ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Measurement Computing CIO-DIO24H* at our website: *Click HERE*

ComputerBoards, Inc.

Universal *Library*™ Programmers Manual

Universal Data Acquisition & Control Library

Revision 2.1, February 1994

ComputerBoards, Inc.
125 High Street, #6, Mansfield, MA 02048
(508) 261-1123 FAX 261-1094

(C) Copyright 1993, 1994 ComputerBoards, Inc. NO PART OF THIS MANUAL MAY BE REPRODUCED IN ANY FORM WITHOUT WRITTEN PERMISSION FROM COMPUTERBOARDS, INC.

Introduction **Page1**
Installation and Configuration **Page2**
Installation **Page2**
Description of Files on the Installation Disks **Page2**
CB.CBG File **Page4**
What if I don't want to worry about a CB.CFG configuration file? **Page5**
Running the InstaCal Installation/Calibration Program **Page5**
Overview of the Universal Library **Page6**
Analog I/O Functions **Page6**
Digital I/O Functions **Page7**
Thermocouple Input Functions **Page7**
Counter Functions **Page7**
Streamer File Functions **Page8**
Misc. Functions **Page8**
Memory Board Functions **Page9**
How to Use the Library **Page9**
Using the Library With Any Language **Page9**
Using the Library with DOS BASIC **Page10**
BASIC Header File Page10
Using Library Within Integrated BASIC Environment Page10
Using The Library With The BASIC Command Line Compiler Page10
Sample Basic Programs Page11
Passing Arguments to Universal Library Page11
DataArray Argument with Multiple Channels Page11
Using String Arguments Page12
BACKGROUND operation Page12
Using the Library with Visual Basic for DOS **Page12**
Compiling Stand Alone EXE files Page12
Using the Library with C **Page12**
Using the Library with Turbo Pascal **Page13**
Using the Library with LabView **Page13**
Using Universal Library from Windows **Page15**
Visual Basic for Windows Page15
Visual C++ For Windows Page15
Borland C/C++ For Windows Page16
Universal Library Functions **Page17**
cbAConvertData() **Page17**
cbAConvertPretrigData() **Page18**
cbAIn() **Page19**

cbAInScan() **Page20**
cbALoadQueue() **Page22**
cbAOut() **Page23**
cbAOutScan() **Page24**
cbAPretrig() **Page26**
cbATrig() **Page28**
COUNTERS - AN INTRODUCTION **Page29**
cbC8254Config() **Page30**
cbC8536Config() **Page31**
cbC8536Init() **Page32**
cbC9513Config() **Page33**
cbC9513Init() **Page35**
cbCFreqIn() **Page37**
cbCIn() **Page38**
cbCLoad() **Page39**
cbCStoreOnInt() **Page40**
cbDBitIn() **Page41**
cbDBitOut() **Page42**
cbDConfigPort() **Page43**
cbDIn() **Page44**
cbDInScan() **Page45**
cbDOut() **Page46**
cbDOutScan() **Page47**
cbErrHandling() **Page48**
cbFileAInScan() **Page49**
cbFileGetInfo() **Page51**
cbFilePretrig() **Page52**
cbFileRead() **Page54**
cbGetErrMsg() **Page55**
cbGetStatus() **Page56**
cbMemRead() **Page57**
cbMemReadPretrig() **Page58**
cbMemReset() **Page59**
cbMemSetDTMode() **Page60**
cbMemWrite() **Page61**
cbStopBackground() **Page62**
cbTIn() **Page63**
cbTInScan() **Page64**

Using the File Functions **Page66**
Overview **Page66**
Hard Disk vs Ram Disk Files **Page66**
Maximum Sampling Speed **Page66**
How To Determine Maximum Sampling Speed **Page66**
Speeding Up Disk Files (De-Fragmenting): **Page67**
What is a Ram Disk? **Page67**
Installing a RAM Disk: **Page67**
Using The Ram Disk **Page68**
Board Specific Information **Page69**
CIO-DAS16 Family Table of Examples **Page69**
CIO-DAS08 Family Table of Examples **Page70**
Analog Output Board Table of Examples **Page70**
Digital & Counter Table of Examples **Page71**
CIO-CTR Series Boards Limitation **Page71**
CIO-DAC Series Boards Limitations **Page71**
CIO-DAS08 Series Boards **Page71**
Analog Input Functions: Page71
Maximum Sampling Rate : Page72
Options: Page72
Wiring Requirements : Page72
Analog Output Functions Page72
Digital I/O Functions Page72
Counter I/O Functions Page72
CIO-DAS16 Series Boards **Page72**
Analog Input Functions: Page72
A/D Range: Page72
Maximum Sampling Rate: Page72
Options: Page73
Analog Output Functions: Page73
Digital I/O Functions: Page73
Counter I/O Functions: Page73
CIO-DAS48/PGA Board **Page73**
Analog Input Functions: Page73
A/D Ranges: Page73
Options Page73
Analog Output Page73
Digital I/O Functions Page73
Counter I/O Functions Page74
CIO-DAS1400 & CIO-DAS1600 Series Boards **Page74**
Analog Input Functions: Page74

CIO-DAS1402, CIO-DAS1602, CIO-DAS1602/16 Ranges Page74
CIO-DAS1401, CIO-DAS1601/12 Ranges Page74
Maximum Sampling Rate: Page74
Options Page74
Analog Output: Page74
Digital I/O Functions: Page74
Counter I/O Functions Page74
CIO-DIO Series Boards **Page75**
CIO-DIO24/CTR3 ..... **Page75**
CIO-DDA06 ..... **Page75**
CIO-DISO48 ..... **Page75**
CIO-EXP Series Boards **Page75**
CIO-INT32 ..... **Page75**
CIO-PDISO8 ..... **Page75**
CIO-PDMA16 ..... **Page75**
CIO-RELAY16 ..... **Page76**
PPIO-DIO24H **Page76**
PPIO-AI8 ..... **Page76**
PPIO-CTR06 Boards **Page76**
cSBX-DDA04 Board **Page76**
PCM-DAS08 Board **Page76**
Appendix A. Error Codes **Page77**

NOTICE OF COPYRIGHT AND TRADEMARKS

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form by any means, electronic, mechanical, by photocopying, recording or otherwise without the prior written permission of ComputerBoards, Inc.

MEGA-FIFO, Universal*Library*, *Insta*Cal, *Harsh Environment Warranty* the CIO-, PCM-, and PPIO- part number prefix and **Computer**Boards are registered trademarks of ComputerBoards, Inc.

DT-Connect is a trademark of Data Translation, Inc.

LabView is a trademark of National Instruments Inc.

IBM, PC and PC/AT are trademarks of International Business Machines Corp.

Information furnished by ComputerBoards, Inc. is believed to be accurate and reliable; however, no responsibility is assumed by ComputerBoards, Inc. for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or copyrights of ComputerBoards, Inc.

Each original copy of Universal Library is licensed for use on one CPU at a time. It is theft to make copies of this program for simultaneous use. There is nothing but your conscience to protect ComputerBoards' intellectual property in this product once it is in your possession. We have sold you the product of our labors at a fair price, with conditions. Do not lower yourself. Do not be a thief.

Introduction

The Computer Boards Universal Library is the software that you need to write your own programs that use any of the Computer Boards data acquisition and control boards.

The library is universal in two ways.

Universal Across Boards - The library contains high level functions for all of the common operations for all boards. Each of the boards has different hardware but the Universal Library hides these differences from your program. So, for example, a program written for use with one A/D board will work "as is" with a different A/D board.

Universal Across Languages - The Universal Library provides the identical set of functions and arguments for BASIC, C, and Pascal. If you switch languages you won't have to relearn a new library, with new syntax, and different features.

Languages supported by Revision 2 of the Universal Library are:

MicroSoft Windows Languages
Visual Basic
Visual C/C++
Quick C for Windows
Microsoft C
MicroSoft DOS Languages
QuickBasic 4.5
Professional Basic 7.0
Visual Basic for DOS
Quick C
Microsoft C

Borland Windows Languages
Borland C++
Borland DOS Languages
Turbo Pascal
Turbo C
Turbo C++
Borland C++
National Instruments
LabView

The ComputerBoards products supported by Revision 2 of the Universal Driver are:

CIO-CTR05	CIO-DAS16/330	CIO-DIO24	CIO-PDMA16
CIO-CTR10	CIO-DAS16/330i	CIO-DIO24/CTR3	CIO-RELAY16
CIO-DAC02	CIO-DAS16/F	CIO-DIO24H	CIO-SSH16
CIO-DAC08	CIO-DAS16/Jr.	CIO-DIO48	cSBX-CTR05
CIO-DAC08i	CIO-DAS16/M1	CIO-DIO48H	cSBX-DDA4/F
CIO-DAC16	CIO-DAS1600/12	CIO-DUAL-AC5	cSBX-DIO24H
CIO-DAC16i	CIO-DAS1600/16	CIO-DIO96	MEGA-FIFO
CIO-DAS08	CIO-DAS16Jr/CTR	CIO-DISO48	PCM-DAS08
CIO-DAS08-AOH, L, M	CIO-DAS48-PGA	CIO-EXP16	PPIO-AI08
CIO-DAS08-PGH, L, M	CIO-DDA06/12	CIO-EXP32	PPIO-CTR06
CIO-DAS1400	CIO-DDA06/16	CIO-INT32	PPIO-DIO24H
CIO-DAS16	CIO-DIO192	CIO-PDISO8	

As new boards are designed, they are added to the Universal Library. Please call if you do not see your board listed here.

Installation and Configuration

This chapter describes how to install the software on your computer and how to configure the software for the boards that you will be using with it.

Installation

On the disk labeled InstaCal, there is an automated installation program, INSTALL.EXE. Please inster the InstaCal disk into the floppy drive and type:

A:\INSTALL <Enter>

You will be prompted for some information. Please follow the instructions and if possible accept the defaults. If this is your first installation, we urge you to accept the defaults. It will be easier to assist you in the event of trouble if you accept the defaults.

Install does these things (explanation presumes you accept defaults).

1) Creates directories:

C:\CB	and numerous directories SUB to that, and
C:\LABVIEW\VI.LIB\CB	if you have LabView and install the LabView drivers.

2) Adds the directory:

;C:\CB	to the PATH statement in your AUTOEXEC.BAT file.

4) Adds the line to AUTOEXEC.BAT:

SET CBDIREC=C:\CB

5) Copies these files into the C:\CB directory:

INSTACAL.EXE	Installation & calibration program and related help files.
INSTACAL.CFG	InstaCal parameters configuration.
CB.CFG	I/O Board address, interrupt and configuration file.
CBW.DLL	The Windows Language DLL
Universal Library	Programmaing language support in sub directories.

6) Copies these files into the C:\LABVIEW\VI.LIB\CB directory:

DAS16.LLB	Library for all LabView VI interfaces to UniversalLibrary.
X###.VI	Examples.

After you have installed all the software using the automated install program, please reboot your computer.

Description of Files on the Installation Disks

The disks contain these files. The executable files contain compressed versions of many other files.

README. contains instructions for expanding compressed files.

REV.DOC contains the revision history of the library; corrected bugs, changes etc.

CBCODE.EXE contains various files that are used with all programming languages.

BASIC.EXE contains files that are used when programming in DOS BASIC (Quick BASIC, Professional BASIC, Visual BASIC for DOS).

PASCAL.EXE contains files that are used when programming in DOS PASCAL (Borland).

C.EXE contains files that are used when programming in DOS C (Microsoft and Borland).

WINDOWS.EXE contains the Windows DLL, CBW.DLL

VBWIN.EXE contains the VisualBasic for Windows examples

CWIN.EXE contains the C for Windows examples

CBLVIEW contains the LabView interface and example programs

CBCODE.EXE - Contains the following files:
INSTACAL.EXE - InstaCal Installation/Calibration Program
INSTACAL.CFG - InstaCal configuration file
HELP.TXT - InstaCal's Help screen text
HELP.IDX - InstaCal's Help screen index
HELVB.FON - InstaCal's text font
CB.CFG - Hardware configuration file
JTHERM.DAT \
KTHERM.DAT -\
TTHERM.DAT --\
ETHERM.DAT ---- Thermocouple linearization tables
STHERM.DAT --/
RTHERM.DAT -/
MAKESTRM.EXE - Creates a "streamer" file
FRAGTEST.EXE - Checks "streamer" file for fragmentation
RDSTREAM.EXE - Reads a "streamer" file

BASIC.EXE contains the following files
CB.BI - BASIC include file
MAKEQB.BAT - Batch file for Quick BASIC
MAKEPB.BAT - Batch file for Professional BASIC
MAKEVB.BAT - Batch file for Visual BASIC
CBQB.LIB - Quick BASIC library
CBQB.QLB - Quick BASIC quick library
CBPB.LIB - Professional BASIC library
CBPB.QLB - Professional BASIC quick library
CBVB.LIB - Visual BASIC library
CBVB.QLB - Visual BASIC quick library
UDAIxx.BAS - Analog Input BASIC example programs.
UDAOxx.BAS - Analog Output BASIC example programs.
UDCTCxx.BAS - Counter input.
UDDIxx.BAS - Digital Input BASIC example programs.
UDDOxx.BAS - Digital Output BASIC example programs.
UDFIxx.BAS - File Input BASIC example programs.
UDGTxx.BAS - Get status BASIC example programs.
UDMMxx.BAS - Memory Board BASIC example programs.
UDTIxx.BAS - Thermocouple Input BASIC example programs.

C.EXE contains the following files:
CB.H - C include file
MAKETC.BAT - Batch file for compiling with Turbo C
MAKEMC.BAT - Batch file for compiling with Microsoft C
MAKEBC.BAT - Batch file for compiling Borland C
CBCS.LIB - C library (small model)
CBCM.LIB - C library (medium model)
CBCL.LIB - C library (large model)
UDAIxx.C - Analog Input C example programs.
UDAOxx.C - Analog Output C example programs.
UDCTCxx.C - Counter input.
UDDIxx.C - Digital Input C example programs.

UDDOxx.C - Digital Output C example programs.
UDFIxx.C - File Input C example programs.
UDGTxx.C - Get status C example programs.
UDMMxx.C - Memory Board C example programs.
UDTIxx.C - Thermocouple Input C example programs.

PASCAL.EXE contains the following files
CB.PAS - Interface unit for Turbo Pascal
CB.TPU - Compiled version of CBDECL.PAS
MAKETP.BAT - Batch file for compiling/linking
CBTPxx.OBJ - Turbo Pascal library
UDAIxx.PAS - Analog Input Pascal example programs.
UDAOxx.PAS - Analog Output Pascal example programs.
UDCTCxx.PAS - Counter input.
UDDIxx.PAS - Digital Input Pascal example programs.
UDDOxx.PAS - Digital Output Pascal example programs.
UDFIxx.PAS - File Input Pascal example programs.
UDGTxx.PAS - Get status Pascal example programs.
UDMMxx.PAS - Memory Board Pascal example programs.
UDTIxx.PAS - Thermocouple Input Pascal example programs.

WINDOWS.EXE contains the following files
CBW.DLL - Windows DLL version of the Universal Library

VBWIN.EXE contains the following files
VBASIC Subdirectory
- CBW.TXT - Universal Library declaration file
- ADTEST.BAS - Source code for example program
- ADTEST.FRM - Form for example program
- ADTEST.MAK - Project file for example program
- SINEWAVE.ICO - Icon for example program

CWIN.EXE contains the following files
WINC Subdirectory
- CBW.H - Universal Library declaration file
- CBW.LIB - Universal Library LIB file
- WINCDEMO.C - Source code for example program
- WINCDEMO.RC - Resource file for example program
- WINCDEMO.DEF - Linker DEF file for example program
- WINCDEMO.MAK - Microsoft projec6 (make) file
- WINCDEMO.PRJ - Borlan project file
- ANALOG.ICO - Icon for example program.

CBLVIEW.EXE contains the following files
VI.LIB\CB Subdirectory
- DAS16.LLB - Universal Library VIs.
- Xname.VI - Examples of the UniversalLibrary VI for A/D, D/A, CTR, DIO

CB.CBG File

All board specific information, including current installed options, are stored in a file CB.CFG, which is read by UniversalLibrary. InstaCal modifies this file when it store board configuration information in it. Universallibrary reads that information from the file. Universal Library cannot update the file CB.CFG The InstaCal program configures the CB.CFG file for the boards that are installed in your system. This program asks you which boards are installed and which options have been selected on the boards (via switches and jumpers).

Whenever you run a program that calls the Computer Boards Universal Library the first library call will cause the program to read the configuration file (CB.CFG).

The program will first look in the current directory. If it doesn't find it there, it will look in the directory that the program is in (which might not be the current directory). If it can't find it there it will print an error message on the screen.

To ensure that UniversalLibrary and InstaCal can always find the CB.CFG file, the path statement should include a refernce to C:\CB and the environment variable CBDIREC should be set to the directory which contains CB.CFG. That is why the Installation program adds the line:
SET CBDIREC=C:\CB
to your AUTOEXEC.BAT

What if I don't want to worry about a CB.CFG configuration file?

We have had that question and there is no easy answer. If you have you are unhappy with keeping a configuration file with your application code please consiedr this. **Almost no modern program of any merit is contained in a single file.** Think about it. Windows has WIN.INI and SYSTEM.INI, most Windows applications have a file named APNAME.INI.

The CB.CFG file contains all the board specific information. Separating the code from the configuration of the A/D board is an enormous benfit. If you are distributing your application code, this means that you can upgrade or downgrade the hardware without having to modify a single line of code! And, you can do so to software installed in the field without having to re-distribute code.

ComputerBoards chose the separate configuration file strategy for its many benfits and that is the only way to use UniversalLibrary. There is no way for you to embed the configuration information into the code.

Take a close look at the software on your computer. You will be hard pressed to find any of it contained wholly in a single file.

Running the InstaCal Installation/Calibration Program

Use the InstaCal program to install boards in your system. To run the program, type:
Instacal <Enter>

Select "Install" from the Main Menu either with a mouse or by pressing the "Alt-I" keys.

Select "Board #0" from the next menu.

Choose the type of board that you want to install.

A menu screen is displayed for that board's options. The title on the menu will tell you the board number that has been assigned to it. The first board is always Board #0 and additional boards are numbered 1,2,3 etc. These board numbers are how you will henceforth refer to the boards from your programs.

Set all of the options for the board. Some of the options are the hardware configuration options that are set by DIP switches and jumpers on the board. Once you have set all of the board options, select the last choice on the menu to display a picture of the board. Verify that all of the switch and jumper settings on your board are the same as in the picture.

Some boards (EXP, SBX, MEGA_FIFO) are only used in conjunction with other boards - never by themselves. These boards are installed by selecting choices from the 'parent' board's option menu. For example, the DAS-16 board's option screen has an item called "Expansion Boards". Select that choice and a new menu will display the possible expansion boards that may be installed (EXP-16, EXP-32, SSH16).

Once you have added all of the boards exit from the program by selecting "Exit" from the Main Menu.

InstaCal saves its configuration information in a file called CB.CFG. This file is read by all programs that use the Universal Library.

CALibrate Option: InstaCal has calibration routines for all ComputerBoards I/O cards which require calibration. You must first install the board using the install pull-down menu. Calibration for I/O boards is fully explained in the hardware manual.

Overview of the Universal Library

The Universal Library consists of a set of functions that are callable from your program. These functions are grouped according to their purpose. All of the groups except for Misc. are based on which type of devices they are used with.

Analog I/O Functions

The Analog function names all begin with cbA. These functions perform analog input and output and convert analog data.

cbAin() - Single Analog input
Takes a single reading from an analog input channel (A/D).

cbAInScan()- Analog Input Scan
Repeatedly scans a range of analog input (A/D) channels. The channel range, the number of iterations, the sampling rate, and the A/D range can all be specified. The data that is collected is stored in an array.

cbALoadQueue() - Load Chan/Gain queue
Loads a series of chan/gain pairs into A/D board's queue. These chan/gains will be used with all subsequent analog input functions.

cbAOut() - Single analog output
Outputs a single value to an analog output (D/A).

cbAOutScan() - Analog output scan
Repeatedly scans a range of analog output (D/A) channels. The channel range, the number of iterations, and the rate can all be specified. The data values from consecutive elements of an array are sent to each D/A channel in the scan.

cbAPretrig() - Analog pre-triggered input
Repeatedly scans a range of analog input (A/D) channels waiting for a trigger signal. When a trigger occurs it returns the specified number of samples and points before the trigger occurred. The channel range, the sampling rate, and the A/D range can all be specified. All of the data that is collected is stored in an array.

cbATrig() - Analog trigger
Reads the analog input and waits until it goes above or below a specified threshold.

cbFileAInScan() - Analog input to a file

cbFilePretrig() - Pre-triggered analog input to a file

cbAConvertData() - Converts analog data
Each raw sample from analog input is a 16 bit value. On 12 bit A/D boards it consists of a 12 bit A/D value along with a four bit channel number. On 16 bit A/D boards it contains the 16 bit A/D value.

This conversion is done automatically by the cbAIn() function. It can also be done automatically by the cbAInScan() function with the CONVERTDATA option. In some cases though, it may be useful or necessary to collect the data and then do the conversion sometime later. The cbAConvertData() function takes a buffer full of unconverted data and converts it.

cbAConvertPreTrigData() - Convert pre-trigger data
When data is collected with the cbAPretrig() function, the same conversion needs to be done as described above for cbAConvertData. There is a further complication though.

cbAPretrig() collects analog data into an array. It treats the array like a circular buffer. While it is waiting for the trigger to occur it fills the array. When it gets to the end it resets to the start and begins again. When the trigger signal occurs it continues collecting data into the circular buffer until the requested number of samples have been collected.

When the data acquisition is complete all of the data is in the array but it is in the wrong order. The first element of the array does not contain the first data point. The data has to be rotated in the correct order.

This conversion can be done automatically by the cbAPretrig() function with the CONVERTDATA option. In some cases though, it may be useful or necessary to collect the data and then do the conversion sometime later. The cbAConvertData() function takes a buffer full of unconverted data and converts it.

Digital I/O Functions

The digital function names all begin with "cbD". These functions perform digital input and output. They operate on all types of digital I/O ports.

cbDBitIn() - Digital bit input
Reads a single bit from a digital input port.

cbDBitOut() - Digital bit output
Sets a single bit on a digital output port.

cbDConfigPort() - Configures digital outputs
Selects whether a digital port is an input or an output.

cbDIn() - Digital byte input
Reads a specified digital input port.

cbDInScan() - Digital multiple byte input
Reads a specified number of bytes from a digital input port at a specified rate.

cbDOut() - Digital byte output
Writes a byte to a digital output port.

cbDOutScan() - Digital multiple byte output
Writes a series of bytes to a digital output port at a specified rate.

Thermocouple Input Functions

The thermocouple function names begin with "cbT". These functions convert a raw analog input from an EXP board to temperature.

cbTIn() - Single thermocouple input
Reads an EXP channel, filters it, does the cold junction compensation, linearization and converts it to temperature.

cbTInScan() - Scan a range of thermocouple inputs
Reads the temperature from a range of EXP channels as described above. Returns the temperature values to an array.

Counter Functions

The counter function names begin with "cbC". These functions load, read and configure counters. There are three types of counter chips used in Computer Board products - 8254s, 8536s and 9513's. Some of the counter commands only apply to one type of counter.

cbC8254Config() - Configures 8254 counter
Selects the basic operating mode of an 8254 counter.

cbC8536Config() - Sets operating mode of 8536 counter.
This function sets all of the programmable options that are associated with an 8536 counter chip.

cbC8536Init() - Initializes 8536 counter
Initializes and selects all of the chip level features for a 8536 counter board. The options that are set by this command are associated with each counter chip, not the individual counters within it.

cbC9513Config() - Sets operating mode of 9513 counter.
This function sets all of the programmable options that are associated with a 9513 counter chip. It is similar in purpose to cbC8254Config() except that it is used with a 9513 counter.

cbC9513Init() - Initializes 9513 counter

Initializes and selects all of the chip level features for a 9513 counter board. The options that are set by this command are associated with the each counter chip, not the individual counters within it.

cbCFreqIn() - Measures frequency of a signal
This function measures the frequency of a signal by counting it for a specified period of time (GatingInterval) and then converting the count to count/sec (Hz). It only works with 9513 counters.

cbCIn() - Reads a counter.
Reads a counters current values.

cbCLoad() - Load a counter.
Loads a counter with an initial count value.

cbCStoreOnInt() - Store counter value when interrupt occurs.
Installs an interrupt handler that will store the current count whenever an interrupt occurs. This function only works with 9513 counters.

Streamer File Functions

The file function names begin with "cbF". These functions create, fill, and read "streamer" files. These functions also let you collect huge amounts of analog input data. The amount of data is limited only by your available disk space.

cbFileAInScan() - Transfer analog input data directly to file.
Very similar to cbAInScan() except that the data is stored in a file instead of an array.

cbFilePretrig() - Pre-triggered analog input to a file.
Very similar to cbAPretrig() except that the data is stored in a file instead of an array.

cbFileGetInfo() - Reads "streamer" file information.
Each streamer file contains information about how much data is in the file and the conditions under which it was collected (sampling rate, channels, etc.). This function reads that information.

cbFileRead() - Reads data from "streamer" file.
Reads a selected number of data points from a streamer file into an array.

Misc. Functions

These functions are the odds and ends. They have to do with error handling and managing background operations.

cbErrHandling() - Selects type of error handling.
The universal library has a number of different methods of handling errors. This function selects which of these methods will be used with all subsequent library calls. The options include stopping the program when an error occurs and printing error messages.

cbGetErrMsg() - Returns an error message for a given error.
All library functions return error codes. This function converts an error code to an error message.

cbGetStatus() - Returns status of background operation.
Once a background operation is started your program will need to periodically check on its progress. This function returns the current status of the process.

cbStopBackground() - Stop a background process.
It is sometimes necessary to stop a background process in the case of an error or if the process has been set up to run continuously forever. This function will stop a background process that is running.

Memory Board Functions

The memory board functions all begin with "cbM". These functions read/write and control memory boards (MEGA-FIFO).

The most common use for the memory boards is to store large amounts of data from an A/D board via a DT-Connect cable between the two boards. To do this, you should use the EXTMEMORY option with cbAInScan() or cbAPretrig().

Once the data has been transferred to the memory board you can use the memory functions to retrieve the data.

cbMemSetDTMode() - Set DT-Connect Mode on Memory Board
The memory boards have a DT-Connect interface which can be used to transfer data through a cable between two boards rather than through the PC's system memory. The DT-Connect port on the memory board can be configured as either an input (from an A/D) or as an output (to a D/A). This function configures the port.

cbMemReset() - Resets the Memory Board Address
The memory board is organized as a sequential device. When data is transferred to the memory board it is automatically put in the next address location. This function resets the current address to the location 0.

cbMemRead() - Read Data From Memory Board
Reads a specified number of points from a memory board starting at a specified address.

cbMemWrite() - Writes Data To The Memory Board
Writes a specified number of points to a memory board starting at a specified address.

cbMemReadPretrig - Reads Data Collected With cbAPretrig()
The cbAPretrig() function writes the pre-triggered data to the memory board in a scrambled order. This function unscrambles the data and returns it in the correct order.

How to Use the Library

The Computer Boards Universal Library is callable from Quick BASIC, Professional BASIC, Visual BASIC, Visual C++, Microsoft C, Quick C, Turbo C, Borland C++, and Borland Turbo Pascal. This chapter describes how to use the library from each of the languages. The first section of the chapter describes details of the library that apply to all languages. The following sections describe the differences for each language.

HOW TO USE THE LIBRARY SUCCESSFULLY

1) Set up and test your boards with InstaCal™. The library will not function until InstaCal™ has created a configuration file.

2) Use the example programs for the language you program in. This manual explains the functions and contains other necessary information, but it is incomplete without reference to and review of the examples

Using the Library With Any Language

The interface to all languages is a set of function calls and a set of constants. The list of function calls and constants are identical for each language. All of the functions and constants are defined in a "header" file for each language. Refer to the sections below for each language.

Function Arguments
Each library function takes a list of arguments and returns an error code. Some functions also return data via their arguments. For example one of the arguments to cbAIn() is the name of a variable in which the analog input value will be stored. All function arguments that return data are listed in the "Returns:".

Constants

Many functions take arguments that must be set to one of a small number of choices. These choices are all given symbolic constant names. So for example, TIn() takes an argument called Scale that must be set to CELSIUS, FAHRENHEIT, or KELVIN. These constant names are defined and assigned a value in the "include" file for each language. Although it is possible to use the numbers rather than the symbolic constant names, we strongly recommend that you use the names. This will make your programs more readable and more compatible with future versions of the library. The numbers may change in future versions but the symbolic names will always remain the same.

Options Arguments:
Some library functions have an argument called Options and all options have a default. Some options have an alternative, such as, DTCONNECT and NODTCONNECT, one of which is the default value. Other options do not have a stated alternative. The alternative is the absence of that option. For example BACKGROUND does not have an alternative called 'foreground', but if you do not choose BACKGROUND in the option argument, it will probably occur in the foreground. The option argument is used to turn on and off various optional features associated with the function. If you set Options=0 then the function will set all of these options to the default value, or off.

Individual options can be turned on by adding them to the Options argument. So for example Options = BACKGROUND will turn on the "background execution" feature. Options = BACKGROUND CONTINUOUS will select both the "background execution" and the "continuous execution" feature.

Using the Library with DOS BASIC

Each of the supported versions of BASIC consists of two distinct systems. Programs can be loaded into the BASIC editor and run from within the integrated BASIC environment. Programs can also be compiled by a command line compiler into stand-alone executable programs that can be run on their own without the help of the integrated BASIC environment. The Universal Library provides the tools for both methods.

BASIC Header File

Every BASIC program that uses the Universal Library must have a line which includes the BASIC Universal Library header file - CB.BI. The following line should appear near the start of every program, before the first library call is made.

```
'$INCLUDE: 'CB.BI'
```

Using Library Within Integrated BASIC Environment

When you start up BASIC you must specify that you want to load the "quick library" version of the Universal Library.

For Quick BASIC type:
qb /l cbqb

For Professional BASIC type:
qb /l cbpb

For Visual Basic for DOS, type:
vbdos /l cbvb

Using The Library With The BASIC Command Line Compiler

To build stand-alone executable files with the command line compiler you must link your compiled BASIC program with the stand-alone version of the Universal Library. To do this you must supply the linker with the library name.

The names of the .lib files are:

QuickBasic	CBQB.LIB
Professional Basic	CBPB.LIB
VisualBasic for DOS	CBVB.LIB

Sample Basic Programs

The sample BASIC programs included demonstrate how to call each function in the Universal Library. These programs can be run from within the integrated BASIC environment. They can also be compiled using the command line compiler with the batch file supplied. The names of the batch files are:

QuickBasic	MAKEQB.BAT
Professional BASIC	MAKEPB.BAT
VisualBasic for DOS	MAKEVB.BAT

Passing Arguments to Universal Library

All of the functions in the library require that arguments be passed to them. The file CB.BI contains the definition of all the argument types that are passed. In general there are two classes of arguments - Inputs and Outputs.

Input Arguments: All agruments that are only used as inputs to a library function are listed in the CB.BI file definition as BYVAL. For these arguments you may either pass a variable or a constant. So for example both of these versions are acceptable:

```
BoardNum% = 0
cbStopBackground (BoardNum%)
```

or

```
cbStopBackground (0)
```

Output Arguments: Some arguments are used by the library to pass information back to the caller. For example, the value from and A/D is returned by cbAIn() to the DataValue% argument. Others are used as both inputs and outputs. For example, the Rate& argument specifies the requested sampling rate for cbAInScan() (Input). The actual sampling rate may vary from the requested sampling rate so the actual rate is returned by cbAInScan() to the Rate& argument (Output).

Output and Input/Output arguments are defined in the CB.BI function definitions as SEG. All SEG arguments may only be passed via a variable. For example:

```
Count& = 1000
Rate& = 15000
cbAInScan (0, 0, 1, Count&, Rate&, BIP5VOLTS, DataArray(0), 0)
```

is correct, but,

```
cbAInScan (0, 0, 1, 1000, 15000, BIP5VOLTS,DataArray(0), 0)
```

is NOT correct.

DataArray Argument with Multiple Channels

Various functions have a DataArray argument. The DataArray either receives the data from an input function such as cbAInScan, or contains the data to be sent to an output function such as cbAOutScan().

The DataArray must be DIMensioned to be large enough to contains all of the data. The array can either be dimensioned with a single dimension or two dimensions. When sampling more than one channel it is often more straightforward to use a multiple dimensioned array. So for example either of the following could be used with the code below:

```
DIM DataBuffer (1999)          'One dimensional array. 0 to 1,999 (2,000) elements.
   OR
DIM DataBuffer (1, 999)        'Two dimensional array. 0 & 1 with 0-999 (1,000) elements each.

LowChan% = 2
HighChan% = 3
Count& = 2000
Rate& = 1000
cbAInScan (0, LowChan%, HighChan%, Count&, Rate&, BIP5VOLTS, DataBuffer(0), 0)
```

The advantage of using the multi-dimesioned array is that you can directly address the data in the array by channel. So in the example above - DataBuffer (0,99) address the 100th sample for channel 2 (channel 2 was the first element in the array; LowChan%).

Using String Arguments

cbGetErrMsg() requires that a string variable be passed as an argument. This string variable must have been previously allocated to be large enough to hold the longest error message. To do this use Quick BASIC's space$ function as it is done in the example program.

```
ErrStr$ = space$ (ERRSTRLEN)
```

BACKGROUND operation

If you use the BACKGROUND option with any function then you must declare the associated data array as '$STATIC.
Unless you declare an array as '$STATIC, BASIC may move the array around in memory as the program is executing. Whenever you use the BACKGROUND option the I/O function reads/writes from the data array in the background while the BASIC program continues executing in the "foreground". If BASIC moves the array while the I/O function is reading/writing it you will cause intermittent and unpredictable problems.

Using the Library with Visual Basic for DOS

Compiling Stand Alone EXE files

Due to a quirk in Visual Basic for DOS, if you compile a stand alone EXE file from within IDE and you set the EXE type to "Stand alone EXE file", you will get the following message: "fix up overflow at 334 in segment -TEXT target external 'B$CEND'". The compiled program will run without error. It appears that the error message is an error.

Using the Library with C

The C libraries included with the system can be used with either the MicroSoft or Borland C compilers.

C Header File
Every C program that uses the Universal Library must have a line which includes the Universal Library C header file - CB.H. The following line should appear near the start of every program, before the first library call is made.

```
#include cb.h"
```

Memory Models
Both Borland and MicroSoft C compilers support different memory models. The Universal Library comes with the following three versions of the library.

CBCS.LIB - For use with small and compact model
CBCM.LIB - For use with medium model
CBCL.LIB - For use with large and huge model

Large Data Arrays
The Universal Library supports input and output from very large > 64K data arrays. If your program is going to use a single array that is greater then 64K then you should compile it for the "huge" model and use the CBCL.LIB.

Compiling The Sample C Programs
The example programs demonstrate how to call each of the Universal Library functions from a C program. Two batch files are provided that show how to compile and link the sample programs using the Microsoft and Borland compilers.
MAKEMC.BAT - compile and link with Microsoft C
MAKETC.BAT - compile and link with Borland C

Using the Library with Turbo Pascal

The Turbo Pascal version of the Universal Library comes pre- compiled for Turbo Pascal Ver 6.0 but the necessary source code is provided to work with other versions.

Using Different Versions of Turbo Pascal

The included CB.TPU file was generated by compiling the CB.PAS file with Version 6.0 of the Turbo Pascal compiler. If you are using a different version of Turbo Pascal, you should recompile CB.PAS to generate a new CB.TPU file that will be compatible with your version of the compiler.

To use the new library just recompile your Pascal program with the new version of CB.TPU Refer to the MAKETP.BAT for an example.

Turbo Pascal Header File
Every Turbo Pascal program that uses the Universal Library must have a line which includes the Turbo Pascal Universal Library header file - CB.PAS. The following line should appear near the start of every program, before the first library call is made.

```
uses cb;
```

Sample Pascal Programs
The included sample Pascal programs demonstrate how to call each function in the Universal Library. The MAKETP.BAT batch file will compile and link the program into an executable file.

Using the Library with LabView

The ComputerBoards Labview driver is a "front end" for ComputerBoard's Universal Library. There are two distinct sets of VI's (LabView Virtual Instruments) that come with the driver. Universal Library VI's and examples.

The Computer Boards installation program installs the new LabView VI's in a new subdirectory called CB, under the VI.LIB directory in the LabView directrory tree. If you used the standard directory name when installing LabView and the Computer Boards software then the directory will be \labview\vi.lib\cb.

The VI's in the DAS16.LLB VI library correspond very closely to the functions in the Universal Library.

The remaining VI's are examples that demonstrate how to use the Universal Library VI's.

Most of the Universal Library VI's exactly match an individual function in the Universal Library. The front panel for each of these VI's have controls and indicators for each of the inputs and outputs the Universal Library function. Refer to the Universal Library User's Manual for a detailed description of each of these inputs and outputs.

The list below describes the VI's that vary in some way from their corresponding Universal Library functions.

Background/Foreground Operation The Universal Library scanning functions (cbAInScan, cbAPretrig, cbAOutScan, cbDInScan, cbDOutScan) can be run in either the background or the foreground by using the BACKGROUND option. In the LabView driver each of these functions has two VI's (AInScFg.vi, AInScBg.vi for example). One of the VI's is for foreground operation and one is for background.

The foreground versions are very straight forward. They take various input parameters and they return an array full of data.The Background versions are more complex. They must be used in combination with two other VI's - GetStatus and StopBg.

Context - The backround scanning functions take various parameters as input and generate a "context" as output. This context contains all of the information about the background process that will be shared by the other VI's. This context will berepresented on the diagram as a thick pink line. A Context is both an input and output to GetStatus. The StopBg VI takes a Context as input and returns the data array as an output.

OptAIn - The analog input scanning functions (cbAInScan and cbAPretrig) in the Universal Library support a large number of optional features via the Options argument. If Options is set to 0 all optional features are set to their default behavior. In the LabView driver a separate VI called OptAIn is provided to set the various options. The resultion Options value can then be used as an input to the input scanning VI's.
Scaling VI's - The ScalePoint and ScaleArray VI's have no corresponding functions in the Universal Library. These VI's take an A/D value (ScalePoint) or an array of A/D values (ScaleArray) as inputs and return a point or an array of points that have been converted to engineering units. The other inputs to these functions are a MinValue, MaxValue and 12 bit/16 bit boolean.

MinValue and MaxValue should be set to the values in engieering units that correspond to the limits of the currently selected range of the A/D converter. So, for example, if the A/D was set for 0-10 volt range and was connected to a transducer that measured pressure, and the scale of the transducer was 100 psi/per volt. MinValue would be set to 0 (0 Volts=0 PSI) and MaxValue should be set to 1000 (10 Volts = 1000 PSI).

If the A/D board has a 16 bit converter (DAS1600/16) the 12/16 bit boolean input must be set to 16. If it's a 12 bit converter it should be set to 12.

IMPORTANT - The data passed to the scaling functions must have been previously "converted". If the data was either read from a 16 bit converter or it was collected with the cbAIn function it will be in the correct format. If the data is coming from one of the scanning input functions from a 12 bit converter then you must either use the scanning function's CONVERTDATA option or else explicitly convert the data using the appropriate conversion VI ACvtData or ACnvPrDt.

Using Universal Library from Windows

A Windows DLL version of the Universal Library (CBW.DLL) may be called from any language that supports calls to external DLLs. Examples programs for Visual Basic and both Borland and MS C illustrate the use of CBW.DLL.

Revision 2 of the Universal Library must be installed from DOS. Do not install it from a DOS window. Exit Windows.

Please use the installation program on the INstaCal disk. see installation in this manual.

Revision 2.0 Windows support of the Universal Library is a subset of the DOS language support. High speed transfer methods such as interrupt, DMA and REP-INSW are not supported. In general, all low speed analog I/O is fully supported and complete counter and digital I/O support is included. Where functions are partially or not supported, a note to that effect appears as the last item in the description of the function.

Visual Basic for Windows

You must include the Universal Library declaration file CBW.TXT in your program. You may do so by including it in the form that calls the library or in the GLOBALS.BAS (recommended). To include the CBW.TXT file in the global declaration:

Select Window, Project Window from the main menu
Select Globals from the list
Select Code, Load Text from the menu
Select or enter CBW.TXT

Once the declarations for the Universal Library have been added to your GLOBALS.BAS file you can call the CBW.DLL from any Form's event handlers.

CB.CFG Locations with Visual Basic

All programs that use the Universal Library read the CB.CFG configuration file. The search for the CB.CFG file begins in the current directory then proceeds to the directory the program calling the library resides in. Be sure the environment variable CBDIREC is set. See installation.

Be sure to include a copy of the CB.CFG configuration file with any compiled stand alone Visual Basic programs you wish to distribute to another machine or directory.

Error Handling in Visual Basic

There is a quirk in Visual Basic's integrated environment that causes a complete exit whenever an error occurs if cbErrHandling() is set to STOPALL or STOPFATAL Save changes to your programs before running them from within the environment. Compiled stand alone programs work correctly. Error handling performs as described in this manual. Using cbErrHandling DONTSTOP is recommeneded during program design.

Visual Basic Example Programs

The ADTEST program is an example of a simple Visual Basic program that calls the Universal Library. It has a single form. The LOAD event handler initializes the Universal Library error handling and the TIMER handler reads the inputs and updates the display.

The DOS Basic example programs illustrate the form of the Universal Library syntax, but will not run from Visual Basic.

Visual C++ For Windows

The Universal Library DLL is a set of C functions. You may use either the DLL or the C functions. To use the C functions from Visual C++ for Windows, include the header file CBW.H and the library file CBW.LIB in your project.

CB.CFG Locations with Visual C++

All programs that use the Universal Library read the CB.CFG configuration file. The search for the CB.CFG file begins in the current directory then proceeds to the directory the program calling the library resides in. Be sure the environment variable CBDIREC is set. See installation

Be sure to include a copy of the CB.CFG configuration file with any compiled stand alone Visual C++ programs you wish to distribute to another machine or directory.

Visual C++ Example Programs

The WINCDEMO program is an example of a simple Visual C++ program that calls the Universal Library. To build the program use WINCDEMO.MAK. The program expects an 8 channel A/D board. You can modify the program for the board you own.

The DOS C example programs illustrate the form of the Universal Library syntax, but will not run from Visual C++.

Borland C/C++ For Windows

The Universal Library DLL is a set of C functions. You may use either the DLL or the C functions. To use the C functions from Borland C++ for Windows, include the header file CBW.H and the library file CBW.LIB in your project.

CB.CFG Locations with Boreland C++

All programs that use the Universal Library read the CB.CFG configuration file. The search for the CB.CFG file begins in the current directory then proceeds to the directory the program calling the library resides in. Be sure the environment variable CBDIREC is set. See installation

Be sure to include a copy of the CB.CFG configuration file with any compiled stand alone Boreland C++ programs you wish to distribute to another machine or directory.

Borland C/C++ Example Programs

The WINCDEMO program is an example of a simple C++ program that calls the Universal Library. To build the program use WINCDEMO.PRJ. The program expects an 8 channel A/D board. You can modify the program for the board you own.

When you build the WINCDEMO program the compiler will generate two warning messages which can be safely ignored. They are,

Parameter 'CmdLine' is never used.
Stack size is less than 1400h. It has been reset to 1400h.

the second of which may be corrected by adding the line STACK = 0x1400 to the WINCDEMO program. It is not included in the WINCDEMO program because it generates an error in MS VC++.

The DOS C example programs illustrate the form of the Universal Library syntax, but will not run from Borland C++.

cbAConvertData()

Description: Converts the raw data collected by cbAinScan() into 12 bit A/D values. The cbAinScan() function can return either raw A/D data or converted data depending on whether or not the CONVERTDATA option was used. The raw data is a 16 bit value that contains a 12 bit A/D value and a 4 bit channel tag. The converted data consists of just the 12 bit A/D value.

Summary: int cbAConvertData (long NumPoints, unsigned ADData[], int ChanTags[])

Arguments: NumPoints - Number of samples to convert
ADData - Pointer to data array
ChanTags - pointer to channel tag array

When you collect data with cbAinScan and you don't use the CONVERTDATA option then you must use this function to convert the data after it is collected. There are cases where the CONVERTDATA option is not allowed. For example - if you using both the DMAIO and BACKGROUND option with cbAinScan. In those cases this function should be used to convert the data after the data collection is complete.

Each raw data point consists of a 12 bit A/D value with a 4 bit channel number. This function pulls each data point apart and puts the A/D value into the ADData array and the channel number into the ChanTags array.

Returns: Error code or 0 if no errors
ADData - converted data
ChanTags - channel tags if available.

Note - 12 Bit A/D Boards
1) Name of array must match that used in cbAInScan.
2) Upon returning from cbAConvertData(), ADData array contains only 12 bit A/D data.

Note - 16 Bit A/D Boards
This function is not for use with 16 bit A/D boards.

cbAConvertPretrigData()

Description: Converts the raw data collected by cbAPretrigger(). The cbAPretrigger() function can return either raw A/D data or converted data depending on whether or not the CONVERTDATA option was used. The raw data as it is collected is not in the correct order. After the data collection is completed it must be rearranged into the correct order.

Summary: int cbAConvertPretrigData (long PreTrigCount, long Total Count, unsigned ADData[], int ChanTags[])

Arguments: PreTrigCount - Number of pre-trigger samples
Total Count - Total number of samples that were
ADData - Pointer to data array
ChanTags - Pointer to channel tag array

When you collect data with cbAPretrig() and you don't use the CONVERTDATA option then you must use this function to convert the data after it is collected. There are cases where the CONVERTDATA option is not allowed. For example - if you use the BACKGROUND option with cbAPretrig(). In those cases this function should be used to convert the data after the data collection is complete.

Each raw data point consists of a 12 bit A/D value with a 4 bit channel number. This function pulls each data point apart and puts the A/D value into the ADData and the channel number into the ChanTags array.

Returns: Error code or 0 if no errors
ADData - converted data

Revision 2 Windows language status - Not Available.

cbAIn()

Description: Reads an A/D input channel. This function reads the specified A/D channel from the specified board. If the specified A/D board has programmable gain then it sets the gain to the specified range. The raw A/D value is converted to an A/D value and returned to DataValue.

Summary: int cbAIn (int BoardNum, int Channel, int Range, unsigned *DataValue);

Arguments:
BoardNum - board number may be 0 to 9.
Channel - A/D channel number
Range - A/D Range code or 0
DataValue - Pointer to data value

Returns:
Error code or 0 if no errors
DataValue - Value of A/D sample returned here

BoardNum - refers to the board number associated with the board when it was installed with the InstaCal™ configuration program. The specified board must have an A/D.

Channel - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example, a CIO-DAS1600 has 8 chans for differential, 16 for single ended.

Range - If the selected A/D board does not have a programmable gain feature then this argument should always be set to 0. If the A/D board does have programmable gain then set the Range argument to the desired A/D range. Not all A/D boards support the same A/D ranges. Refer to the A/D board manual for a list of supported A/D Ranges.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI51VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

Revision 2 Windows language. Do not attempt to acquire large blocks of data with a Windows language by placing cbAIn in tight FOR:NEXT or other loops. You will hang the computer.

cbAInScan()

Description: Scans a range of A/D channels and stores the samples in an array. This function reads the specified number of A/D samples at the specified sampling rate from the specified range of A/D channels from the specified board. If the A/D board has programmable gain then it sets the gain to the specified range. The collected data is returned to Data Array.

Summary: int cbAInScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, unsigned ADData[], int Options)

Arguments:
BoardNum - board number may be 0 to 9.
LowChan - First A/D channel of scan
HighChan - Last A/D channel of scan
Count - Number of A/D samples to collect
Rate - Sample rate in scans per second
Range - A/D range code or 0
ADData - Data array to store A/D values in
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the InstaCal configuration program. The specified board must have an A/D.

Channel - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example, a CIO-DAS1600 has 8 chans for differential, 16 for single ended.

Count - Specifies the total number of A/D samples that will be collected. If more than one channel is being sampled then the number of samples collected per channel is equal to Count / (HighChan- LowChan+1).

Rate - This is the rate at which scans are triggered. If you are sampling 4 channels, 0-3, then specifying a rate of 10,000 scans per second (10KHz) will result in the A/D converter rate of 40KHz: 4 channels at 10,000 samples per channel per second. This is different from some software where you specify the total A/D chip rate. In those systems, the per channel rate is equal to the A/D rate divided by the number of channels in a scan.

Caution! You will generate an error if you specify a total A/D rate beyond the capability of the board. For example; if you specify rate LowChan = 0, HighChan = 7 (8 channels total) and Rate = 20,000 and you are using a CIO-DAS16/Jr, you will get an error. You have specified a total rate of 8*20,000 = 160,000. The CIO-DAS16/Jr is capable of converting 100,000 samples per second. The maximum sampling rate depends on the A/D board that is being used. It is also dependent on the sampling mode options.

Range - If the selected A/D board does not have a programmable range feature, then this argument should be set to 0. Otherwise the gain can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for the list of ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

ADData - The data array must be big enough to hold at least Count number of integers.

The Options field may contain any combination of non-contradictory choices from the following list:

DTCONNECT - All A/D values will be sent to the A/D board's DT CONNECT port.

Trigger & Transfer Method Options: If none of these options is specified (recommended), the optimum sampling mode will automatically be chosen based on board type and sampling speed.

FREERUN - You may not supply this argument in the options argument. It is the default cbAInScan A/D trigger and transfer method for boards which have no hardware trigger capability, such as the PPIO-AI8. This argument is not available to boards with hardware trigger methods. Because there is no hardware paced trigger used in FREERUN, the Rate value is meaningless. A/D conversions are triggered as fast as the computer will execute a tight software trigger & transfer. The rate achieved can be determined only by experimenting (timing) with the machine running the program you write.

SINGLEIO - A/D conversions and transfers to memory are initiated by an interrupt. One interrupt per conversion.

DMAIO - A/D conversions are initiated by a trigger. Transfers are initiated by a DMA request.

BLOCKIO - A/D conversions are initiated by a trigger. Transfers are handled by REP-INSW.

Revision 2 Windows language: Supports only a subset of FREERUN. The limitation is that (0 < Count < (HighChan - LowChan)). Do not attempt to acquire large blocks of data with a Windows language by placing cbAIn or cbAInScan in tight FOR:NEXT or other loops. You will hang the computer.

BURSTMODE - Enables burst mode sampling. The A/D is clocked at the maximum A/D rate between samples in order to minimize channel to channel skew. Scans are initiated at the rate specified by rate

CONVERTDATA - If the CONVERTDATA option is used then the data that is returned to ADData will automatically be converted to 12 bit A/D values. If CONVERTDATA is not used then the data from 12 bit A/D boartds will be returned as 16 bit values that will contain both a 12 bit A/D value and a 4 bit channel number. After the data collection is complete you can call cbAConvertData() to convert the data after the fact. CONVERTDATA may not be specified if you are using the BACKGROUND option and DMA transfers.

BACKGROUND - If the BACKGROUND option is not used then the cbAInScan() function will not return to your program until all of the requested data has been collected and returned to ADData. When the BACKGROUND option is used, control will return immediately to the next line in your program and the data collection from the A/D into ADData will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed.

CONTINUOUS - This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of ADData and begins again. The only way to stop this operation is with cbStopBackground(). Normally this option should be used in combination with BACKGROUND so that your program will regain control.

EXTCLOCK - If this option is used then conversions will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each conversion will be triggered on the rising edge of the trigger input signal. When this option is used the Rate argument is ignored. The sampling rate is dependent on the trigger signal.

EXTMEMORY/NORMMEMORY - NORMMEMORY is the default. Data is returned to a data buffer (array). EXTMEMORY causes the command to send the data to a connected memory board via the DT-Connect interface rather than returning the data to ADData. Everytime this option is used it overwrites any data already stored in the memory board. The data should be unloaded with the cbMemRead() function before collecting new data.

RETURNS: error code or 0 if no errors
Rate = actual sampling rate used
ADData - Collected A/D data returned here

cbALoadQueue()

Description: Loads A/D board's channel/gain queue. This function only works with A/D boards that have channel/gain queue hardware.

Summary: int cbALoadQueue (int BoardNum, int ChanArray[], int GainArray[], int Count);

Arguments: BoardNum - board number may be 0 to 9.
ChanArray - Array containing channel values
GainArray - Array containing A/D range values
Count - Number of elements in ChanArray and GainArray or 0 to disable chan/gain queue

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D and a channel/gain queue.

ChanArray - This array should contain all of the channels that will be loaded into the channel gain queue.

GainArray - This array should contain each of the A/D ranges that will be loaded into the channel gain queue.

Count - Specifies the total number of chan/gain pairs that will be loaded into the queue. ChanArray and GainArray should contain at least Count elements. Set Count=0 to disable the board's chan/gain queue.

Normally the cbAInScan() function scans a fixed range of channels (from LowChan to HighChan) at a fixed A/D range. If you load the channel gain queue with this function then all subsequent calls to cbAInScan() will cycle through the chan/range pairs that you have loaded into the queue.

Returns: error code or 0 if no errors

cbAOut()

Description: Sets the value of a D/A output

Summary: int cbAOut (int BoardNum, int Channel, unsigned DataValue)

Arguments: BoardNum - board number may be 0 to 9.
Channel - A/D channel number
DataValue - Value to set D/A to

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a D/A.

Channel - The maximum allowable channel depends on which type of D/A board is being used.

DataValue - Must be in the range 0 - N where N is the value 2 ^ Resolution - 1 of the converter.

Returns: Error code or 0 if no errors

Simultaneous Update Boards: If you have set the simultaneous update jumper for simultaneous operation, you should use cbAOutScan() for simultaneous update of multiple channels. cbAOut() always writes the D/A data then reads the D/A, which causes the D/A output to be updated.

cbAOutScan()

Description: Outputs values to a range of D/A channels

Summary: int cbAOutScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, unsigned DataBuffer[], int Options)

Arguments:
BoardNum - board number may be 0 to 9.
LowChan - First D/A channel of scan
HighChan - Last D/A channel of scan
Count - number of D/A values to output
Rate - Output rate (in scans/sec)
DataBuffer - Data array full of D/A values.
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a D/A.

LowChan/HighChan - The maximum allowable channel depends on which type of D/A board is being used. When using the SBX-DDA04 board you may specify the channel range with LowChan/HighChan OR you can set LowChan and HighChan = NOTUSED and specify the channel number in the 4 MSB's of the data values in DataBuffer. The D/A channels in the range specified by LowChan- HighChan are set from consecutive elements in DataBuffer.

Count - Specifies the total number of D/A values that will be output. Most D/A boards (all except for cSBX-DDA04) do not support timed outputs. For these boards count should be set to the number of channels in the scan.

Rate - For most D/A boards the Rate is ignored and can be set to NOTUSED. For D/A boards with trigger and transfer methods which allow fast output rates, such as the cSBX-DDA04 , Rate should be set to the D/A output rate (in scans/sec)

If supported, this is the rate at which scans are triggered. If you are updating 4 channels, 0-3, then specifying a rate of 10,000 scans per second (10KHz) will result in the D/A converter rates of 10KHz: (one D/A per channel). The data transfer rate will be 40,000 words per second; 4 channels * 10,000 updates per scan.

Caution! You will generate an error if you specify a total D/A rate beyond the capability of the board. For example; if you specify rate LowChan = 0, HighChan = 3 (4 channels total) and Rate = 100,000 and you are using a cSBX-DDA04, you will get an error. You have specified a total rate of 4*100,000 = 400,000. The cSBX-DDA04 is rated to 330,000 updates per second.

The maximum update rate depends on the D/A board that is being used. It is also dependent on the sampling mode options.

DataBuffer - The data array should be filled with D/A values in the range 0 - N where N is the value 2 ^ Resolution-1 of the converter. There should be at least HighChan-LowChan+1 elements in the array.

Options:

CONTINUOUS - This option may only be used with boards which support interrupt, DMA or REP-INSW transfer methods. This option puts the function in an endless loop. Once it outputs the specified (by Count) number of D/A values, it resets to the start of ADData and begins again. The only way to stop this operation is with cbStopBackground(). This option should only be used in combination with BACKGROUND so that your program can regain control.

BACKGROUND - This option may only be used with boards which support interrupt, DMA or REP-INSW transfer methods. When this option is used the D/A operations will begin running in the background and control will immediately return to the next line of your program. Use cbGetStatus() to check the status of background operation. Use cbStopBackground() to terminate background operations before they are completed.

SIMULTANEOUS - When this options is used (and the appropriate switches are set on the board) all of the D/A voltages will be updated simultaneously when the last D/A in the scan is updtaed. This generally means that all the D/A values will be written to the board, then a read of a D/A address causes all D/As to be updated with new values simultaneously.

Returns: Error code or 0 if no errors

Revision 2 Windows Languages. Options, BACKGROUND and SIMULTANEOUS, are ignored.

cbAPretrig()

Description: Waits for a trigger to occur and then returns a specified number of analog samples before and after the trigger occurred. The trigger input line (See board users manual but generally Digital In 0) must be at TTL low before this function is called or a TRIGSTATE error will occur. The trigger occurs on the rising edge of the trigger input signal (transition to TTL high).

Summary: int cbAPretrig (int BoardNum, int LowChan, int HighChan, long *PretrigCount, long *TotalCount, long *Rate, int Range, unsigned ADData[], int Options)

Arguments:
BoardNum - board number may be 0 to 9.
LowChan - First A/D channel of scan
HighChan - Last A/D channel of scan
PretrigCount - Number of pre-trigger A/D samples to collects
TotalCount - Total number of A/D samples to collects
Rate - Sample rate in scans per second
Range - A/D Range code or 0
ADData - Data array to store A/D values in.
IMPORTANT: the data array must be big enough to hold TotalCount+512 integers
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D.

LowChan/HighChan - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured (8 chans for differential, 16 for single ended).

PretrigCount - Specifies the number of samples before the trigger that will be returned. PretrigCount must be less than 32000 and PretrigCount must also be less than TotalCount-512. If the trigger occurs too early, then fewer then the requested number of pre-trigger samples will be collected. In that case a TOOFEW error will occur. The PretrigCount will be set to indicate how many samples were collected and the post trigger samples will still be collected.

TotalCount - Specifies the total number of samples that will be collected and stored in ADData. TotalCount must be greater than or equal to PretrigCount+512. If the trigger occurs too early then fewer than the requested number of samples will be collected. In that case a TOOFEW error will occur. The TotalCount will be set to indicate how many samples were actually collected.

Range - If the selected A/D board does not have a programmable gain feature then this argument should be set to 0. Otherwise the Range can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for a list of the A/D ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

ADData - VERY IMPORTANT: The data array must be big enough to hold at least TotalCount+512 integers

Options:

DTCONNECT - When DTCONNECT option is used with this function the data from ALL A/D conversions is sent out the DT-CONNECT interface. While this function is waiting for a trigger to occur, it will send data out the DT-CONNECT interface continuously. If you have a Computer Boards memory board plugged into the DT-CONNECT interface then you should use EXTMEMORY option rather than this option.

EXTMEMORY - If you use this option to send the data to a connected memory board then you must use cbMemReadPretrig() to later read the pre-trigger data from the memory board. If you use cbMemRead() the data will NOT be in the correct order. Everytime this option is used it will overwrite any data that is already stored in the memory board. All data should be read from the board (with cbMemReadPretrig()) before collecting any new data.

CONVERTDATA - The data is collected into a "circular" buffer. When the data collection is complete the data is in the wrong order. If you use the CONVERTDATA option then the data will be automatically rotated into the correct order and converted to 12 bit values when the data acquisition is complete. Otherwise you must call cbAConvertPretrigData() to rotate the data. You can not use the CONVERTDATA option in combination with the BACKGROUND operation.

BACKGROUND - If the BACKGROUND option is not used then the cbAPretrig() function will not return to your program until all of the requested data has been collected and returned to ADData. When the BACKGROUND option is used, control will return immediately to the next line in your program and the data collection from the A/D into ADData will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed. You can not use the BACKGROUND option with the CONVERTDATA option.

Returns: error code or 0 if no errors
PretrigCount - Number of pre-trigger samples
TotalCount - Total number of samples collected
Rate = actual sampling rate
ADData - Collected A/D data returned here

Revision 2 Windows languages. This function is not available.

cbATrig()

Description: Waits for a specified analog input channel to go above or below a specified value. This function continuously reads the specified channel and compares its value to TrigValue. Depending on whether TrigType is ABOVE or BELOW it waits for the first A/D sample that is above or below TrigType. It returns the first sample that meets the trigger criteria to DataValue.

Summary: int cbATrig (int BoardNum, int Chan, int TrigType, int TrigValue, int Range, unsigned *DataValue)

Arguments:
BoardNum - board number may be 0 to 9.
Chan - A/D channel number (0-7 for differential input, 0-15 for single ended)
TrigType - TRIGABOVE or TRIGBELOW - Specifies whether waiting for the analog input to be ABOVE or BELOW the specified trigger value.
TrigValue - The threshold value that all A/D values are compared to.
Range - Gain code or 0
DataValue - The value of the first A/D sample that met the trigger criteria is returned here.

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D.

Chan - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured. For example a CIO-DAS1600 has 8 chans for differential, 16 for single ended.

TrigValue - Must be in the range 0 - 4095 for 12 bit A/D boards, or 0-65,535 for 16 bit A/D boards.

Range - If the selected A/D board does not have a programmable gain feature then this argument should be set to 0. Otherwise the Range can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for a list of the A/D ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5IVOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

Returns: - Error code or 0 if no errors

COUNTERS - AN INTRODUCTION

Universal Library provides functions for initialization and configuration of counter chips. It is important to note what this means:

1) Universal Library can configure a counter for any of the counters operations.

2) Counter configuration does not include USE of counters such as event counting and pulse width. Counter use is accomplished by programs which use the counter functions.

3) Some counter USE functions may be available. A function cbFreqIn() is provided (Revision 1 on). Others may be added to later revisions.

For you to use a counter for any but the simplest counting function, you must have, read, understand and employ the information contained in the chip manufacturer's data sheet. Technical support of the Universal Library does not include providing, interpreting or explaining the counter chip data sheet.

Counter chip data sheets are available from:

82C54

Intel Corporation
Literature Department
(800) 548-4725
Request part No. 230843, Micro and Peripherals Handbook, a 2 volume set.
As of this writing there is no charge for this set.

AM9513

Advanced Micro Devices, Inc.
901 Thompson Place
P.O. Box 3453
Sunnyvale, CA 94088
(408) 732-2400

As of this writing the data book, the Am9513 Technical Manual is available free of charge.

Z8536

As of this writing, the only ComputerBoards part the employs the Z8536 is the CIO-INT32. The data book for the chip is included with the CIO-INT32.

Counter Chip Variables

Universal Library counter initialization and configuration functions include names for bit patterns, such as ALEGATE, which stands for Active Low Enabled Gate N. In any case where Universal Library has a name for a bit pattern, it is allowed to substitute the bit pattern as a numeric. This will work, but your programs will be harder to read and debug.

cbC8254Config()

Description: Configures 8254 counter for desired operation. There are three types of counter chips used by Computer Boards - 8254, 8536 and 9513 chips. This function can only be used with 8254 counters.

Summary: int cbC8254Config (int BoardNum, int CounterNum, int Config)

Arguments: BoardNum - board number may be 0 to 9.
CounterNum - counter number to configure
Config - the action to take on terminal count and the wave form if any.

EXPLANATION OF THE VARIABLES:

CounterNum - Selects one of the counter channels. An 8254 has 3 counters. The value may be 1, 2 or 3.

Config - Refer to the 8254 data sheet for a detailed description of each of the configurations. May be set to one of the following constants:

HIGHONLASTCOUNT: Output of counter (OUT N) transitions from low to high on terminal count and remains high until reset. See Mode0 on 8254 data sheet.

ONESHOT: Output of counter (OUT N) transitions from high to low on rising edge of GATE N, then back to high on terminal count. See mode 1 on 8254 data sheet

RATEGENERATOR: Output of counter (OUT N) pulses low for one count on terminal count and reloads the counter and re-cycles. See mode 2 on 8254 data sheet.

SQUAREWAVE : Output of counter (OUT N) is high for count < 1/2 terminal count then low until terminal count, whereupon it recycles. This mode generates a square wave. See mode 3 on 8254 data sheet.

SOFTWARESTROBE : Output of counter (OUT N) pulses low for one clock cycle on terminal count. Count starts afetr counter is loaded. See mode 4 on 8254 data sheet.

HARDWARESTROBE : Output of counter (OUT N) pulses low for one clock cycle on terminal count. Count starts on rising edge at GATE N input. See mode 5 on 8254 data sheet.

Returns: Error code or 0 if no errors

cbC8536Config()

Description: Configures 8536 counter for desired operation. There are three types of counter chips used by Computer Boards - 8254, 8536 and 9513 chips. This function can only be used with 8536 counters.

Summary: int cbC8536Config (int BoardNum, int CounterNum, int OutputControl, int RecycleMode, int Retrigger)

Arguments: BoardNum - board number may be 0 to 9.
CounterNum - counter number to configure
OutputControl - Specifies counter output signal used.
RecycleMode - Execute once or reload and rexecute until stopped.
Retrigger - Enable or disable retriggering.

EXPLANATION OF THE VARIABLES:

CounterNum - Selects one of the counter channels. An 8536 has 3 counters. The value may be 1, 2 or 3.

OutputControl - Specifies the action of the output signal. The options for this argument are:
HIGHPULSEONTC - Output will transition from low to high for one clock pulse on terminal count.
TOGGLEONTC - Output will change state on terminal count.
HIGHUNTILTC - Output will transition to high at the start of counting then go low on terminal count.

RecycleMode - If set to RECYCLE (as opposed to ONETIME) then the counter will automatically reload to the starting count every time it reaches 0, then counting will continue.

Retrigger - If set to ENABLED then every trigger on the counter's trigger input will initiate loading of the initial count and counting will proceed from initial count.

Returns: Error code or 0 if no errors

cbC8536Init()

Description: Initializes the counter linking features of an 8536 counter chip. See the 8536 data sheet Counter/Timer Link Controls section for a complete description of the hardware affected by this mode. The linking of counters 1 & 2 must be accomplished prior to enabling the counters.

Summary: int cbC8536Init (int BoardNum, int ChipNum, int CtrlOutput)

Arguments: BoardNum - board number may be 0 to 9.
ChipNum - chip number to configure
CtrlOutputI - Specifies counter output signal used.

EXPLANATION OF THE VARIABLES:

BoardNum - Board number of board with 8536 counter installed.

ChipNum - Selects one of the 8536 chips on the board, 1 to n.

CtrlOutput - Specifies how the counter 1 is to be linked to counter 2, if at all. The options for this argument are:
NOTLINKED -Counter 1 is not connected to any other counters inputs.
GATECTR2 - Output of counter 1 is connected to the GATE of counter #2.
TRIGCTR2 - Output of counter 1 is connected to the trigger of counter #2.
INCTR2 - Output of counter 1 is connected to counter #2 clock input.

Returns - Error code or 0 if no errors.

cbC9513Config()

Description: Sets all of the configurable options of a 9513 counter.

Summary: int cbC9513Config (int BoardNum, int CounterNum, int GateControl, int CounterEdge, int CountSource, int SpecialGate, int Reload, int RecycleMode, int BCDMode, int CountDirection, int OutputControl)

Arguments:
BoardNum - board number may be 0 to 9.
CounterNum - counter number (1 - n)
GateControl - gate control
CountEdge - which edge to count
CountSource - which of the available count sources to use.
SpecialGate - special gate may be enabled or disabled.
Reload - Load or load and hold.
RecycleMode - Execute once or reload and recycle.
BCDMode - Counter may operate in Binary Coded Decimal if desired.
CountDirection - AM9513 may count up or down.
OutputControl - The type of output desired.

EXPLANATION OF THE VARIABLES:

ADVICE: *The information provided here and in cb9513Init() will only help you understand how Universal Library syntax corresponds to the 9513 data sheet. It is not a substitute for the data sheet. You cannot program and use a 9513 without the manufacturers' data book.*

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

CounterNum - counter number (1 - n) where n is the number of counters on the board. A CIO-CTR5 has 5, a CIO-CTR10 has 10.

GateControl - gate control variables are:

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
NOGATE	No gating
AHLTCPREVCTR	Active high TCN-1
AHLNEXTGATE	Active High Level GATE N + 1
AHLPREVGATE	Active High Level GATE N - 1
AHLGATE	Active High Level GATE N
ALLGATE	Active Low Level GATE N
AHEGATE	Active High Edge GATE N
ALEGATE	Active Low Edge GATE N

CountEdge - which edge to count. Refered to Source Edge in 9513 data book.

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
POSITIVEEDGE	Count on Rising Edge
NEGATIVEEDGE	Count on Falling Edge

CountSource -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description	
TCPREVCTR	TCN - 1	(Terminal count of previous counter)
CTRINPUT1	SRC 1	(Counter Input 1)
CTRINPUT2	SRC 2	(Counter Input 2)
CTRINPUT3	SRC 3	(Counter Input 3)
CTRINPUT4	SRC 4	(Counter Input 4)
CTRINPUT5	SRC 5	(Counter Input 5)
GATE1	GATE 1	
GATE2	GATE 2	
GATE3	GATE 3	
GATE4	GATE 4	

GATE5	GATE 5
FREQ1	F1
FREQ2	F2
FREQ3	F3
FREQ4	F4
FREQ5	F5

SpecialGate -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
ENABLED	Enable Special Gate
DISABLED	Disable Special Gate

Reload -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
LOADREG	Reload from Load
LOADANDHOLDREG	Reload from Load or Hold except in Mode X which reloads only from Load

RecycleMode -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
ONETIME	Count Once
RECYCLE	Count Repetitively

BCDMode -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
DISABLED	Binary Count
ENABLED	BCD Count

CountDirection -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
COUNTDOWN	Count Down
COUNTUP	Count Up

OutputControl -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
ALWAYSLOW	Inactive, Output Low
HIGHPULSEONTC	Active High Terminal Count Pulse
TOGGLEONTC	TC Toggled
DISCONNECTED	Inactive, Output High Impedence
LOWPULSEONTC	Active Low Terminal Count Pulse
3, 6, 7 (numeric values)	Illegal

Returns: error code or 0 if no errors

cbC9513Init()

Description: Initializes all of the chip level features of a 9513 counter chip. This function can only be used with 9513 counters.

Summary: int cbC9513Init (int BoardNum, int ChipNum, int FoutDivider, int FoutSource, int Compare1, int Compare2, int TimeOfDay)

Arguments:
BoardNum - board number may be 0 to 9.
ChipNum - Specifies which 9513 chip is to be initialized.
FoutDivider - F-Out divider (0-15)
FoutSource - Specifies source of the signal for F-Out signal.
Compare1 - ENABLED or DISABLED
Compare2 - ENABLED or DISABLED
TimeOfDay - DISABLED or 1-3

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

ChipNum - Specifies which 9513 chip is to be initialized. For a CTR05 board this should be set to 1. For a CTR10 board it should be either 1 or 2.

FoutDivider -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
0	Divide by 16
1	Divide by 1
2 ... 15	Divide by the number 2 ... 15

FoutSource -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description	
TCPREVCTR	TCN - 1	(Terminal count of previous counter)
CTRINPUT1	SRC 1	(Counter Input 1)
CTRINPUT2	SRC 2	(Counter Input 2)
CTRINPUT3	SRC 3	(Counter Input 3)
CTRINPUT4	SRC 4	(Counter Input 4)
CTRINPUT5	SRC 5	(Counter Input 5)
GATE1	GATE 1	
GATE2	GATE 2	
GATE3	GATE 3	
GATE4	GATE 4	
GATE5	GATE 5	
FREQ1	F1	
FREQ2	F2	
FREQ3	F3	
FREQ4	F4	
FREQ5	F5	

Compare1 -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
DISABLED	Disabled
ENABLED	Enabled

Compare2 -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
DISABLED	Disabled
ENABLED	Enabled

TimeOf Day -

Universal Library Syntax	Corresponds to 9513 description in Counter Mode Register Description
DISABLED	TOD Disabled
1	TOD Enabled / 5 Input
2	TOD Enabled / 6 Input
3	TOD Enabled / 10 Input

No Arguments - For:

Universal Library Set To	Corresponds to 9513 description in Counter Mode Register Description
0 (FOUT on)	FOUT Gate
0 (Data bus matches board)	Data Bus Width
1 (Disable Increment)	Data Pointer Control
1 (BCD Scaling)	Scalar Control

Returns: Error code or 0 if no errors

cbCFreqIn()

Description: Measures the frequency of a signal. This function can only be used with 9513 counters. This function uses internal counters #5 and #4.

Summary: int cbCFreqIn (int BoardNum, int SigSource, int GateInterval, unsigned *Count, long *Freq)

Arguments: BoardNum - board number may be 0 to 9.
SigSource - specifies which signal will be measured
GateInterval - gating interval in milliseconds
Count - The raw count is returned here
Freq - the measured frequency in Hz returned here.

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

SigSource - specifies which signal will be measured

CTRINPUT1	Counter input to counter 1. CTR1
CTRINPUT2 .. 10	Counter inputs 2 through 10
GATE1 .. 10	Gate 1 through 10
FREQ1 .. 10	Frequency source 1 through 10

The counters in a single chip are numbered CTR1 through CTR5. On a board with 2 9513 chips, the second 9513 chip holds counters 6-10.

GateInterval - specifies the time (in milliseconds) that the counter will be counting. The optimum GateInterval depends on the frequency of the measured signal. The counter can count up to 65535.

If the gating interval is too low, then the count will be too low and the resolution of the frequency measurement will be poor. For example, if the count changes from 1 to 2 the measured frequency doubles.
If the gating interval in too long then the counter will overflow and a FREQOVERFLOW error will occur.

SigSource - CTRINPUTx refers to the external counter inputs on the board connector. All of the other choices refer to internal signals within the 9513 counter chip. Refer to the 9513 data sheet for details on these signal sources.

Returns: Error code or 0 if no errors
Count - Count that frequency calculation based on returned here
Freq - Measured frequency in Hz returned here

cbCIn()

Description: Reads the current count from a counter

Summary: int cbCIn (int BoardNum, int CounterNum, unsigned int *Count)

Arguments: BoardNum - board number may be 0 to 9.
CounterNum - counter number to read
Count - Count returned here

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a counter.

CounterNum - The counter to read current count from. Valid values are 1 to 10, up to the number of counters on the board.

1	Counter input to counter 1. CTR1
2 .. 10	Counter inputs 2 through 10

Returns: Error code or 0 if no errors
Count - counter value returned here . The range of counter values returned are:
0 to 65,535 for C or PASCAL languages.
-32,768 to 32,767 for BASIC languages. BASIC counts, and therefore loads counters as:

-1	loads as	65,535	
-32,768	loads as	32,768	
32,767	loads as	32,767	
2	loads as	2	Note: You cannot load a counter with less than 2.

cbCLoad()

Description: Loads the specified counter's LOAD, HOLD or ALARM register with a count. When you want to load a counter with a value to count from, it is never loaded directly into the counter's count register. It is loaded into the load register. From there, the counter, once enebled, loads the count from the hold register, generally on the first valid pulse.

Summary: int cbCLoad (int BoardNum, int RegName, unsigned int LoadValue)

Arguments: BoardNum - board number may be 0 to 9.
RegName - Register to load LoadValue into.
LoadValue - Value to be loaded into RegName

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a counter.

RegName - The counter to read current count from. Valid values are:

LOADREG1 .. 10	Load registers 1 through 10. This may span several chips.
HOLDREG1 .. 10	Hold registers 1 through 10. This may span several chips. (9513 only)
ALARMREG1CHIP1	Alarm register 1 of the first counter chip. (9513 only)
ALARMREG2CHIP1	Alarm register 2 of the first counter chip. (9513 only)
ALARMREG1CHIP2	Alarm register 1 of the second counter chip. (9513 only)
ALARMREG2CHIP2	Alarm register 2 of the second counter chip. (9513 only)

LoadValue - The value to be loaded. Must be bewteen 0 and 2 ^ resolution - 1 of the counter. For example, a 16 bit counter is 2 ^ 16 - 1, or 65,535.

Returns: Error code or 0 if no errors

Counter Types: There are several counter types supported. Please refer to the data sheet for the registers available for a counter type.

cbCStoreOnInt()

Description: Installs an interrupt handler that will store the current count whenever an interrupt occurs. This function can only be used with 9513 counters.

Summary: int cbCStoreOnInt (int BoardNum, int IntCount, int CntrControl[], unsigned DataBuffer[])

Arguments: BoardNum - board number may be 0 to 9.
IntCount - Number of interrupts
CntrControl - Array with each element set to either ENABLED or DISABLED
DataBuffer - Array where counts will be stored

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have a 9513 counter.

IntCount - The counters will be read every time an interrupt occurs until IntCount interrupts have occurred.

CntrControl - The array should have an element for each counter on the board. (5 elements for CTR-05 board, 10 elements for a CTR-10). Each element corresponds to a possible counter channel. Each element should be set to either DISABLED or ENABLED. All channels that are set to ENABLED will be read when an interrupt occurs.

DataBuffer - The array should have an element for each counter on the board. (5 elements for CTR-05 board, 10 elements for a CTR-10). Each element corresponds to a possible counter channel. Each channel that is marked as ENABLED in the CntrControl array will be read when an interrupt occurs. The count value will be stored in the DataBuffer element associated with that channel.

Returns: Error code or 0 if no errors

Revision 2 Windows Languages: Not suported.

cbDBitIn()

Description: Reads the state of a single digital input bit. This function treats all of the DIO ports on a boards as a single very large port. It lets you read the state of any individual bit within this large port. You must use cbDConfig() to configure the port for input first.

Summary: int cbDBitIn (int BoardNum, int PortType, int BitNum, int *BitValue);

Arguments: BoardNum - Board number may be 0 to 9.
PortType - Specifies which type of digital port to read
BitNum - Specifies which bit to read BitValue
BitValue - Place holder for return value of bit - the bit's value (0 or 1) is returned here

EXPLANATION OF THE VARIABLES:

PortType- There are two general types of digital I/O - 8255 and other. Some boards (DIO Series) use an 8255 for digital I/O. For these boards PortType should be set to FIRSTPORTA. Other boards don't use 8255. For these boards PortType should be set to AUXPORT. Some boards have both types of digital I/O (DAS1600). Set PortNum to either FIRSTPORTA or AUXPORT depending on which digital inputs you wish to read.

BitNum - This specifies the bit number within the single large port. The specified bit must be in a port that is currently configured as an input.

The tables below show which bit numbers are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

82C55 Bit#	Chip #	Address
0 - 23	1	Base + 0
24 - 47	2	Base + 4
48 - 71	3	Base + 8
72 - 96	4	Base + 12
96 - 119	5	Base + 16
120 - 143	6	Base + 20
144 - 167	7	Base + 24
168 - 191	8	Base + 28

8536 Bit#	Chip #	Address
0 - 19	1	Base + 0
20 - 39	2	Base + 4

BitValue - Place holder for return value of bit. Value will be 0 or 1. A 0 indicates a low reading, a 1 indicates a logic high reading. Logic high does not necessarily mean 5V. See the board manual for chip input specifications.

Returns: Error code or 0 if no errors
BitValue - value (0 or 1) of specified bit returned here

cbDBitOut()

Description: Sets the state of a single digital output bit. This function treats all of the DIO chips on a board as a single very large port. It lets you set the state of any individual bit within this large port. You must use cbDConfig() to configure the port for output first.

Summary: int cbDBitOut (int BoardNum, int PortType, int BitNum, int BitValue);

Arguments: BoardNum - board number may be 0 to 9.
PortNum - Specifies which digital port (AUXPORT, FIRSTPORTA).
BitNum - specifies which bit to write
BitValue - the bit's value (0 or 1)

EXPLANATION OF THE VARIABLES:

PortType - There are two general types of digital I/O - 8255 and other. Some boards (DIO Series) use an 8255 for digital I/O. For these boards PortType should be set to FIRSTPORTA. Other boards don't use 8255. For these boards PortType should be set to AUXPORT. Some boards have both types of digital I/O (CIO-DAS1600). Set PortNum to either FIRSTPORTA or AUXPORT depending on which digital inputs you wish to read.

BitNum - This specifies the bit number within the single large port. The specified bit must be in a port that is currently configured as an output.

The tables below show which bit numbers are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

82C55 Bit#	Chip #	Address
0 - 23	1	Base + 0
24 - 47	2	Base + 4
48 - 71	3	Base + 8
72 - 96	4	Base + 12
96 - 119	5	Base + 16
120 - 143	6	Base + 20
144 - 167	7	Base + 24
168 - 191	8	Base + 28

8536 Bit#	Chip #	Address
0 - 19	1	Base + 0
20 - 39	2	Base + 4

BitValue - The value to set the bit to. Value will be 0 or 1. A 0 indicates a logic low output, a 1 indicates a logic high output. Logic high does not necessarily mean 5V. See the board manual for chip specifications.

Returns: Error code or 0 if no errors

cbDConfigPort()

Description: Configures a digital port as Input or Output. This mode is for use with 82C55 chips and 8536 chips. See the board user's manual for details of chip operation.

Summary: int cbDConfigPort (int BoardNum, int PortNum, int Direction)

Arguments: BoardNum - board number may be 0 to 9.
PortNum - Specifies which digital I/O port to configure.
Direction - DIGITALOUT or DIGITALIN

EXPLANATION OF THE VARIABLES:

PortNum - The specified port must be configurable. The AUXPORT is not configurable. It is always configured for inputs and outputs.

The tables below show which ports and bit numbers are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

Mnemonic	Bit#	8255 Chip #	Chip Address	8536 Chip #	Chip Address
FIRSTPORTA	0 - 7	1A	Base + 0	1A	Base + 0
FIRSTPORTB	8 - 15	1B		1B	
FIRSTPORTCL	16 - 19	1CL		1C	
FIRSTPORTCH	20 - 23	1CH		Not present	
SECONDPORTA	24 - 31	2A	Base + 4	2A	Base + 4
SECONDPORTB	32 - 39	2B		2B	
SECONDPORTCL	40 - 43	2CL		2C	
SECONDPORTCH	44 - 47	2CH		No port C High in 8536 chips	

and so on to the last chip on the board as: THIRDPORT@, FOURTHPORT@, FIFTHPORT@, SIXTHPORT@, SEVENTHPORT@ and

EIGHTHPORTA	168 - 175	8A	Base + 28
EIGHTHPORTB	176 - 183	8B	
EIGHTHPORTCL	184 - 187	8CL	
EIGHTHPORTCH	188 - 191	8CH	

Direction - DIGITALOUT or DIGITALIN configures entire eight or four bit port for output or input.

Returns: Error code or 0 if no errors

cbDIn()

Description: Reads a digital input port

Summary: int cbDIn (int BoardNum, int PortNum, int *DataValue);

Arguments: BoardNum - board number may be 0 to 9.
PortNum - Specifies which digital I/O port to read.
*DataValue - Digital input value returned here.

EXPLANATION OF THE VARIABLES:

PortNum - The specified port must be configured for input. The AUXPORT is not configurable.

The tables below show which ports are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

Mnemonic	8255 Chip #	Chip Address	8536 Chip #	Chip Address
FIRSTPORTA	1A	Base + 0	1A	Base + 0
FIRSTPORTB	1B		1B	
FIRSTPORTCL	1CL		1C	
FIRSTPORTCH	1CH		Not present	
SECONDPORTA	2A	Base + 4	2A	Base + 4
SECONDPORTB	2B		2B	
SECONDPORTCL	2CL		2C	
SECONDPORTCH	2CH		No port C High in 8536 chips	

and so on to the last chip on the board as: THIRDPORT@, FOURTHPORT@, FIFTHPORT@, SIXTHPORT@, SEVENTHPORT@ and

EIGHTHPORTA	8A	Base + 28
EIGHTHPORTB	8B	
EIGHTHPORTCL	8CL	
EIGHTHPORTCH	8CH	

The size of the ports vary. If it is an eight bit port then the returned value will be in the range 0- 255. If it is a four bit port the value will be in the range 0 -15.

Returns: Error code or 0 if no errors
*DataValue - Digital input value returned here

IMPORTANT NOTE: Be sure to look at the example programs and the board specific information contained elsewhere in this manual for clarification of valid PortNum values.

cbDInScan()

Description: Multiple reads of digital input port of a high speed digital port on a board with a pacer clock. As of this revision of the manual and software, that is the CIO-PDMA16 only.

Summary: int cbDInScan (int BoardNum, int PortNum, long Count, long *Rate, int DataBuffer[], int Options);

Arguments:
BoardNum - board number may be 0 to 9.
PortNum - Specifies which digital I/O port to read
Count - number of times to read digital input
*Rate - Number of times per second (Hz) to read
DataBuffer - Digital input values returned here.
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

PortNum - Specifies which digital I/O port to read. The two choices are FIRSTPORTA or FIRSTPORTB. The specified port must be configured as an input.

Count - The number of times to read digital input

***Rate** - Number of times per second (Hz) to read the port. The actual sampling rate in some cases will vary a small amount from the requested rate. The actual rate will be returned to the Rate argument.

DataBuffer - The data array must be at least large enough to hold Count integers and must be capable of holding 16 bit values if the Option WORDXFER is specified.

Options:
BACKGROUND - If the BACKGROUND option is not used then the cbDInScan() function will not return to your program until all of the requested data has been collected and returned to DataBuffer.

When the BACKGROUND option is used, control will return immediately to the next line in your program and the transfer from the digital input port to DataBuffer will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed.

CONTINUOUS - This option puts the function in an endless loop. Once it transfers the required number of bytes it resets to the start of DataBuffer and begins again. The only way to stop this operation is with cbStopBackground(). Normally this option should be used in combination with BACKGROUND so that your program will regain control.

EXTCLOCK - If this option is used then transfers will be controlled by the signal on the trigger input line rather then by the internal pacer clock. Each transfer will be triggered on the rising edge of the trigger input signal. When this option is used the Rate argument is ignored. The transfer rate is dependent on the trigger signal.

WORDXFER - Normally this function reads a single (byte) port. If WORDXFER is specified then it will read two adjacent ports on each read and store the value of both ports together as the low and high byte of a single array element in DataBuffer[].

Transfer Method - May not be specified. DMA is used.

Returns:
Error code or 0 if no errors.
*Rate - Actual sampling rate returned here.
DataBuffer[] - Digital input value returned here.

Revision 2 Windows Languages: Option BACKGROUND and CONTINUOUS not supported. *Rate argument is ignored. Samples are clocked as fast as the instructions may be executed. Transfer Method is by polled I/O only.

cbDOut()

Description: Writes a byte to a digital output port .
Summary: int cbDOut (int BoardNum, int PortNum, int DataValue);

Arguments: BoardNum - board number may be 0 to 9.
PortNum - Specifies which digital I/O port to write.
DataValue - Digital output byte value

EXPLANATION OF THE VARIABLES:

PortNum - The specified port must be configured for output. The AUXPORT is not configurable.

The tables below show which ports are in which 82C55 and 8536 digital chips. The most 82C55 chips on a single board is eight (8), on the CIO-DIO196. The most (2) 8536 chips occur on the CIO-INT32.

Mnemonic	8255 Chip #	Chip Address	8536 Chip #	Chip Address
FIRSTPORTA	1A	Base + 0	1A	Base + 0
FIRSTPORTB	1B		1B	
FIRSTPORTCL	1CL		1C	
FIRSTPORTCH	1CH		Not present	
SECONDPORTA	2A	Base + 4	2A	Base + 4
SECONDPORTB	2B		2B	
SECONDPORTCL	2CL		2C	
SECONDPORTCH	2CH		No port C High in 8536 chips	

and so on to the last chip on the board as: THIRDPORT@, FOURTHPORT@, FIFTHPORT@, SIXTHPORT@, SEVENTHPORT@ and

EIGHTHPORTA	8A	Base + 28
EIGHTHPORTB	8B	
EIGHTHPORTCL	8CL	
EIGHTHPORTCH	8CH	

The size of the ports varies. If it is an eight bit port then the output value should be in the range 0- 255. If it is a four bit port the value should be in the range 0 - 15.

Returns: Error code or 0 if no errors.

IMPORTANT NOTE: Be sure to look at the example programs and the board specific information contained elsewhere in this manual for clarification of valid PortNum values.

cbDOutScan()

Description: Multiple writes to digital output port of a high speed digital port on a board with a pacer clock. As of this revision of the manual and software, that is the CIO-PDMA16 only.

Summary: int cbDOutScan (int BoardNum, int PortNum, long Count, long *Rate, int DataBuffer[], int Options);

Arguments:
BoardNum - board number may be 0 to 9.
PortNum - Specifies which digital I/O port to read
Count - number of times to read digital input
*Rate - Number of times per second (Hz) to read
DataBuffer - Digital input values returned here.
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

PortNum - Specifies which digital I/O port to read. The two choices are FIRSTPORTA or FIRSTPORTB. The specified port must be configured as an input.

Count - The number of times to read digital input

***Rate** - Number of times per second (Hz) to write to the port. The actual update rate in some cases will vary a small amount from the requested rate. The actual rate will be returned to the Rate argument.

DataBuffer - The data array must be at least large enough to hold Count integers and must be capable of holding 16 bit values if the Option WORDXFER is specified.

Options:
BACKGROUND - If the BACKGROUND option is not used then the cbDInScan() function will not return to your program until all of the requested data has been output.

When the BACKGROUND option is used, control will return immediately to the next line in your program and the transfer to the digital output port from DataBuffer will continue in the background. Use cbGetStatus() to check on the status of the background operation. Use cbStopBackground() to terminate the background process before it has completed.

CONTINUOUS - This option puts the function in an endless loop. Once it transfers the required number of bytes it resets to the start of DataBuffer and begins again. The only way to stop this operation is with cbStopBackground(). Normally this option should be used in combination with BACKGROUND so that your program will regain control.

EXTCLOCK - If this option is used then transfers will be controlled by the signal on the trigger input line rather then by the internal pacer clock. Each transfer will be triggered on the rising edge of the trigger input signal. When this option is used the Rate argument is ignored. The transfer rate is dependent on the trigger signal.

WORDXFER - Normally this function reads a single (byte) port. If WORDXFER is specified then it will read two adjacent ports on each read and store the value of both ports together as the low and high byte of a single array element in DataBuffer[].

Transfer Method - May not be specified. DMA is used.

Returns:
Error code or 0 if no errors.
*Rate - Actual sampling rate returned here.
DataBuffer[] - Digital input value returned here.

Revision 2 Windows Languages: Option BACKGROUND and CONTINUOUS not supported. *Rate argument is ignored. Samples are clocked as fast as the instructions may be executed. Transfer Method is by polled I/O only.

cbErrHandling()

Description: Sets the error handling for all subsequent function calls. All functions return error codes after each call. In addition other error handling features have been built into the library. This function controls those features. If the Universal Library cannot find the configuration file CB.CFG, it always terminates the program regardless of the cbErrHandling() setting.

Summary: void cbErrHandling (int ErrReporting, int ErrHandling)

Arguments: ErrReporting - type of error reporting.
ErrHandling - type of error handling.

EXPLANATION OF THE VARIABLES:

Warnings vs Fatal Errors - All errors that can occur are classified as either "warnings" or "fatal". Errors that can occur in normal operation in a bug free program (disk is full, too few samples before trigger occurred) are classified as "warnings". All other errors indicate a more serious problem and are classified as "fatal".

ErrReporting - This argument controls when the library will print error messages on the screen. The default is DONTPRINT If it is set to:

DONTPRINT - Errors will not generate a message to the screen. In that case your program must always check the returned error code after each library call to determine if an error occurred.

PRINTWARNINGS - Only warning errors will generate a message to the screen. Your program will have to check for fatal errors.

PRINTFATAL - Only fatal errors will generate a message to the screen. Your program must check for warning errors.

PRINTALL - All errors will generate a message to the screen.

ErrHandling - This argument specifies what class of error will cause the program to halt. The default is DONTSTOP. If ErrReporting is set to:

DONTSTOP - The program will always continue executing when an error occurs.

STOPFATAL - The program will halt if a "fatal" error occurs.

STOPALL - Will stop whenever any error occurs.

Returns: - None.

cbFileAInScan()

Description: Scan a range of A/D channels and store the samples in a disk file. This function reads the specified number of A/D samples at the specified sampling rate from the specified range of A/D channels from the specified board. If the A/D board has programmable gain then it sets the gain to the specified range. The collected data is returned to a file. cbFilexxx() functions are only for boards with REP-INSW transfer capability.

Summary: int cbFileAInScan (int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, char *FileName, unsigned Options)

Arguments:
BoardNum - board number may be 0 to 9.
LowChan - First A/D channel of scan
HighChan - Last A/D channel of scan
Count - Number of samples to collect
Rate - Sample rate in samples per second (Hz)
Range - Range code or 0
FileName - Name of disk file
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D.

Channel - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured (8 chans for differential, 16 for single ended).

Count - Specifies the total number of A/D samples that will be collected. If more than one channel is being sampled then the number of samples collected per channel is equal to Count / (HighChanLowChan+1).

Rate - The maximum sampling rate depends on the A/D board that is being used.

Range - If the selected A/D board does not have a programmable range feature, then this argument should be set to 0. Otherwise the gain can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for the list of ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI51VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts
BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

FileName - The name file must already exist. It should have been previously created with the MAKESTRM.EXE program.

Options:
TIMED - The A/D rate is driven by the on-board pacer clock at the rate specified in the Rate argument. This is the default.

EXTCLOCK - If this option is used then conversions will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each conversion will be triggered on the rising edge of the trigger input signal. When this option is used the Rate argument is ignored. The sampling rate is dependent on the trigger signal.

DTCONNECT - Samples are sent to the DT-Connect port if the board is equipped with one.

NODTCONNECT - Samples are not output to the DT-Connect port, if any. This is the default.

Returns: Error code or 0 if no errors
*Rate = actual sampling rate

OVERRUN Error - This error indicates that the data was not written to the file as fast as the data was sampled. Consequently some data was lost. The value in *Count will be the number of points that were successfully collected.

Revision 2 Windows languages - Not supported.

cbFileGetInfo()

Description: Returns information about a streamer file. When cbFileAInScan() or cbFilePretrig() fills the streamer file, information is stored about how the data was collected (sample rate, channels sampled etc.). This function returns that information. cbFilexxx() functions are only for boards with REP-INSW transfer capability.

Summary: int cbFileGetInfo (char *FileName, int *LowChan, int *HighChan, long *PreTrigCount, int *TotalCount, long *Rate, int *Range)

Arguments:
FileName - Name of streamer file
LowChan - Variable to return LowChan to
HighChan - Variable to return HighChan to
TotalCount - Variable to return TotalCount to
PreTrigCount - Variable to return PreTrigCount to
Rate - Variable to return sampling rate to
Range - Variable to return A/D range code to

Returns:
Error code or 0 if no errors
*LowChan - low A/D channel of scan
*HighChan - high A/D channel of scan
*TotalCount - total number of points collected
*PreTrigCount - number of pre-trigger points collected
*Rate - sampling rate when data was collected
*Range - Range of A/D when data was collected

Revision 2 Windows languages - Not supported.

cbFilePretrig()

Description: Scan a range of channels continuously while waiting for a trigger. Once the trigger occurs, return the specified number of samples including the specified number of pre-trigger samples to a disk file. This function waits for a trigger signal to occur on the Trigger Input. Once the trigger occurs, it returns the specified number (TotalCount) of A/D samples including the specified number of pre trigger points. It collects the data at the specified sampling rate (Rate) from the specified range (LowChan-HighChan) of A/D channels from the specified board. If the A/D board has programmable gain then it sets the gain to the specified range. The collected data is returned to a file. cbFilexxx() functions are only for boards with REP-INSW transfer capability.

Summary: int cbFilePretrig (int BoardNum, int LowChan, int HighChan, long *PreTrigCount, long *TotalCount, long *Rate, int Range, char *FileName, unsigned Options)

Arguments:
BoardNum - board number may be 0 to 9.
LowChan - First A/D channel of scan
HighChan - Last A/D channel of scan
PreTrigCount - Number of pre-trigger samples to collect.
TotalCount - Total number of samples to collect
Rate - Sample rate in samples per second (Hz)
Range - A/D Range or 0
FileName - Name of disk file
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

BoardNum - refers to the board number associated with the board when it was installed with the configuration program. The specified board must have an A/D and pretrigger capability.

Channel - The maximum allowable channel depends on which type of A/D board is being used. For boards that have both single ended and differential inputs the maximum allowable channel number also depends on how the board is configured (8 chans for differential, 16 for single ended).

PreTrigCount - Specifies the number of samples before the trigger that will be returned. PreTrigCount must be less than 16000 and PreTrigCount must also be less then TotalCount-512.

If the trigger occurs too early, then fewer than the requested number of pre-trigger samples will be collected. In that case a TOOFEW error will occur. The PretrigCount will be set to indicate how many samples were collected and the post trigger samples will still be collected.

TotalCount - Specifies the total number of samples that will be collected and stored in the file. TotalCount must be greater then or equal to PretrigCount+512. If the trigger occurs too early then fewer than the requested number of samples will be collected. In that case a TOOFEW error will occur. The TotalCount will be set to indicate how many samples were actually collected.

Rate - The maximum sampling rate depends on the A/D board that is being used.

Range - If the selected A/D board does not have a programmable range feature, then this argument should be set to 0. Otherwise the gain can be set to any of the following ranges that are supported by the selected A/D board. Refer to board specific information for the list of ranges supported by each board.

BIP10VOLTS	+/- 10 volts	UNI10VOLTS	0 - 10 volts
BIP5VOLTS	+/- 5 volts	UNI5VOLTS	0 - 5 volts
BIP2PT5VOLTS	+/- 2.5 volts	UNI2PT5VOLTS	0 - 2.5 volts
BIP1PT25VOLTS	+/- 1.25 volts	UNI2VOLTS	0 - 2 volts
BIP1VOLTS	+/- 1 volts	UNI1PT25VOLTS	0 - 1.25 volts
BIPPT625VOLTS	+/- 0.625 volts	UNI1VOLTS	0 - 1 volts
BIPPT5VOLTS	+/- 0.5 volts	UNIPT1VOLTS	0 - 0.1 volts

BIPPT1VOLTS	+/- 0.1 volts	UNIPT01VOLTS	0 - 0.01 volts
BIPPT05VOLTS	+/-0.05 volts	MA4TO20	4 - 20 mA
BIPPT01VOLTS	+/- 001 volts	MA2TO10	2 - 10 mA
BIPPT005VOLTS	+/- 0.005 volts	MA1TO5	1 - 5 mA
		MAPT5TO2PT5	0.5 - 2.5 mA

FileName - The named file must already exist. It should have been previously created with the MAKESTRM.EXE program.

Options:
TIMED - The A/D rate is driven by the on-board pacer clock at the rate specified in the Rate argument. This is the default.

EXTCLOCK - If this option is used then conversions will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each conversion will be triggered on the rising edge of the trigger input signal. When this option is used the Rate argument is ignored. The sampling rate is dependent on the trigger signal.

DTCONNECT - Samples are sent to the DT-Connect port if the board is equipped with one.

NODTCONNECT - Samples are not output to the DT-Connect port, if any. This is the default.

Returns: error code or 0 if no errors
*PreTrigCount - actual number of pre-trigger samples collected
*TotalCount - actual number of samples collected
*Rate = actual sampling rate

OVERRUN Error - This error indicates that the data was not written to the file as fast as the data was sampled. Consequently some data was lost. The value in *Count will be the number of points that were successfully collected.

Revision 2 Windows languages - Not supported.

cbFileRead()

Description: Reads data from a streamer file. cbFilexxx() functions are only for boards with REP-INSW transfer capability.

Summary int cbFileRead (char *FileName, long FirstPoint, long *Total Count, int *DataBuffer)

Arguments: FileName - Name of streamer file
FirstPoint - Index of first point to read
Total Count - Number of points to read from file
DataBuffer - Pointer to data buffer that data will be read into.

Data Format - The data is returned as 12 bit analog samples.

Loading Portions of Files - The file may contain much more data then can fit in DataBuffer. In those cases use Total Count and FirstPoint to read a selected piece of the file into DataBuffer. Call cbFileGetInfo() first to find out how many points are in the file.

Returns: Error code or 0 if no errors
DataBuffer - data read from file
Total Count - number of points actually read. This may be less then the requested number of points if an error occurs.

Revision 2 Windows languages - Not supported.

cbGetErrMsg()

Description: Returns the error message associated with an error code. Each function returns an error code. If the error code is not equal to 0 it indicates that an error occurred. Call this function to convert the returned error code to a descriptive error message.

Summary: int cbGetErrMsg (int ErrCode, char ErrMsg[ERRSTRLEN])

Arguments: ErrCode - error code that was returned by any function in library .
ErrMsg - error message returned here

The ErrMsg variable must be pre-allocated to be at least as large as ERRSTRLEN. This size is guaranteed to be large enough to hold the longest error message.

See also cbErrHandling() for an alternate method of handling errors.

Returns: Error code or 0 if no errors
*ErrMsg - error message string is returned here

cbGetStatus()

Description: Returns status about background operation currently running

Summary: int cbGetStatus (int BoardNum, int *Status, long *CurCount, long *CurIndex)

Arguments: BoardNum - board number may be 0 to 9.
Status - pointer to where status will be returned
CurCount - current count returned to this variable.
CurIndex - current index returned to this variable.

EXPLANATION OF THE VARIABLES:

Status - status indicates whether or not a background process is currently executing.

CurCount - Specifies how many points have been input or output. It can be used to gauge how far along the operation is towards completion.

CurIndex - This is an index into the data buffer that points at the start of the last completed channel scan. This can be used to provide a real- time display for a background operation. DataBuffer[CurIndex] points to the last point that was put in or taken out of the buffer. If no points in the buffer have been accessed yet then CurIndex will equal -1.

If you use the CONVERTDATA option with either the CONTINUOUS option or with pre-triggering functions then CBIndex will return the index of the last A/D sample, rather than the start of the last completed channel scan.

For many background operations CurCount=CurIndex. For Pre-Trigger inputs though they are different. CurCount indicates how many points of the TotalCount have been collected. CurCount will rise to PreTrigCount and then stop until the trigger occurs. CurIndex though will constantly increase and reset as it goes around and around the circular buffer while waiting for the trigger to occur.

Returns: Error code or 0 if no errors
*Status= IDLE - No background operation has been executed RUNNING - Background operation still underway
*CurCount = current number of samples collected
*CurIndex = Current sample index

Revision 2 Windows languages - Not supported.

cbMemRead()

Description: Reads data from a memory board into an array.

Summary: int cbMemRead (int BoardNum, unsigned *DataBuffer[], long FirstPoint, long Count)

Arguments: BoardNum - board number may be 0 to 9.
DataBuffer - Pointer to the data array
FirstPoint - Index of first point to read or FROMHERE.
Count - Number of points (words) to read

EXPLANATION OF THE VARIABLES:

FirstPoint - Use the FirstPoint argument to specify the first point to be read. For example, to read points #200 - #250, set FirstPoint=200 and Count=50.

If you are going to read a large amount of data from the board in small chunks then set FirstPoint to FROMHERE to read each successive chunk. Using FROMHERE speeds up the operation of cbMemRead() when working with large amounts of data.

For example, to read 300,000 points in 100,000 point chunks the calls would look like this

```
cbMemRead (0, DataBuffer, 0, 100000)
cbMemRead (0, DataBuffer, FROMHERE, 1000000)
cbMemRead (0, DataBuffer, FROMHERE, 1000000)
```

DT-CONNECT Conflicts - The cbMemRead() function can not be called while a DT-CONNECT transfer is in progress. For example, if you start collecting A/D data to the memory board in the background (by calling cbAInScan() with the DTCONNECT + BACKGROUND options). You can not call cbMemRead() until the cbAInScan has completed. If you do you will get a DTACTIVE error.

Returns: Error code or 0 if no errors
DataBuffer - data read from memory board

cbMemReadPretrig()

Description: Reads pre-trigger data from a memory board that has been collected with the cbAPretrigger() function. This function can only be used to retrieve data that has been collected with the cbAPretrigger() function. After each cbAPretrigger() call, all data must be unloaded from the memory board with this function. If any more data is sent to the memory board then the pre-trigger data will be lost.

Summary: int cbMemReadPretrig (int BoardNum, unsigned *DataBuffer[], long FirstPoint, long Count)

Arguments: BoardNum - board number may be 0 to 9.
DataBuffer - Pointer to the data array
FirstPoint - Index of first point to read or FROMHERE.
Count - Number of points (words) to read

EXPLANATION OF THE VARIABLES:

FirstPoint - Use the FirstPoint argument to specify the first point to be read. For example, to read points #200 - #250, set FirstPoint=200 and Count=50.

If you are going to read a large amount of data from the board in small chunks then set FirstPoint to FROMHERE to read each successive chunk. Using FROMHERE speeds up the operation of cbMemRead() when working with large amounts of data.

For example, to read 300,000 points in 100,000 chunks the calls would look like this

```
cbMemRead (0, DataBuffer, 0, 100000)
cbMemRead (0, DataBuffer, FROMHERE, 1000000)
cbMemRead (0, DataBuffer, FROMHERE, 1000000)
```

DT Connect Conflicts - The cbMemRead() function can not be called while a DT Connect transfer is in progress. For example, if you start collecting A/D data to the memory board in the background (by calling cbAInScan() with the DTCONNECT + BACKGROUND options). You can not call cbMemRead() until the cbAInScan has completed. If you do you will get a DTACTIVE error.

Returns: Error code or 0 if no errors
DataBuffer - data read from memory board

Revision 2 Windows languages - Not Supported

cbMemReset()

Description: Resets the memory board pointer to the start of the data. The memory boards are sequential devices. They contain a counter which points to the 'current' word in memory. Every time a word is read or written this counter increments to the next word.

Summary: int cbMemReset (int BoardNum)

Arguments: BoardNum - board number may be 0 to 9.

This function is used to reset the counter back to the start of the memory. Between successive calls to cbAInScan() you would call this function so that the second cbAInScan() overwrites the data from the first call. Otherwise the data from the first cbAInScan() will be followed by the data from the second cbAInScan() in the memory on the card.

Likewise, anytime you call cbMemRead() or cbMemWrite it will leave the counter pointing to the next memory location after the data that you read or wrote. Call cbMemReset() to reset back to the start of the memory buffer before the next call to cbAInScan().

Returns: Error code or 0 if no errors

cbMemSetDTMode()

Description: Sets the DT Connect Mode of a Memory Board

Summary: int cbMemSetDTMode (int BoardNum, int Mode)

Arguments: BoardNum - board number may be 0 to 9.
Mode - Direction of memory board DT Transfer

EXPLANATION OF THE VARIABLES:

Mode - Must be set to either DTIN or DTOUT. Set the Mode on the memory board to DTIN if you wish to transfer data from an A/D board to the memory board. Set Mode=DTOUT if you wish to transfer data from a memory board to a D/A board.

This command only controls the direction of data transfer between the memory board and another board that is connected to it via a DT Connect cable.

Returns: Error code or 0 if no errors

cbMemWrite()

Description: Writes data from an array to the memory card

Summary: int cbMemWrite(int BoardNum, unsigned *DataBuffer[], long FirstPoint, long Count)

Arguments: BoardNum - board number may be 0 to 9.
DataBuffer - Pointer to the data array
FirstPoint - Index of first point to write or FROMHERE.
Count - Number of points (words) to write

EXPLANATION OF THE VARIABLES:

FirstPoint - Use the FirstPoint argument to specify where in the board's memory to write the first point. For example, to write to locations #200 - #250, set FirstPoint=200 and Count=50.

If you are going to write a large amount of data to the board in small chunks then set FirstPoint to FROMHERE to write each successive chunk. Using FROMHERE speeds up the operation of cbMemWrite() when working with large amounts of data.

For example, to write 300,000 points in 100,000 point chunks the calls would look like this

```
cbMemWrite (0, DataBuffer, 0, 100000)
cbMemWrite (0, DataBuffer, FROMHERE, 1000000)
cbMemWrite (0, DataBuffer, FROMHERE, 1000000)
```

DT Connect Conflicts - The cbMemWrite() function can not be called while a DT Connect transfer is in progress. For example, if you start collecting A/D data to the memory board in the background (by calling cbAInScan() with the DTCONNECT + BACKGROUND options). You can not call cbMemWrite() until the cbAInScan has completed. If you do you will get a DTACTIVE error.

Returns: Error code or 0 if no errors

cbStopBackground()

Description: Stops any background operation that is in progress for the specified board. This function can be used to stop any function that is running in the background. This includes any function that was started with the BACKGROUND option as well as cbCStoreOnIntSt() which always runs in the background.

Summary: int cbStopBackground (int BoardNum)

Arguments: BoardNum - board number may be 0 to 9.

Returns: Error code or 0 if no errors

cbTIn()

Description: Reads a multiplexer input channel, linearizes it according to selected thermocouple type, and returns the temperature in whole degrees. The CJC channel, the gain, and thermocouple type, are read from the configuration file. They should be set by running the InstaCal™ configuration program.

Summary: int cbTIn (int BoardNum, int Chan, int Scale, float *TempVal, int Options)

Arguments: BoardNum - board number may be 0 to 9.
Chan - Channel to read
Scale - The temperature scale for which to calulate the temperature in degrees.
TempVal - Temperature returned here
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

Chan - Input channel to read. The channel number is calculated using the following formula:
A/DChan = A/D channel that mux is connected to
MuxChan = Mux board input channel number
Chan = (ADChan+1) * 16 + MuxChan

For example, if you had an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0 (remember, DAS08 channels are numbered 0,1,2,3,4,5,6 & 7), AND if you had a thermocouple connected to channel 5 of the EXP16, the value for Chan would be (0+1)*16 + 5 = 21.

Scale - Specifies the temperature scale that the input will be converted to. Choices are CELSIUS, FAHRENHEIT and KELVIN.

TempVal - The temperature in degrees is returned here. Temperature is reported in degrees.

A/D Range - IMPORTANT - If the EXP board is connected to an A/D that does not have programmable gain (DAS08, DAS16, DAS16F) then the A/D board (switches) must be set for +- 5 volt range.

Options:
FILTER - The TIn applies a smoothing function to thermocouple readings very much like the electrical smoothing inherent in all themocouple instruments. This is the default.

FILTEROFF - If you use the NOFILTER option then the thermocouple will readings will not be smoothed and you will see a scattering of readings around an mean.

Returns: Error code or 0 if no errors *TempVal - Temperature returned here

Note on CJC Channel: The CJC channel is set in the InstaCal install program. If you have multiple EXP boards, Universal Library will apply the CJC reading to the linearization formula in this manner. First, if you have chosen a CJC channel for the EXP board that the channel you are reading is on, it will use the CJC temp reading from that channel. Second, if you have left the CJC channel for the EXP board that the channel you are reading is on to NOT SET, the library will use the CJC reading from the next lower EXP board with a CJC channel selected.

For example: If you have 4 CIO-EXP16 boards connected to a CIO-DAS08 on channel 0,1,2 and 3, and you have chosen CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7 , AND, you have left CIO-EXP16s 2, 3 and 4 CJC channels to NOT SET, then those CIO-EXP boards will all use the CJC reading from CIO-EXP16 #1, connected to channel 7 for linearization. As you can see, it is important to keep the CIO-EXP boards in the same case and out of any breezes to ensure a clean CJC reading.

cbTInScan()

Description: Reads a range of channels from a multiplexer board, linearizes them according to thermocouple type, and returns the temperatures to an array in whole degrees. The CJC channel, the gain, and thermocouple type are read from the configuration file. Use the InstaCal™ configuration program to change any of these options.

Summary: int cbTInScan (int BoardNum, int LowChan, int HighChan, int Scale, float DataBuffer[], int Options)

Arguments:
BoardNum - board number may be 0 to 9.
LowChan - Low mux channel of scan
HighChan - High mux channel of scan
Scale - CELSIUS, FAHRENHEIT or KELVIN
DataBuffer - Temperature returned here
Options - Bit fields that control various options

EXPLANATION OF THE VARIABLES:

LowChan/HighChan - Specify the range of multiplexer channels that will be scanned. These channel numbers are calculated using the following formula:

A/DChan = A/D channel that mux is connected to
MuxChan = Mux board input channel number
Chan = (ADChan+1) * 16 + MuxChan

For example, if you had an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0 (remember, DAS08 channels are numbered 0,1,2,3,4,5,6 & 7), AND if you had a thermocouple connected to channel 5 of the EXP16, the value for Chan would be (0+1)*16 + 5 = 21.

Scale - Specifies the temperature scale that the input will be converted to. Choices are CELSIUS, FAHRENHEIT and KELVIN.

DataBuffer[] - The temperature is returned in degrees. Each element in the array corresponds to a channel in the scan. DataBuffer must be at least large enough to hold HighChan - LowChan + 1 temperature values.

Chan - Input channel to read. The channel number is calculated using the following formula:

A/DChan = A/D channel that mux is connected to
MuxChan = Mux board input channel number
Chan = (ADChan+1) * 16 + MuxChan

Options:
FILTER - The TIn applies a smoothing function to thermocouple readings very much like the electrical smoothing inherent in all themocouple instruments. This is the default.

FILTEROFF - If you use the NOFILTER option then the thermocouple will readings will not be smoothed and you will see a scattering of readings around an mean.

Range - IMPORTANT - If the EXP board is connected to an A/D that does not have programmable gain (DAS08, DAS16, DAS16F) then the A/D board (switches) must be set for +- 5 volt range.

Returns: Error code or 0 if no errors
DataBuffer[] - Temperature values in degrees are returned here for each channel in scan

Note on CJC Channel: The CJC channel is set in the InstaCal install program. If you have multiple EXP boards, Universal Library will apply the CJC reading to the linearization formula in this manner. First, if you have chosen a CJC channel for the EXP board that the channel you are reading is on, it will use the CJC temp reading from that channel. Second, if you have left the CJC channel for the EXP board that the channel you are reading is on to NOT SET, the library will use the CJC reading from the next lower EXP board with a CJC channel selected.

For example: If you have 4 CIO-EXP16 boards connected to a CIO-DAS08 on channel 0,1,2 and 3, and you have chosen CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7 , AND, you have left CIO-EXP16s 2, 3 and 4 CJC channels to NOT SET, then those CIO-EXP boards will all use the CJC reading from CIO-EXP16 #1, connected to channel 7 for linearization. As you can see, it is important to keep the CIO-EXP boards in the same case and out of any breezes to ensure a clean CJC reading.

Using the File Functions

One of the most powerful features of the ComputerBoards Universal Library is the ability to collect very large amounts of data to a "streamer" file. The amount of data that can be collected is limited only by the size of your hard disk.

Once all of the data has been streamed to a file your program can read it back into arrays and process it in chunks

.

Overview

The library contains four functions that are used with "streamer" files. cbFileAInScan() and cbFilePretrig() read the A/D and store the data in a "streamer" file. cbFILEGetInfo() returns information about the streamer file. cbFileRead() reads data from a "streamer" file to an array.

In addition to these library functions the library comes with three utility programs - MAKESTRM.EXE, FRAGTEST.EXE and RDSTREAM.EXE

MAKESTRM creates a "streamer" file. This program should be run to allocate a file large enough to hold all of the data that will be later collected with cbFileAInScan() or cbFilePretrig().

FRAGTEST checks an existing disk file to see if it is fragmented. In order to run at the faster sampling rates the "streamer" file must not be fragmented. Refer to "Speeding up Disk Files" below for more information.

RDSTREAM reads a "streamer" file and prints its contents on the screen.

Hard Disk vs Ram Disk Files

The simplest type of files to use is a standard DOS file on a hard disk. The advantage of hard disk files is that they can be very large. The file size is only limited by the amount of free space on the disk. Hard disk files have the disadvantage of being slower than RAM disks. RAM disk (or virtual disk) files are faster but they are limited in size by the amount of available memory in your computer.

Maximum Sampling Speed

The maximum sustainable sampling rate that can be specified with the cbFile functions is very hard to predict. It depends on the speed of the CPU and the speed of the disk.

In addition to the variation in sampling speed from machine to machine there can also be variations on the same machine between consecutive operations of the same program. When reading an A/D to memory (non-streaming modes) there is a hard and fast maximum sampling speed that can not be exceeded. When using the streaming modes the maximum rate is much fuzzier and must be arrived at by trial and error.

A rough guideline of attainable speeds are that on a 33 MHz 80386 machine with a fast hard disk it should be possible to collect a megabyte of data at 200khz sampling rate to a disk file. It should also be possible to collect a megabyte of data to a RAM disk at 330 kHz.

In general the maximum sustainable speed for cbFilePretrig() will be somewhat less than for cbFileAInScan().

Another characteristic of these "streaming" modes is that the more data you collect the lower the maximum speed will be. On any machine with any speed disk you can collect 32000 samples to a disk file at the maximum A/D speed of 330 kHz. If you are pushing the upper limits of speed you will find that you can collect 100k samples at a faster rate than you can collect 500k samples etc.

How To Determine Maximum Sampling Speed

The only way to determine the maximum safe speed is to try it repeatedly. Remember if it works the first time it will not necessarily work the next time. Therefore the only way to be sure that you can reliably run at a particular speed is to try it numerous times or else increase the speed to the point where it begins to fail everytime so that you get some sense of whether or not you are pushing the speed limit on your computer.

To test it, write a program that calls cbFileAInScan() or cbFilePretrig() (depending on whether you need pre-trigger data). Check the error code that is returned. If you get an OVERRUN error (error code of 29) it means that the sampling rate is too high. Whenever you get OVERRUN error, some data was collected but not all of it. It is often useful to check how much data was collected to find out whether it was almost fast enough or not even close.

Speeding Up Disk Files (De-Fragmenting):

Because of the way that disks work, the time that it takes to write to them can vary tremendously. A large disk file is made up of many small pieces that are written individually to the disk. If the file is contiguous (each piece is side by side) the speed is very fast. If the file is fragmented (pieces are in different places on the disk) the speed is much slower. If you create a large disk file the odds are overwhelming that it will be fragmented to some degree and the maximum sampling speed will be much lower than it would be for an unfragmented file.

To get around this problem you should use a Disk Optimizer or De-Fragmenter program immediately before creating the streamer file with MAKESTRM. Once you create the streamer file it will remain unfragmented so long as you don't erase it and recreate it.

Probably the most widely known Disk Optimizer program comes as part of the Norton Utilities, it is called Speed Disk or SD. To run it type: SD /Q

This will execute the "Quick" optimize, which for these purposes works as well as the Full Optimization.

After de-fragmenting the disk create a streamer file that is large enough to hold as much data as you plan to collect with cbFileAInScan() or cbFilePretrig(). To create the disk file run the standalone MAKESTRM.EXE program. This will create a streamer file of the required size.

Once the file is created run FRAGTEST.EXE to see whether or not the file is fragmented. It is possible that the file may be fragmented even though you just de-fragmented the disk. The reason for this is that the disk may contain some bad sectors which could not be moved when the disk was optimized. When you create the new file if it hits one of these bad sectors it has to skip over it - hence fragmention.

If FRAGTEST reports that the file is fragmented create a second file and test that with FRAGTEST. Repeat this until FRAGTEST reports that the file is OK. Once you have an unfragmented disk file you can try using it with cbFileAInScan() or cbFilePretrig() to collect data. If the maximum sampling speed is still too slow you should probably switch to a RAM disk.

What is a Ram Disk?

A RAM disk is not really a disk. It is a device driver that sets aside some of the computer's memory and makes it appear to DOS as a disk drive. When you install a RAM disk on your computer it appears exactly as if you have another VERY fast hard disk drive. For example if you have one hard disk - drive C: then when you install the RAM disk it will appear as if you have another hard disk - drive D:

Once the RAM disk is installed all DOS commands work exactly the same on the RAM disk as on the hard disk. For example you can COPY, DEL, MKDIR, CD just as you would on a hard disk.

Installing a RAM Disk:

The RAM disk driver comes with DOS. Refer to your DOS manual for more information. In most versions of DOS it is called either RAMDRIVE.SYS or VDISK.SYS. To install it you must add one line to your \CONFIG.SYS file. Find which directory the DOS files are installed in on your machine. CD to that directory and look for a file called RAMDRIVE.SYS or VDISK.SYS. If it is not there look at the other .SYS files in the directory and refer to your DOS manual to find out if any of them are a RAM Disk driver. Once you have located the file add an entry to the \CONFIG.SYS file.

If the RAMDRIVE.SYS file was in a directory called DOS then you would add the following line to the \CONFIG.SYS file.

```
device=c:\dos\ramdrive.sys
```

The default size for the RAM disk is usually 64K. You will almost certainly want to make it larger than that. The larger you make it the more data you can collect but the less memory will be available for other programs.

To set up a 4 megabyte RAM disk you would add the following line to your CONFIG.SYS file: device=c:\dos\ramdrive.sys 4000

If your computer is an 80286 or an 80386 then you should install the RAM disk in extended memory (above 1M) by specifying the /e option: device=c:\dos\ramdrive.sys 4000 /e

Once you add the new line to the \CONFIG.SYS file, reboot the computer (Press CTRL-ALT-DEL) to install the RAM disk. When the machine reboots it should print a message on the screen describing the RAM disk.

Using The Ram Disk

To use the RAM disk you just specify the drive letter in the FileName argument of cbFileAInScan() or cbFilePretrig(). For example if the RAM disk is drive D: on you system then you could set the name of the "streamer" file in your program to "d:TEST.DAT"

When you execute cbFileAInScan() or cbFilePreTrig() it will fill up the file on your RAM drive. This file will be lost as soon as the power is switched off so if you wish to keep the data you must copy it to the hard disk before turning the computer off.

Board Specific Information

This chapter describes the uses of the Universal Library with each of the data acquisition boards in Computer Boards product line. The following chart shows the matrix of which library functions can be used with which boards.

CIO-DAS16 Family Table of Examples

NAME	Functions Used - DAS16 FAMILY	16 & 16/F	16/Jr & 1400	1600	16/330	16/M1
ULAI01	cbAIn()	X	X	X	X	X
ULAI02	cbAInScan(), Foreground Mode	X	X	X	X	X
ULAI03	cbAInScan(), Background Mode	X	X	X	X	X
ULAI04	cbAInScan() with cbAConvertData()	X	X	X	X	X
ULAI05	cbAInScan() with manual conversion	X	X	X	X	X
ULAI06	cbAInScan(), continuous background mode	X	X	X	X	X
ULAI07	cbATrig()	X	X	X	X	X
ULAI08	cbAPretrig()				X	X
ULAI09	cbAPretrig() with cbAConvertPretrigData()				X	X
ULAI10	cbALoadQueue()				X	
ULAI11	cbAIn() with raw data to volts conversion	X	X	X	X	X
ULAI12	cbAInScan() with external clock	X	X	X	X	X
ULAI13	cbAInScan() with user specified sampling	X	X	X	X	X
ULCT01	cbCConfig(), cbCLoad & cbCIn 8254 counters	X	X	X	X	X
ULAO01	cbAOut()	X		X		
ULAO02	cbAOutScan()	X		X		
ULDI01	cbDIn() with cbDConfigPort()	X	X	X	X	X
ULDI02	cbDBitIn() with cbDConfigPort()	X	X	X	X	X
ULDO01	cbDOut() with cbDConfigPort()	X	X	X	X	X
ULDO02	cbDBitOut() with cbDConfigPort	X	X	X	X	X
ULGT01	cbGetErrMsg()	X	X	X	X	X
ULGT02	cbGetStatus() with cbStopBackground()	X	X	X	X	X
ULFI01	cbFileAInScan() with MAKESTREAM.EXE				X	X
ULFI02	cbFileRead() with cbFileGetInfo()				X	X
ULMM01	cbAPretrig(), cbMemReadPretrig()		X	X	X	X
ULMM02	cbMemRead(), cbMemWrite() - MegaFifo Example					
ULMM03	cbAInScan(), cbMemRead()		X	X	X	X
ULTI01	cbTIn() TC example works with any board					
ULTI02	cbTInScan() that supports EXP boards.					

CIO-DAS08 Family Table of Examples

NAME	Functions Used - DAS08 FAMILY	08	08/ PG	08/ AO	EXP 16	EXP 32	PCM 08
ULAI01	cbAIn()	X	X	X	X	X	X
ULAI02	cbAInScan(), Foreground Mode	X	X	X			X
ULAI03	cbAInScan(), Background Mode	X	X	X			X
ULAI04	cbAInScan() with cbAConvert	X	X	X			X
ULAI05	cbAInScan() with manual conversion	X	X	X			X
ULAI06	cbAInScan(), continuous background mode	X	X	X			X
ULAI07	cbATrig()	X	X	X	X	X	X
ULAI11	cbAIn() with raw data to volts conversion	X	X	X	X	X	XX
ULAI12	cbAInScan() with external clock	X	X	X			X
ULAI13	cbAInScan() with user specified sampling	X	X	X			X
ULCT01	cbCConfig(), cbCLoad & cbCIn 8254 counters	X	X	X			X
ULAO01	cbAOut()			X			
ULAO02	cbAOutScan()			X			
ULDI01	cbDIn() with cbDConfigPort()	X		X			X
ULDI02	cbDBitIn() with cbDConfigPort()	X		X			X
ULDO01	cbDOut() with cbDConfigPort()	X		X			X
ULDO02	cbDBitOut() with cbDConfigPort	X		X			X
ULGT01	cbGetErrMsg()	X	X	X			X
ULGT02	cbGetStatus() with cbStopBackground()	X	X	X			X

Analog Output Board Table of Examples

NAME	Functions Used - DDA & DAC FAMILY	DDA 06	DAC 16	DAC 08	DAC 02	DDA 4/F
ULAO01	cbAOut()	X	X	X	X	X
ULAO02	cbAOutScan()	X	X	X	X	X
ULDI01	cbDIn() with cbDConfigPort()	X				
ULDI02	cbDBitIn() with cbDConfigPort()	X				
ULDO01	cbDOut() with cbDConfigPort()	X				
ULDO02	cbDBitOut() with cbDConfigPort	X				
ULGT01	cbGetErrMsg()	X	X	X	X	X

Digital & Counter Table of Examples

NAME	Functions Used - DIO & CTR FAMILY	DIO	CTR	PDMA16	INT32	DAS16	DAS08	DDA06
ULCT01	cbC8254Config(), cbCLoad() & cbCIn for 8254 chips			X		X	X	X
ULCT02	cbC9513Init, cbCLoad & cbCIn for 9513 chips		X					
ULCT03	cbC9513Config() & cbCStoreOnInt() for 9513 chips		X					
ULCT04	cbCFreqIn()		X					
ULCT05	cbC8536Init(), cbCLoad() & cbCIn() for 8536 chips				X			
ULDI01	cbDIn() with cbDConfigPort()	X		X	X	X	X	X
ULDI02	cbDBitIn() with cbDConfigPort()	X		X	X	X	X	X
ULDI03	cbDInScan()			X				
ULDI04	cbDIn() with AUXPORT		X			X	X	
ULDO01	cbDOut() with cbDConfigPort()	X		X	X	X	X	X
ULDO02	cbDBitOut() with cbDConfigPort	X		X	X	X	X	X
ULDO03	cbDOutScan()			X				
ULDO04	cbDOut() with AUXPORT		X			X	X	
ULGT01	cbGetErrMsg()	X	X	X	X	X	X	X

CIO-CTR Series Boards Limitation

Counter Functions: These boards use a 9513 counter chip so they don't support 8254 functions (cbC8254Config()).

Digital I/O Functions These boards support all of the digital I/O functions except cbDInScan() and cbDOutScan(). The PortNum is AUXPORT and it reads 8 and writes 8 digital bits.

CIO-DAC Series Boards Limitations

These boards only support the analog output functions. When used with cbAOutScan() Rate is ignored, Count should be set to HighChan - LowChan + 1, and the CONTINUOUS and BACKGROUND options are not supported.

CIO-DAS08 Series Boards

Analog Input Functions:

All analog input functions except for cbAPreTrigger() any cbFilexxx() are available.
A/D Ranges DAS08 - The DAS08 does not have programmable gain so the Range argument to analog input functions should be set to 0.

CIO-DAS08-PGH and CIO-DAS08-AOH supports the following A/D ranges:

BIP10VOLTS = +- 10 volts	BIP5VOLTS = +- 5 volts
BIP1VOLTS = +- 1 volts	BIPPT5VOLTS = +- .5 volts
BIPPT1VOLTS = +- .1 volts	BIPPT05VOLTS = +- .05 volts
BIPPT01VOLTS = +- .01 volts	BIPPT005VOLTS = +- .005 volts
UNI10VOLTS = 0 - 10 volts	UNI1VOLTS = 0 - 1 volts
UNIPT1VOLTS = 0 - .1 volts	UNIPT01VOLTS = 0 - .01 volts

CIO-DAS08-PGL and CIO-DAS08-AOL supports the following A/D ranges:

BIP10VOLTS = +- 10 volts
BIP5VOLTS = +- 5 volts
BIP2PT5VOLTS = +- 2.5 volts
BIP1PT25VOLTS = +- 1.25 volts
BIPPT625VOLTS = + 0.625 volts

UNI10VOLTS = 0 - 10 volts
UNI5VOLTS = 0 - 5 volts
UNI2PT5VOLTS = 0 - 2.5 volts
UNI1PT25VOLTS = 0 - 1.25 volts

Maximum Sampling Rate :

The maximum analog sampling rate is dependent on the speed of the computer that the board is installed in. In general it is somewhere in the range of 5-50 kHz. If the speed you request cannot be sustained, an overrun error will occur.

Options:

The DTCONNECT, EXTMEMORY, BLOCKIO and DMAIO options are not available with this board. BURSTMODE generates a warning message.

Wiring Requirements :

Before using the timed analog input functions (cbAInScan(), cbFileAInScan()). The output of counter 1 (pin #6) must be wired to the Interrupt input (pin #24)

Analog Output Functions

cbAOutScan() ignores the Rate argument and Count must be less then HighChan - LowChan + 1. The CONTINUOUS and BACKGROUND operations are not allowed. The CIO-DAS08, CIO-DAS08-PGL and CIO-DAS08-PGH boards do not support the anlocg output functions.

Digital I/O Functions

The CIO-DAS08 and CIO-DAS08/AO support the following digital I/O ports AUXPORT, FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, andFIRSTPORTCH. These boards to do not support the cbDInScan() and cbDOutScan() functions.

Counter I/O Functions

The CIO-DAS08 and CIO-DAS08AO have an 8254 counter. Only those functions that apply to the 8254 can be used.

CIO-DAS16 Series Boards

Analog Input Functions:

Pre-trigger functions - cbAPreTrigger(), cbFilexxx() are available only with the DAS16/330, DAS16/330i, DAS16/M1; boards which support the REPINSW command.

A/D Range:

CIO-DAS16 and CIO-DAS16F do not have programmable gain so the Range argument to analog input functions should be set to 0.

CIO-DAS16/Jr, CIO-DAS16/330, CIO-DAS16/330i, CIO-DAS16/M1 support the following programmable A/D ranges:
BIP10VOLTS = +- 10 volts
BIP5VOLTS = +- 5 volts
BIP2PT5VOLTS = +- 2.5 volts
BIP1PT25VOLTS = +- 1.25 volts
BIPPT625VOLTS = += .625 volts

UNI10VOLTS = 0 - 10 volts
UNI5VOLTS = 0 - 5 volts
UNI2PT5VOLTS = 0 - 2.5 volts
UNI1PT25VOLTS = 0 - 1.25 volts

Maximum Sampling Rate:

The maximum analog sampling rate varies per board according to the following table.:

CIO-DAS16	50 kHz	CIO-DAS16F	100 kHz	CIO-DAS16Jr	120 kHz
CIO-DAS16/330	330 kHz	CIO-DAS16/330i	330 kHz		

CIO-DAS16/M1: When using EXTMEMORY option, or if count <= 512, 1MHz, otherwise about 500KHz to PC memory.

Options:

The DTCONNECT & EXTMEMORY & BLOCKIO options are only available with the CIO-DAS16/330, CIO-DAS16/330i and the CIO-DAS16/M1 board.

The DMAIO option can not be used while using the chan/gain queue on the DAS-330i board.

Analog Output Functions:

cbAOutScan() ignores the Rate argument and Count must be less then HighChan - LowChan + 1. The CONTINUOUS and BACKGROUND operations are not allowed.

Digital I/O Functions:

All of the boards support the AUXPORT. The CIO-DAS1600, CIO-DAS16 and CIO-DAS16F also support FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, and FIRSTPORTCH. These boards do not support the cbDInScan() and cbDOutScan() functions.

Counter I/O Functions:

The CIO-DAS16 boards have an 8254 counter. Only those functions that apply to the 8254 can be used.

CIO-DAS48/PGA Board

Analog Input Functions:

This board does not support any of the pre-triggering functions.
This board does not have an A/D pacer clock so the Rate argument to cbAInScan() is ignored and the Count argument must never be greater than the number of channels in the scan. i.e. cbAinScan() can only be used to take a single A/D sample from a range of channels.

A/D Ranges:

The board may be configured with a jumper for either voltage or current input.
The voltage input ranges are:

BIP10VOLTS = +/- 10 volts	BIP5VOLTS = +/- 5 volts
BIP2PT5VOLTS = +/- 2.5 volts	BIP1PT25VOLTS = +/- 1.25 volts
BIPPT625VOLTS = +/- 1.625 volts	
UNI10VOLTS= 0 - 10 volts	UNI5VOLTs = 0 -5 volts
UNI2PT5VOLTS= 0 - 2.5 volts	UNI1PT25VOLTS = 0 - 1.25 volts

The current input ranges are:

MA4TO20 = 4 - 20 ma	MA2TO10 = 2 - 10 ma
MA1TO5 = 1 - 5 ma	MAPT5TO2PT5 = .5 - 2.5 ma

Options

No options are allowed with the cbAinScan() function.

Analog Output

The CIO-DAS49/PGA board does not support the analog output functions.

Digital I/O Functions

The CIO-DAS48/PGA does not support any of the digital I/O functions.

Counter I/O Functions

The CIO-DAS48/PGA does not support any of the counter I/O functions.

CIO-DAS1400 & CIO-DAS1600 Series Boards

Analog Input Functions:

All analog input functions except for cbAPreTrigger() and cbFilePretrig() can be used with these boards.
A/D Ranges.

IMPORTANT: The CIO-DAS1400 and CIO-DAS1600 A/D ranges are configured with a combination of a switch (Unipoalr / Bi-polar) and a programmable gain code. The state of this switch is set in the configuration file using InstaCal™. Once the UNI/BIP switch setting is selected, only matching ranges may be used in Universal Library programs.

The allowed A/D voltage ranges are:

CIO-DAS1402, CIO-DAS1602, CIO-DAS1602/16 Ranges

BIP10VOLTS= +/- 10 volts	BIP5VOLTS= +/- 5 volts
BIP2PT5VOLTS= +/- 2.5 volts	BIP1PT25VOLTS= +- 1.25 volts
UNI10VOLTS= 0 - 10 volts	UNI5VOLTS= 0 - 5 volts
UNI2PT5VOLTS= 0 - 2.5 volts	UNI1PT25VOLTS = 0 - 1.25 volts

CIO-DAS1401, CIO-DAS1601/12 Ranges

BIP10VOLTS = +/- 10 volts	BIP1VOLTS = +/- 1 volts
BIPPT1VOLTS = +/- .1 volts	BIPPT01VOLTS = +/- .01 volts
UNI10VOLTS = 0 - 10 volts	UNI1VOLTS = 0 - 1 volt
UNIPT1VOLTS = 0 - .1 volt	UNIPT01VOLTS = 0 - .01 volt

Maximum Sampling Rate:

The maximum analog (A/D) sampling rate for CIO-DAS1400 & CIO-DAS1600is: 330 kHZ to DT-Connect, 160 kHZ to memory
CIO-DAS1600/16 - 100 kHz

Options

The BLOCKIO option can not be used with these boards.
The DTCONNECT and EXTMEMORY options cannot be used with the DAS1400 board.
The CONVERTDATA option can not be used with the DAS1600/16 board. This board does not return channel tags so there is no need for this option.

Analog Output:

The CIO-DAS1400 board does not support the analog output functions.
The CIO-DAS1600 series boards do. cbAOutScan ignores the Rate argument and Count must be no more then HighChanLowChan+1, The CONTINUOUS and BACKGROUND operations are not allowed.

Digital I/O Functions:

All of the boards have an AUXPORT, FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, and FIRSTPORTCH.
The AUXPORT has 4 digital inputs and 4 digital outputs. These boards support all of the digital I/O functions except cbDIn-Scan() and cbDOutScan().

Counter I/O Functions

These boards all have an 8254 counter. They do not support any of the 9513 counter functions - cbC9513Init(), cbC9513Config(), cbCStoreOnInt(), cbCFreqIn().

CIO-DIO Series Boards

These boards support the digital I/O commands except for cbDInScan() and cbDOutScan().
All of these boards support the following digital I/O ports - FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, and FIRSTPORTCH.

The DIO48 board also supports: SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH
The DIO-96 board also supports: SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH, THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH, FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

The DIO-192 board also supports: FIFTHPORTA ..... EIGHTHPORTCH
The maximum bit number for cbDBitIn() and cbDBitOut() is 23 for the DIO-24, 47 for the DIO-48, 95 for the DIO-96, and 191 for the DIO-192.

CIO-DIO24/CTR3

Digital Functions: This board supports all the digital functions except cbDInScan() and cbDOutScan(). Ports supported are FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL and FIRSTPORTCH.

Counter Functions: This board has an 82C54 counter chip and is supported by the config and init functions for the 82C54.

CIO-DDA06

cbAOutScan() ignores the Rate argument and Count must be less than HighChan - LowChan + 1. The CONTINUOUS and BACKGROUND operations are not allowed.

Digital I/O Functions: This board supports all the digital functions except cbDInScan() and cbDOutScan(). The board supports the following digital ports: FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH.

CIO-DISO48

Digital Functions: This board supports all the digital functions except cbDInScan() and cbDOutScan() Ports supported are FIRSTPORTA, SECONDPORTA, THIRDPORTA, FOURTHPORTA, FIFTHPORTA & SIXTHPORTA.

CIO-EXP Series Boards

EXP boards are only used in combination with an A/D board.

CIO-INT32

Counter Functions: This board has two 8536 chips, which have both counter and digital I/O and interrupt vectoring capabilities. Use the counter functions for configuration and initialization that specify the 8536.

Digital I/O Functions: This board supports all the digital functions except cbDInScan() and cbDOutScan(). The board supports the following digital ports: FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, SECONDPORTA, SECONDPORTB and SECONDPORTCL.

CIO-PDISO8

This boards support all of the digital I/O commands except for cbDInScan() and cbDOutScan(). The digital I/O port is addressed as AUXPORT.

CIO-PDMA16

This boards support all of the digital I/O commands except for cbDBitIn() and cbDBitOut().
This board supports the AUXPORT, FIRSTPORTA, and FIRSTPORTB. The cbDInScan() and cbDOutScan() functions only work with FIRSTPORTA and FIRSTPORTB ports.

CIO-RELAY16

Digital Functions: This board supports all digital output functions except cbDOutScan(). Ports are addressed as FIRSTPORTA and SECONDPORTA

PPIO-DIO24H

Digital Functions: This board supports all digital functions except cbDInScan() and cbDOutScan(). Ports are addressed as FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL and FIRSTPORTCH.

PPIO-AI8

Analog Input: This board does not support timed A/D input. cbAInScan() Rate is ignored, BACKGROUND and CONTINUOUS options are not supported.

Digital I/O - AUXPORT is used for access to the digital I/O bits.

The PPIO-AI8 does not have programmable gain so the Range argument to analog input functions should be set to 0.

PPIO-CTR06 Boards

Counter Functions: This board uses an 82C54 so it supports the 8254 functions - cbC8254Config().
Digital I/O Functions This board has a single digital port - AUXPORT. Depending on a jumper setting on the board it either has 8 digital inputs and 4 digital outputs OR 4 digital inputs and 8 digital outputs. This board does not support the cbDInScan() or cbDOutScan() functions.

cSBX-DDA04 Board

Digital I/O - AUXPORT is 8 bits of input, 14 bits of output.
Analog Output cbAOutScan() maximum allowable Rate is 300,000.
Digital Output cbAOutScan() maximum allowable rate is 500,000 - READ NOTE

Note: The cSBX-DDA4 boards allows interleaving of analog and digital output data. In order to support interleaving, a control bit, the MSb of each 16 bit word of analog or digital data indicates the data type. MSb = 0 for analog data and MSb = 1 for digital data. The data is passed to the board and there directed to the correct output type by hardware on the board which detects and acts on the MSb control bit.

PCM-DAS08 Board

Analog Input Funtions: Supports all analog input except cbAPretrig and cbFilePretrig
A/D Range argument is ignored because board range is fixed at +/-5V
Sample Rate. Please see the PCM-DAS08 manual for the table of sample rate vs. channels sampled.
Options for cbAinScan: Does not support DMAIO, BLOCKIO, DTCONNECT, EXTMEMORY
Analog Output Functions: None on the board.
DigitalI/O Functions: Supports digital input and output as AUXPORT. Max cbDBit number is 3

Appendix A. Error Codes

Each entry in the list has four parts. The error code number, it's symbolic name, it's error message, and a explanation.

\# **SYMBOLIC_NAME** *Error message*
Description

0 **NOERRORS** *No error has occurred.*
The function executed successfully

1 **BADBOARD** *Invalid board number.*
The board number that was specified does not match any of the boards that are listed in the configuration file. Run the configuration program to check which board numbers are configured.

2 **DEADDIGITALDEV** *Digital device is not responding - is base address correct?*
The digital device on the specified board is not responding. Either the board was installed incorrectly or the board is defective. Run the configuration program and make sure that the correct boards was installed.

3 **DEADCOUNTERDEV** *Counter device is not responding - is base address correct?*
The counter device on the specified board is not responding. Either the board was installed incorrectly or the board is defective. Run the configuration program and make sure that the correct board was installed.

4 **DEADDADEV** *D/A is not responding - is base address correct?*
The D/A device on the specified board is not responding. Either the board was installed incorrectly or the board is defective. Run the configuration program and make sure that the correct board was installed.

5 **DEADADDEV** *A/D is not responding - is base address correct?*
The A/D device on the specified board is not responding. Either the board was installed incorrectly or the board is defective. Run the configuration program and make sure that the correct boards was installed.

6 **NOTDIGITALCONF** *Selected board does not have digital I/O.*
A digital I/O function was called with a board number that referred to a board that does not support digital I/O. Run the configuration program to see which type of board that board number refers to.

7 **NOTCOUNTERCONF** *Selected board does not have a counter.*
A counter function was called with a board number that referred to a board that does not have a counter. Run the configuration program to see which type of board that board number refers to.

8 **NOTDACONF** *Selected board does not have a D/A.*
An analog output function was called with a board number that referred to a board that does not have an analog output (D/A). Run the configuration program to see which type of board that board number refers to.

9 **NOTADCONF** *Selected board does not have an A/D.*
An analog input function was called with a board number that referred to a board that does not have an analog input (A/D). Run the configuration program to see which type of board that board number refers to.

10 **NOTMUXCONF** *Selected board does not have thermocouple input.*
A thermocouple input function was called with a board number that is not connected to an EXP board. Run the configuration program to view/change the board configuration.

11 **BADPORTNUM** *Invalid digital port number.*
The Port number that was specified for a digital I/O function does not exist on the board that was specified.

12 **BADCOUNTERDEVNUM** *Invalid counter device.*
The Counter Number that was specified for a counter function does not exist on the board that was specified.

13 **BADDADEVNUM** *Invalid D/A device.*
The D/A channel that was specified for an analog output function does not exist on the board that was specified.

14 **BADSAMPLEMODE** *Invalid sample mode.*

A sample mode (INTIO, DMAIO or BLOCKIO) was specified in the Options argument that is not supported on this board. Try running the function without setting any of the Sample Mode options.

15 BADINT *Board configured for invalid interrupt level.*
The interrupt level for this board was set in the configuration program to either 10 or 11. The switch on the board is set for "compatible mode". These interrupts are not valid in compatible mode. Either change the switch setting on the board, or change the interrupt level with the configuration program to something other than 10 or 11.

16 BADADCHAN *Invalid A/D channel number.*
An invalid channel argument was passed to an analog input function. The range of valid channel numbers depends on which A/D board you are using - refer to the board manual. For some boards it also depends on how the board is configured (with a switch). For those boards run the configuration program and check how many channels the board is configured for.

17 BADCOUNT *Invalid count.*
An invalid Count argument was specified to a function.

18 BADCNTRCONFIG *Invalid counter configuration specified.*
An invalid Config argument was passed to cbC8254Config. The only legal values are HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE and HARDWARESTROBE.

19 BADDAVAL *Invalid D/A value.*
An invalid D/A value was passed as an argument to an analog output function. The only legal values are 0 - 4095 for 12 bit boards or 0-65,535 for 16 bit boards.

20 BADDACHAN *Invalid D/A channel number.*
An inavlid D/A channel was passed as an argument to an analog output function. The legal range of values depends on which D/A board you are using. Refer to the board manual to find how many D/A channels it has.

22 ALREADYACTIVE *Background operation already in progress.*
An attempt was made to start a second background process before the first one had completed. Background processes are started whenever the BACKGROUND option is used or by the cbCStoreOnInt function. To stop a background operation call cbStop-Background(). To wait for a background process to complete call cbGetStatus() and wait for status=IDLE.

24 BADRATE *Invalid sampling rate.*
Invalid sampling rate argument was specified. The rate was either a negative number or it was higher than the selected board supports. Refer to board specific information for maximum rates for each board.

25 COMPATMODE *Board switches set for Compatible mode.*
An operation was attempted that is not possible when the board's switch is set for 'compatible' operation. Most likely causes are using the BLOCKIO option or the pre-triggering functions. Either turn off the 'compatible' mode switch on the board or don't use the BLOCKIO option or the pre-triggering functions.

26 TRIGSTATE *Incorrect initial trigger state - D0 must start at TTL low.*
The pre-triggering command requires that the trigger be "off" when the function is first called. It then waits for the trigger signal. Make sure that the Trigger Input line (D0) is held at TTL low before calling the pre-trigger function.

27 ADSTATUSHUNG *A/D is not responding.*
The A/D board is not responding as it should. Usually indicates some kind of hardware problem - either defective hardware or more than one board at the same base address.

28 TOOFEW *Trigger occurred before requested number of samples were collected.*
A pre-trigger function was called and the trigger signal occurred before the requested number of samples could be collected. This is only a warning message. The function continued anyway. The data that was returned to the array will contain fewer than the expected number of points. The function will return the actual number of pre-trigger points and the total number of points. You can use these two values to find your way around the data in the array.

29 OVERRUN *Data overrun - data was lost.*

Data was lost during an analog input because the computer could not keep up with the A/D sampling rate. This typically can only happen with the file input functions. Possible solutions include lowering the sampling rate, defragmenting the "streamer" file, switching to a RAM disk, or lowering the count.

30 BADRANGE *Invalid voltage or counter range..*

Invalid Range argument was specified to an analog input function. There are two possible cases. Either the board does not support software programmable gains, in which case the gain arguments should be set to 0, or else it does support programmable gain but not the one you specified. Refer to board specific information for a list of allowable A/D ranges for each board.

31 NOPROGGAIN *This A/D board does not have programmable gain.*

Invalid Range argument was passed to an analog input function. The selected board does not support programmable gains so the only valid Range argument is 0.

32 BADFILENAME *Specified file name is not valid'.*

The FileName argument that was passed to a file function is not valid. It is either an empty string or a NULL pointer.

33 DISKISFULL *Disk is full, could not complete operation.*

A file operation failed before completing because the disk that it was writing to is full. Try erasing some files from the disk. If this error occurred during either cbFileAInScan() or cbFilePretrig() it indicates another problem. The disk space for these commands should have been previously allocated with MAKESTRM.EXE program. If this error is generated when data is being collected it indicates that you did not allocate a large enough file with MAKESTRM.EXE.

34 COMPATWARN *Board switch set to compatible mode - sampling speed may be limited.*

The board's switch is set for 'Compatible mode". When in "compatible mode" BLOCKIO transfers are not possible. BLOCKIO sampling was specified but it has automatically been changed to DMAIO transfers. The maximum smapling rate will be limited to the maximum rate for DMA transfers. Change the "compatible mode" switch on the board if you want to use BLOCKIO transfers.

35 BADPOINTER *Pointer is not valid.*

An invalid (NULL) pointer was passed as an argument to a function.

37 RATEWARNING *Sample rate may be too fast for SINGLEIO mode.*

The specified sampling rate MAY be too high. The maximum allowable sampling rate depends very much on the computer that the program is running on. This warning is generated based on the slowest CPU speed. Your computer may be able to sustain faster rates, but, you should expect the computer to lock up (fail to respond to keyboard input) if you do exceed the sampling rate your computer can sustain.

38 CONVERTDMA *CONVERTDATA cannot be used with DMA I/O and BACKGROUND.*

The CONVERTDATA and BACKGROUND options can not be used together when the board is transferring data via DMA. Possible solutions include: Use cbAConvertData() to convert the data after it is collected. Don't use BACKGROUND option. Use BLOCKIO Option if your A/D board supports it. Use INTIO Option if your computer is fast enough to support the selected sampling rate.

39 DTCONNECTERR *Board does not support DTCONNECT option.*

The DTCONNECT Option was passed to an analog input function. The selected board does not support that option.

40 FORECONTINUOUS *CONTINUOUS can only be run with BACKGROUND.*

The CONTINUOUS Option was passed to a function without also setting the BACKGROUND Option. This is not allowed. Any time you set the CONTINUOUS Option you must also set the BACKGROUND option.

41 BADBOARDTYPE This function can not be used with this board'.

An attempt was made to call a function for a board that does not support that function.

42 WRONGDIGCONFIG *Digital port not configured correctly for requested operation.*

Some of the digital ports (FIRSTPORTA - EIGHTHPORTCH) must be configured as inputs OR outputs but not both. An attempt was made to use a digital input function on a port that was configured as an output or vice versa. Use cbDConfigPort to switch a port's direction. If you do not call cbDConfigPort() in your program then all of the configurable ports will be in in an unknown state (input or output).

43 NOTCONFIGURABLE *This digital port is not configurable (its an In/Out port).*
cbDConfigPort() was called for a port that is not configurable. Check the Port Number argument passed to cbDConfigPort() and make sure that it is in the range FIRSTPORTA - EIGHTHPORTCH. If not then there is no need to call this function.

44 BADPORTCONFIG *Invalid digital port configuration.*
The Direction argument passed to cbDConfigPort is invalid. It must be set to either DIGITALIN or DIGITALOUT.

45 BADFIRSTPOINT *First point number is not valid'.*
The First Point argument to cbReadFile() is invalid. It is either a negative number or it is larger then the number of points in the file.

46 ENDOFFILE *Attempted to read past the end of the file.*
cbFileRead() attempted to read beyond the end of the file. Check the file length with cbFileGetInfo() and make sure that the First Point and Count arguments to cbFileRead() are correct for that file length.

47 NOT8254CTR *This board does not have 8254 counter.*
The cbC8254Config() function was called for a board that has a counter but not an 8254 counter. This function can only be used with an 8254.

48 NOT9513CTR *This board does not have 9513 counter.*
A function was called for a board that has a counter but not a 9513 counter. This function can only be used with an 9513.

49 BADTRIGTYPE *Invalid TrigType.*
cbATrig() was called with an invalid TrigType argument. It must be set to either TRIGABOVE or TRIGBELOW.

50 BADTRIGVALUE *Invalid TrigValue (> 4095).*
cbATrig() was called with an invalid TrigValue argument. It must be in the range 0 - 4095.

52 BADOPTION *Invalid option specified for this function.*
The Option argument contains an option that is not valid for this function.

53 BADPRETRIGCOUNT *Invalid PreTrigCount specified.*
Either cbAPretrig() or cbFilePretrig() was called with an invalid preTrigCount argument. A valid pre-trigger count must be > 0 and less then TotalCount-512. It also must be less then 32000 for cbAPretrig() and less than 16,000 for cbFilePretrig().

55 BADDIVIDER *Invalid Fout Divider value.*
The Fout Divider argument to cbC9513Init() is not valid. It must be in the range 0 - 15.

56 BADSOURCE *Invalid Fout Source value.*
The Source argument to cbC9513Init() is not valid. It must be one of the following values CTRINPUT1, CTRINPUT2, CTRINPUT3, CTRINPUT4, CTRINPUT5, GATE1, GATE2, GATE3, GATE4, GATE5, FREQ1, FERQ2, FREQ3, FREQ4, FREQ5 (i.e. 0 - 15)

57 BADCOMPARE *Invalid compare value.*
One or both of the compare arguments to cbC9513Init() is not valid. They must be set to ENABLED or DISABLED.

58 BADTIMEOFDAY *Invalid Time Of Day value.*
The TimeOfDay argument to cbC9513Init() is not valid. It must be set to either ENABLED or DISABLED.

59 BADGATEINTERVAL *Invalid Gate Interval value.*
The GateInterval argument to cbCFreqIn() is not valid. It must be greater then 0.

60 BADGATECNTRL *Invalid Gate Control value.*
The GateControl argument to cbC9513Config() is not valid. It must be in the range 0 -7.

61 BADCOUNTEREDGE *Invalid Counter Edge value*
The CounterEdge argument to cbC9513Config() is not valid. It must be set to either POSITIVEEDGE or NEGATIVEEDGE.

62 BADSPCLGATE *Invalid Special Gate value.*

The SpecialGate argument to cbC9513Config() is not valid. It must be set to either ENABLED or DISABLED.

63 **BADRELOAD** *Invalid Reload value.*
The Reload argument to cbC9513Config() is not valid. It must be set to either LOADREG or LOADANDHOLDREG

64 **BADRECYCLEFLAG** *Invalid Recycle Mode value.*
The RecycleMode argument to cbC9513Config() is not valid. It must be set to either ENABLED or DISABLED.

65 **BADBCDFLAG** *Invalid BCD Mode value.*
The BCDMode argument to cbC9513Config() is not valid. It must be set to either ENABLED or DISABLED.

66 **BADDIRECTION** *Invalid Count Direction value.*
The CountDirection argument to cbC9513Config() is not valid. It must be set to either COUNTUP or COUNTDOWN.

67 **BADOUTCONTROL** *Invalid Output Control value.*
The OutputControl argument to cbC9513Config() is not valid. It must be set to either ALWAYSLOW, HIGHPULSEONTC, TOGGLEONTC, DISCONNECTED or LOWPULSEONTC.

68 **BADBITNUMBER** *Invalid Bitnum specified.*
The BitNum argument to cbDBitIn() or cbDBitOut() is not valid. The valid range of bit numbers depends on the selected board. If it is a DIO24 compatible board the maximum bit number is 23. If it's a DIO96, the maximum bit number is 95.

69 **NONEENABLED** *None of the counter channels were enabled.*
None of the counter channels were marked as ENABLED in the CntrCntrl array that was passed to cbCStoreOnInt(). At least one of the counter channels must be enabled.

70 **BADCTRCONTROL** *An element of Cntr Control array not set to DISABLED or ENABLED.*
One of the elements of the CntrControl array that was passed to cbCStoreOnInt() was set to something other then ENABLED or DISABLED. The array must have at least ten elements and the first ten elements must be set to either ENABLED or DISABLED.

71 **BADEXPCHAN** *Invalid EXP channel specified.*
An invalid channel was passed to one of the thermocouple input commands. The channel number must be >= 0. The maximum allowable channel number depends on which EXP board is being used. Refer to the board manual to find the number of channels.

72 **WRONGADRANGE** *Board set to wrong A/D range for reading thermocouples.*
A thermocouple input function was called to read an EXP board input. The EXP board is connected to an A/D board that is set to the wrong range. The A/D must be set to the -5 to +5 volt range to work with the EXP board.

73 **OUTOFRANGE** *Temperature input is out of range.*
A thermocouple input function returned an inavalid temperature. This usually indicates an open connection in the thermocouple or its connection to the mux board.

74 **BADTEMPSCALE** *Invalid temperature scale specified.*
The Scale argument to a thermocouple input function is not valid. It must be set to either CELSIUS, FAHRENHEIT or KELVIN.

75 **BADERRCODE** *Invalid error code.*
The ErrCode argument to cbGetErrMsg() is invalid. It is not a known error code.

76 **NOQUEUE** *Specified board does not have channel/gain queue.*
The function that was called requires that the board has a channel/gain queue. The specified board does not have a queue.

77 **CONTINUOSCOUNT** *Count must be > 1023 to use recycle mode*
The count argument is not valid.

78 **UNDERRUN** *D/A FIFO went empty during output.*
The specified D/A output rate could not be sustained. This error should not normally occur.

79 **BADMEMMODE** *Invalid memory mode specified.*

The memory mode that was selected with cbMemSetDTMode is not one of the valid modes.

80 FREQOVERRUN *Measured frequency too high for selected gating interval.*
The gating interval used with cbFreqIn is too long to measure the frequency of the signal connected to the counter. The counter is overflowing. Decrease the gating interval to eliminate the error.

81 NOCJCCHAN *A CJC Channel must be configured to make temperature measurements.*
When the board was installed (with the InstaCal installation program) no CJC channel was selected. To use the temperature measurement functions you must first select a CJC channel on the A/D board and then rerun the installation program.

82 BADCHIPNUM *Invalid Chipnum specified.*
An invalid ChipNum argument was used with the cbC9513Init() function. If the board is CTR05 then ChipNum should be set to 0. If it is a CTR10, ChipNum should be either 0 or 1.

83 DIGNOTENABLED *The digital I/O on this board is not enabled.*
When the board was installed (with the InstaCal installation program), the expansion digital I/O was set to DISABLED. To use these digital I/O lines you must enable the digital I/O on the board (with a jumper) and then re-run the installation program and set the digital I/O to ENABLED.

84 CONVERT16BITS *CONVERT option can not be used with 16 bit A/D converters.*
When using a 16 bit A/D (DAS1600/16), if you try to use the CONVERT option with cbAInScan() or call cbAConvertData() you will get this error.

85 NOMEMBOARD *The EXTMEMORY option requires that a MEGA-FIFO be attached..*
Attempt to use a cbMem function without a MEGA-FIFO board installed. Install MEGA-FIFO through InstaCal.

86 DTACTIVE *No memory read/write allowed while DT transfer in progress.*
A read or write to a memory board was attempted while data was being transferred via DT Connect.

87 NOTMEMCONF *Specified board is not a memory board.*
The specified board is not a memory board. This function only works with memory boards.

88 ODDCHAN *The first channel in scan and number of channels must be even (0, 2, 4, etc).*
Some boards use a channel/gain queue that require the first channel in the queue and the number of channels in the queue always be an even channel. This error can occur even when you are not in the process of loading the queue. Some boards use the queue automatically with cbAInScan(). On those boards the low channel must be an even number.

89 CTRNOINIT *Counter was not configured or initialized.*
You attempted to use cbCLoad() or cbCIn() before inititalizing and configuring the counter.

90 NOT8536CTR *This board does not have an 8536 counter chip.*
Attempt to use 8536 initialization or configuration on board without 8536 chip.

91 FREERUNNING *Board doesn't time A/D sampling. Collecting at fastest possible speed.*
This board does not have an A/D pacer mechanism and you have called the function cbAInScan(). The A/D will be sampled in a tight software loop as fast as the CPU can execute the instructions. The speed of sampling is dependent on the computer and the concurrent tasks.

92 INTERRUPTED *Operation interrupted with Ctrl-C key.*
A background operation was stopped before completion because either the Ctrl-C or Ctrl-Break keys were pressed.

93 NOSELECTORS *No selector could be allocated.*
A Windows selector required by the library could not be allocated. Close any unneeded open Windows applications and try again.

94 NOBURSTMODE *This board does not support burst mode.*
An attempt was made to use the BURSTMODE option on a board which does not support that option.

95 NOTWINDOWSFUNC *This function is not available in Windows library.*

The library function you called is not supported in the the current revision of Universal Library for Windows Languages. It may be supported in the future. Please call.

96 NOTSIMULCONF *Board not configured for SIMULTANEOUS option.*
The configuration file of the D/A board in InstaCal must be set for simultaneous update before you use the SIMULTANEOUS option of cbAOutScan() function. Of course, it goes without saying that the jumpers on the D/A board must be set for simultaneous update before it will work.

97 EVENODDMISMATCH *An even channel is in an odd slot in the queue, or vice versa.*
The channel gain queue on some A/D boards has a restriction that even channel numbers must be in even queue positions and odd channel numbers must be in odd queue positions.

98 M1RATEWARNING *Sampling speed to system MAY be too fast.*
The A/D board sampling speed you have requested may be too fast for the computer system bus transfer to complete before the next packet is ready for transfer. If this occurs, data will overrun and sample data will be garbled. This warning is inititaed whenever you request a sample rate over 625KHz and the sample set is larger than the FIFO buffer on the board. Your system may be able to handle the rate you reqest, but that is system dependent and only experiment will teach you what your computer is capable of.

200 INTERNALERR *Internal error occurred in library.*
An internal error occurred in the library. This should not happen. Please report to Computer Board's Technical Support Dept.

501 DOSBADFUNC *Invalid DOS function number specified *** Internal Error.*
An internal error occurred in the library. This should not happen. Please report to Computer Board's Technical Support Dept.

502 DOSFILENOTFOUND *Cannot find file.*
A file name that was passed to cbFileGetInfo() or cbFileRead() does not exist.

503 DOSPATHNOTFOUND *Cannot find path.*
A file name that was passed to cbFileGetInfo() or cbFileRead() contained an invalid path (directory) name.

504 DOSNOHANDLES *No DOS handles left.*
An attempt to open a file failed because there were no more DOS file handles available. There are too many open files. If you need to have more files open simultaneously add a line to your system's \CONFIG.SYS file that says files = 20

505 DOSACCESSDENIED *File Access denied*
An attempt was made to open a file that is locked. This error usualy indicates that another program (or another user on a network) has already opened this file. The file will remain locked until they close it.

506 DOSINVALIDHANDLE *Invalid handle - *** Internal Error.*
An internal error occurred in the library. This should not happen. Please report to Computer Board's Technical Support Dept.

507 DOSNOMEMORY *DOS has no more memory available.*
No memory is available. Try removing any TSR programs or device drivers that you are not using from memory.

515 DOSBADDRIVE *Invalid disk drive.*
An invalid drive was specified for a file operation.

518 DOSTOOMANYFILES *Too many disk files.*
There are too many files on your disk. This is an unlikely error. It will occur only if you are attempting to add greater than the number of entries allowed on the file allocation table.

519 DOSWRITEPROTECT *Disk is write protected.*
An attempt was made to write on a write-protected disk. This should only happen when writing to floppy disks. Remove the write-protect sticker from the disk and try again.

521 DOSDRIVENOTREADY *Disk drive is not ready.*
An attempt was made to write on a disk drive that isn't ready. This should only happen when writing to floppy disks. It indicates that there is no disk in the drive or the drive door is open.

525 DOSSEEKERROR *DOS seek error.*
This error indicates a serious error on the disk or drive.

529 DOSWRITEFAULT *DOS write error.*
This error indicates a serious error on the disk or drive.

530 DOSREADFAULT *DOS read error.*
This error indicates a serious error on the disk or drive.

531 DOSGENERALFAULT *DOS general fault.*
This error indicates a serious error on the disk or drive.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP