputs.

11101 TC_5B_CJC_CHAN_ERR: (Net Link) The CJC channel parameter passed to the TCMeasure() function must be set to TC_NO_CJC. The cold-junction compensation is built into the non-linearized 5B thermocouple modules.

11102 TC_5B_CJC_TEMP_ERR: (Net Link) The CJC temperature parameter passed to the TCLinearize() function must be set to 0.0. The cold-junction compensation is built into the non-linearized 5B thermocouple modules.

**Thermocouple Warning Codes: -11000 to -11999**

-11000 TC_FILE_OPEN_ERR: (Net Link) One of the thermocouple files (TYPEJ.TC, TYPEK.TC, TYPET.TC, TYPEJ5B.TC, TYPEK5B.TC or TYPE5B.TC) could not be found. If you are using the thermocouple functions, make sure that the files corresponding to the thermocouple types you are using are in the working directory of your application.

**Paced Acquisition Error Codes: 14000 to 14999**

14000 HS_NO_DATA_ERR: There is no data for the specified paced data acquisition process.

14001 HS_HANDLE_ERR: The specified paced data acquisition handle is invalid.

14002 HS_NOT_CONFIGURED_ERR: The specified paced data acquisition process has not been successfully configured.

14003 HS_START_MODE_ERR: The specified paced data acquisition start mode is invalid.

14006 HS_STOP_MODE_ERR: The specified paced data acquisition stop mode is invalid.
14010  HS_BUFFER_SIZE_ERR: (NetLink) The size of the specified buffer is either too small for an input process (i.e. HS buffer cannot accommodate the cluster count previously specified) or too large for an output process (i.e. on-board FIFO cannot accommodate the entire buffer).

14011  HS_ZERO_CLUSTER_COUNT_ERR: A count of zero was specified in the cluster count parameter of the nsAIHSConfigureList() function, or the terminal count parameter of the nsAIHSEnable() function.

14012  HS_PACER_SOURCE_ERR: The specified pacer source is invalid or not supported.

14013  HS_TRIGGER_SOURCE_ERR: The specified trigger source is invalid, not supported, or inconsistent with the specified start/stop mode.

14014  HS_MODULE_ERR: All channels specified must lie on the same analog input module.

14015  HS_ZEROCHANNEL_ERR: This is an illegal auto-zero reference channel. It is not allowed for paced data acquisition functions.

14016  HS_RUNNING_ERR: Configuration functions and free can not be performed while the paced data acquisition process is running.

14017  HS_NO_HANDLE_ERR: There are no more paced data acquisition handles available.

14018  HS_SED_INTERNAL_ERR: This is an EDAS CE paced data acquisition function lookup failure. Contact the factory.

14019  HS_OWNER_ERR: A paced data acquisition using this module has already been configured by another user.

1402  HS_INVALID_LIST_ERR: The paced data acquisition channel list specified is invalid. The wrong channel sequence occurred, or gain and range restrictions were violated.

1403  HS_CLUSTERSIZE_ERR: The specified paced data acquisition clustersize specified in the function call is different from the clustersize that was configured.

**Alarm Error Codes: 16000 to 16999**

16001  ALARM_HANDLE_ERR: The specified alarm handle is invalid.

16003  ALARM_TYPE_ERR: The specified alarm type is invalid.

16004  ALARM_MODE_ERR: The specified alarm mode is invalid.

16005  ALARM_INPUT_NULL_ERR: (Net Link) The alarm input event can not be NULL.

16006  ALARM_INTERNAL_ERR: An internal error was detected while manipulating an alarm. Contact the factory.
16007 ALARM_SYSTEM_ERR: An error creating alarm system structures occurred.

16008 ALARM_CONFIGURE_ERR: This is an internal error, the alarm configuration data is invalid.

16010 ALARM_INPUT_IOTYPE_ERR: The alarm input I/O type is invalid.

16011 ALARM_INPUT_SUPPORT_ERR: The alarm input channel is not supported.

16020 ALARM_OUTPUT_IOTYPE_ERR: The alarm output I/O type is invalid.

16021 ALARM_OUTPUT_SUPPORT_ERR: The alarm output channel is not supported.

16030 ALARM_XFER_MODE_ERR: The Alarm transfer mode is invalid.

16031 ALARM_XFER_MESSAGE_ERR: (NetLink) The Alarm message length and pointer parameters are not compatible (i.e. the message length is non-zero, but the message pointer is zero, or the message length is zero, but the message pointer is non-zero.

16032 ALARM_XFER_FCN_ERR: The alarm transfer function for CE Link is not provided, and a network transfer option is requested.

16033 ALARM_LOG_FILE_ERR: The alarm log file can not be opened or written.

Quadrature Error Codes: 17000 to 17999

17002 QUAD_MODE_ERR: The mode specified for the quadrature channel is not valid.

17010 QUAD_INPUT_BIT_MODE_ERR: The general-purpose input bit mode specified is not valid.

17011 QUAD_OUTPUT_BIT_MODE_ERR: The general-purpose output bit mode specified is not valid.

Timestamp Error Codes: 19000 to 19999

19000 Error in converting the timestamp data to system time.

19001 Error in converting the timestamp data to string date and time. This error occurs in time conversion.

19002 Error in converting the timestamp data to string date and time. This error occurs in date conversion.
**Exception Error Codes: 22000 to 22999**

These error codes indicate that an exception occurred in the Net Link command processor (**suprcatE.exe**) on the EDAS CE. Exceptions are not handled for CE Link applications: you must provide your own exception handling.

22002 **EXCEPTION_DATATYPE_MISALIGNMENT**: The thread tried to read or write data that is misaligned on hardware that does not provide alignment. For example, 16-bit values must be aligned on 2-byte boundaries; 32-bit values on 4-byte boundaries, and so on.

22003 **EXCEPTION_BREAKPOINT**: A breakpoint was encountered.

22004 **EXCEPTION_SINGLE_STEP**: A trace trap or other single-instruction mechanism signaled that one instruction has been executed.

22005 **EXCEPTION_ACCESS_VIOLATION**: The thread tried to read from, or write to, a virtual address for which it does not have the appropriate access.

22006 **EXCEPTION_IN_PAGE_ERROR**: The thread tried to access a page that was not present, and the system was unable to load the page. For example, this exception might occur if a network connection is lost while running a program over the network.

22029 **EXCEPTION_ILLEGAL_INSTRUCTION**: The thread tried to execute an invalid instruction.

22037 **EXCEPTION_NONCONTINUABLE_EXCEPTION**: The thread tried to continue execution after a noncontrollable exception occurred.

22038 **EXCEPTION_INVALID_DISPOSITION**: An exception handler returned an invalid disposition to the exception dispatcher. Programmers using a high-level language such as C should never encounter this exception.

22100 **EXCEPTION_ARRAY_BOUNDS_EXCEEDED**: The thread tried to access an array element that is out of bounds and the underlying hardware supports bounds checking.

22140 **EXCEPTION_FLT_DENORMAL_OPERAND**: One of the operands in a floating-point operation is denormal. A denormal value is one that is too small to represent as a standard floating-point value.

22141 **EXCEPTION_FLT_DIVIDE_BY_ZERO**: The thread tried to divide a floating-point value by a floating-point divisor of zero.

22143 **EXCEPTION_FLT_INEXACT_RESULT**: The result of a floating-point operation cannot be represented exactly as a decimal fraction.

22144 **EXCEPTION_FLT_INVALID_OPERATION**: This exception represents any floating-point exception not included in this list.
22145 EXCEPTION_FLT_OVERFLOW: The exponent of a floating-point operation is greater than the magnitude allowed by the corresponding type.

22146 EXCEPTION_FLT_STACK_CHECK: The stack overflowed or underflowed as the result of a floating-point operation.

22147 EXCEPTION_FLT_UNDERFLOW: The exponent of a floating-point operation is less than the magnitude allowed by the corresponding type.

22148 EXCEPTION_INT_DIVIDE_BY_ZERO: The thread tried to divide an integer value by an integer divisor of zero.

22149 EXCEPTION_INT_OVERFLOW: The result of an integer operation caused a carry out of the most significant bit of the result.

22150 EXCEPTION_PRIV_INSTRUCTION: The thread tried to execute an instruction whose operation is not allowed in the current machine mode.

22253 EXCEPTION_STACK_OVERFLOW: The thread used up its stack.

**Winsock Error Codes: 28000 to 30000**

These error codes indicate that a network error occurred in the Net Link command processor (suprcatE.exe) on the EDAS CE.

28004 WSAEINTR: A blocking operation was interrupted by a call to WSACancelBlockingCall().

28009 WSAEBADF: The file handle supplied is not valid.

28013 WSAEACCES: An attempt was made to access a socket in a way forbidden by its access permissions.

28014 WSAEFAULT: The system detected an invalid pointer address in attempting to use a pointer argument of a call.

28022 WSAEINVAL: Some invalid argument was supplied.

28024 WSAEMFILE: Too many open sockets.

28035 WSAEWOULDBLOCK: A non-blocking socket operation could not be completed immediately.

28036 WSAEINPROGRESS: A blocking operation is currently executing.

28037 WSAEALREADY: An operation was attempted on a non-blocking socket that already had an operation in progress.

28038 WSAENOTSOCK: An operation was attempted on something that is not a socket.

28039 WSAEDESTADDRREQ: A required address was omitted from an operation on a socket.

28040 WSAEMSGSIZE: A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the
buffer used to receive a datagram into was smaller than the datagram itself.

28041 WSAEPROTOTYPE: A protocol was specified in the socket() function call that does not support the semantics of the socket type requested.

28042 WSAENOPROTOOPT: An unknown, invalid, or unsupported option or level was specified in a getsockopt() or setsockopt() call.

28043 WSAEPROTONOSUPPORT: The requested protocol has not been configured into the system, or implementation for it does not exist.

28044 WSAESOCKTNOSUPPORT: The support for the specified socket type does not exist in this address family.

28045 WSAEOPNOTSUPP: The attempted operation is not supported for the type of object referenced.

28046 WSAEPPNOSUPPORT: The protocol family has not been configured into the system or implementation for it does not exist.

28047 WSAEAFNOSUPPORT: An address incompatible with the requested protocol was used.

28048 WSAEADDRINUSE: Only one usage of each socket address (protocol/network address/port) is normally permitted.

28049 WSAEADDRNOTAVAIL: The requested address is not valid in its context.

28050 WSAENETDOWN: A socket operation encountered a dead network.

28051 WSAENETUNREACH: A socket operation was attempted to an unreachable network.

28052 WSAENETRESET: The connection has been broken due to keep-alive activity detecting a failure while the operation was in progress.

28053 WSAECONNABORTED: An established connection was aborted by the software in the host machine.

28054 WSAECONNRESET: An existing connection was forcibly closed by the remote host.

28055 WSAENOBUFS: An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

28056 WSAEISCONN: A connect request was made on an already connected socket.

28057 WSAENOTCONN: A request to send or receive data was disallowed because the socket is not connected. An address was not supplied when sending on a datagram socket using sendto().
WSAESHUTDOWN: A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown() call.

WSAETOOMANYREFS: There are too many references to a kernel object.

WSAETIMEDOUT: A connection attempt failed because the connected party did not properly respond after a period of time, or an established connection failed because the connected host failed to respond.

WSAECNNREFUSED: The target machine actively refused to make a connection.

WSAELoop: Unable to translate a name.

WSAENAMETOOLONG: Name component or name was too long.

WSAEHOSTDOWN: A socket operation failed because the destination host was down.

WSAEHOSTUNREACH: A socket operation was attempted to an unreachable host.

WSAENOTEMPTY: Cannot remove a directory that is not empty.

WSAEPROCLIM: A Windows Sockets implementation may have a limit on the number of applications that may use it simultaneously.

WSAEOUSERS: Ran out of Quota.

WSAEDQUOT: Ran out of disk Quota.

WSAESTALE: File handle reference is no longer available.

WSAEREMOTE: Item is not available locally.

WSASYSNOTREADDY: WSASStartup() cannot function at this time because the underlying system it uses to provide network services is currently unavailable.

WSAERNOTSUPPORTED: The Windows Sockets version requested is not supported.

WSANOTINITIALISED: Either the application has not called WSASStartup(), or WSASStartup() failed.

WSAEDISCON: Returned by WSARcv(), WSARcvFrom() indicates the remote party has initiated a graceful shutdown sequence.

WSAENOMORE: WSALookupServiceNext can not return any more results.

WSAECANCELLED: A call to WSALookupServiceEnd was made while this call was still processing. The call has been canceled.

WSAENVALIDPROCTABLE: The procedure call table is invalid.

WSAENVALIDPROVIDER: The requested service provider is invalid.
28106 WSAEPROVIDERFAILEDINIT: The requested service provider could not be loaded or initialized.
28107 WSASYSCALLFAILURE: This error is returned when there is a failure of a system call that should never fail.
28108 WSA_SERVICE_NOT_FOUND: No such service is known. The service cannot be found in the specified name space.
28109 WSATYPE_NOT_FOUND: The specified class was not found.
28110 WSA_E_NO_MORE: No more results can be returned by WSALookupServiceNext.
28111 WSA_E_CANCELED: The call has been canceled because a call to WSALookupServiceEnd was made while this call was still processing.
28112 WSA_E_REFUSED: A database query failed because it was actively refused.
29001 WSAHOST_NOT_FOUND: No such host is known.
29002 WSA_TRY_AGAIN: This is usually a temporary error during hostname resolution and means that the local server did not receive a response from an authoritative server.
29003 WSANO_RECOVERY: A non-recoverable error occurred during a database lookup.
29004 WSANO_DATA: The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for.
29005 WSA_QOS_RECEIVERS: At least one -Reserve has arrived.
29006 WSA_QOS_SENDERS: At least one Path has arrived.
29007 WSA_QOS_NO_SENDERS: There are no senders.
29008 WSA_QOS_NO_RECEIVERS: There are no receivers.
29009 WSA_QOS_REQUEST_CONFIRMED: Reserve has been confirmed.
29010 WSA_QOS_ADMISSION_FAILURE: Error due to lack of resources.
29011 WSA_QOS_POLICY_FAILURE: Rejected for administrative reasons – bad credentials.
29012 WSA_QOS_BAD_STYLE: Unknown or conflicting style.
29013 WSA_QOS_BAD_OBJECT: A problem occurred with part of the filterspec or provider specific buffer.
29014 WSA_QOS_TRAFFIC_CTRL_ERROR: Problem with part of the flowspec occurred.
29015 WSA_QOS_GENERIC_ERROR: A general QOS error occurred.
This page intentionally left blank.
alarms, 133
analog input module, 11
  channel scanner, 14
  current input configuration, 12
  pacing, 14
  rate generator, 14
  specifications, 14
  switch settings, 12
  thermocouple input configuration, 12
  voltage input configuration, 12
analog output module, 20
  specifications, 20
application generating, 72
applications
  loading, 79
  writing, 72
base unit, 2
battery
  real-time clock, 4
baud rate
  reading, 41
  reconfiguring, 43
bit numbers, 85
calibration data, 147
CE Link
  debugging applications, 82
  DLLs on the EDAS CE, 85
  examples
    see CE Link API examples, 86
  hints and tips, 81
  path for files, 85
  priority, 86
  registry entries, 85
  required files, 85
CE Link API
  alarms, 133
    configuration, 134
    data options for transfer, 141
    digital input count, 136
    digital input latch, 136
    disable transfer, 141
    enabling and disabling, 137
    Ethernet transfer, 141
    freeing resources, 143
    handle, 137
    IP address for transfer, 141
    local response, 133
    log file name, 141
    logging options, 139
    modes, 136
  analog output, 105
    configuration, 105
    output data organization, 107
    range, 105
    writing data, 106
  application deinitialization, 88
  bit numbers, 85
  channel configuration, 125
  channel numbers, 85
  channel specification, 85
  closing a session, 87
  counter, 125
counter input data organization, 130
enabling and disabling, 128
reading channels, 129
date and time stamping, 83
digital I/O, 107
  input/output flag, 109
  output port data organization, 113
  port configuration, 108
  port input data organization, 111
  reading ports, 110
  writing digital ports, 112
digital I/O bit, 113
  configuration, 113
  count input, 115, 118
  delay output, 116, 118
  disabling channels, 117
  enabling channels, 117
  latch input, 115, 118
  modes, 115
  normal input, 115
  normal output, 116
  output data organization, 122
  portbit masks, 113
  pulse output, 116, 118
  reading input channels, 119
  reset on read, 120
  return data organization, 120
  setting output state, 121
  square wave output, 116, 119
  writing output channels, 121
delay output, 116, 118
digital I/O bit write, 122
digital I/O bit, 113
digital I/O, 107
examples
  see CE Link API examples, 86
  function call basics, 82
  function call format, 81
  hardware initialization, 86
  initialization functions, 86
  memory allocation, 83
  memory deallocation, 83
  module numbers, 85
  opening a session, 87
  port numbers, 85
  quadrature, 125
    channel configuration, 125
    configuring interrupts, 131
    enabling and disabling, 128
    quadrature input data organization, 130
    reading channels, 129
  rate generator, 123
    enabling and disabling, 124
    frequency configuration, 123
  returned data arrays, 83
  system
    calibration, 147
    I/O locking, 145
locking I/O channels, 145
memory allocation, 152
memory de-allocation, 152
memory management, 152
programming the calibration system, 150
reading calibration data, 147
time stamping, 145
version information, 144
writing calibration data, 148
system functions, 143
  system information, 144
time stamping, 83
CE Link API examples
  analog input configuration, 90
  converting current input to mA, 93
digital I/O bit write, 122
reading analog input data, 94
CE Link API functions
  i3free(), 152
  i3malloc(), 152
  nsAIConfigureList(), 88
  nsAIHSConfigureList(), 95
  nsAIHSEnable(), 98
  nsAIHSRead(), 99
  nsAIHSStatus(), 103
  nsAIRead(), 91
  nsALARMConfigure(), 134
  nsALARMEnable(), 137
  nsALARMFree(), 143
  nsALARMStatus(), 138
  nsALARMTransferOptions(), 139
  nsAOConfigureList(), 105
  nsAOWrite(), 106
  nsCloseSession(), 87
  nsDACWrite(), 150
  nsDIOBITConfigureList(), 113
  nsDIOBITEnable(), 117
  nsDIOBITRead(), 119
  nsDIOBITWrite(), 121
  nsDIOConfigureList(), 108
  nsDIORead(), 110
  nsDIOWrite(), 112
  nsEERead(), 147
  nsEEWrite(), 148
  nsHWInit(), 86
  nsIOLock(), 145
  nsOpenSession(), 87
  nsQUADConfigure(), 125
  nsQUADEnable(), 128
  nsQUADInterrupt(), 131
  nsQUADRde(), 129
  nsRGConfigure(), 123
  nsRGENable(), 124
Index 180

nsSWDeinit(), 88
nsSYSInquire(), 144
nsVERInquire(), 144
CEFlush utility, 62
overview, 34
channel numbers, 85
channel scanner, 14
COM port mappings, 23
COM ports
   SerialSocket utility, 61
compact flash, 4
configuration
   for downloading program files, 65, 66
connection troubleshooting, 71
connector
   power, 7
current input configuration, 12
custom applications
   loading, 79
date and time stamping, 83
debug workstation
   configuration, 57
debug workstations
   configuring, 37
   reading, 42
   reconfiguring, 43
DebugLauncher utility, 57
overview, 34
debugworkstations, 58
debugworkstation.txt, 57
debugworkstations.txt File Format, 57
DHCP enable flag
   configuring, 37
   reading, 41
   reconfiguring, 42
digital I/O module, 8
   specifications, 9
digital input module, 16
   specifications, 17
digital output module, 18
   specifications, 19
digital output triac module, 26
   specifications, 27
DNS addresses
   configuring, 37
   reading, 41
   reconfiguring, 42
downloading files
   configuring EDAS CE unit, 65
DRAM, 4
EDAS CE SDK, 63
installation, 64
uninstalling, 65
EDAS-2002M
digital I/O bit, 116
digital port configuration, 109
digital port input, 111
digital port output, 113
EDAS-2003M
   analog input configuration, 91
   paced analog input configuration, 98
   rate generator, 124, 125
   reading analog input, 94
   reading calibration data, 148
   writing calibration DACs, 151
   writing calibration data, 150
EDAS-2004M
   digital port configuration, 109
digital port input, 111
EDAS-2005M
   digital port configuration, 109
digital port input, 111
digital port output, 113
EDAS-2006M
   analog output configuration, 106
   reading calibration data, 148
   writing analog output data, 107
   writing calibration DACs, 151
   writing calibration data, 150
EDAS-2010M
   digital port configuration, 109
digital port input, 111
digital port output, 113
EDAS-2011M
   digital port configuration, 109
digital port input, 111
digital port output, 113
EDAS-2015M
   quadrature, 129
   quadrature read, 131
EE data storage, 147
eMbedded Visual Tools, 63
embedded web server, 45
ethernet, 3
   Telnet connection, 39
ethernet address, 37
files
   required for CE Link, 85
flow control (serial)
   reading, 41
   reconfiguring, 43
folders
   protected in remote manager, 53
gateway address
   configuring, 37
   reading, 41
   reconfiguring, 42
hardware pacing, 14
header files, 85
help
  Remote Manager, 47
HyperTerminal
  withLCmdSet, 39
  withMonitorCE, 35
  withSerialSocket utility, 61
ID code
  module ID, 145
  unit ID, 86, 144
installation
  EDAS CE SDK, 64
  Microsoft eMbedded Visual Tools, 64
IP address
  alarm transfer destination, 141
  configuring, 37
  debug workstation, 57
  reading, 41
  reconfiguring, 42
LCmdSet utility, 39
  change commands, 42
  operational commands, 44
  overview, 33
  set commands, 40
  show commands, 40
library
  CE Link, 81
loading custom programs, 79
local time
  reading, 41
  setting, 43
Microsoft eMbedded Visual Tools
  remote file viewer, 79
  troubleshooting a connection, 71
module numbers, 85
modules
  analog input, 11
  analog output, 20
  base unit, 2
  digital I/O, 8
  digital input, 16
  digital output, 18
  digital output triac, 26
  power supply, 6
  quadrature, 28
  relay output, 24
  serial port, 22
MonitorCE utility
  overview, 33
  program commands, 37
  required tools, 35
  serial communications program, 35
  starting a session, 36
mounting options
  wall mount, 30
  NV-RAM option, 5
  parity (serial)
    reading, 41
    reconfiguring, 43
  password, 43
  path for CE Link files, 85
  port numbers, 85
  portbit masks, 113
  power connector, 7
  power supply, 6
    specifications, 7
  process list, 42
  program files
    installation of SDK, 64
  programs
    sample applications, 79
    writing custom applications, 72
  project
    generating, 72
  quadrature module, 28
  specifications, 29
  rate generator, 14
  real-time clock
    battery, 4
    setting, 38
    time synchronization, 59
  reboot from Telnet session, 44
registry
  CE Link entries, 85
  saving settings, 62
  SerialSocket utility configuration, 61
  relay output module, 24
  specifications, 25
remote access to serial ports, 61
remote file viewer, 79
Remote Manager
  Application Manager, 50
  Developer’s Guide, 56
  errors, troubleshooting, 46
  File Manager, 51
  help links, 47
  home page, 47
  management, 49
    capabilities, other, 47
  Module Manager, 52
  password protection, 46
  security manager
    folders, allowing access to, 53
  System Manager, 53
  terminal information, 48
  Update Manager, 56
  using, 46
  WebDevice
server system configuration, 55
Remote Manager utility, 45
reset timer, 5
run program from Telnet session, 44
sample programs, 79
saving registry settings, 62
SDK for EDAS CE, 63
installation, 64
uninstalling, 65
security
Remote Manager, 53
serial communications program, 35
HyperTerminal, 35
serial module
specifications, 22
serial port module, 22
COM port mappings, 23
serial ports, 4
remote computer access, 61
SerialSocket utility, 61
overview, 34
registry storage, 61
Setting up Communications to the EDAS CE, 65
Setting up your Development Computer, 63
software development kit, 64
software license agreement, 64
software pacing, 14
specifications, 3
analog input module, 14
analog output module, 20
digital I/O module, 9
digital input module, 17
digital output module, 19
digital output triac module, 27
power supply, 7
relay output module, 25
serial module, 22
start program from Telnet session, 44
startup directory
DebugLauncher, 57
LCmdSet, 39
stop bits (serial)
reading, 41
reconfiguring, 43
stop program from Telnet session, 44
subnet mask
configuring, 37
reading, 41
reconfiguring, 42
system requirements
development computer, 63
TCP alarm transfer mode, 141
TCP/IP port
debug workstation, 57
TCP/IP sockets
used by SerialSocket utility, 61
Telnet, 39
thermocouple input configuration, 12
time stamping, 83, 145
time synchronization, 59
time zone
reading, 41
setting, 43
TimeSync utility, 59
command line arguments, 59
overview, 34
triac module, 26
specifications, 27
troubleshooting
connection to eMbedded Visual Tools, 71
UDP alarm transfer mode, 141
uninstalling
EDAS CE SDK, 65
uploading files
device connection, confirming, 71
utilities
CEFlush, 34, 62
DebugLauncher, 34, 57
LCmdSet, 33, 39
MonitorCE, 33, 35
Remote Manager, 45
SerialSocket, 34, 61
summary of capabilities, 33
TimeSync, 34, 59
utility programs
Remote Manager
error, troubleshooting, 46
help links, 47
home page, 47
password protection, 46
security manager
allowing access to folders, 45, 53
WebDevice
server system configuration, 55
version number, 145
voltage input configuration, 12
wall mount option, 30
watchdog timer, 5
WebDevice, 45
server system configuration, 55
WINS addresses
configuring, 37
reading, 41
reconfiguring, 42
wizards, 72
word size (serial)
reading, 41
reconfiguring, 43

writing custom programs, 72