



## Introducing the Universal Library™

Congratulations and thank you for selecting the Universal Library! We believe it is the easiest and most comprehensive data acquisition software interface available anywhere.

The Universal Library (UL) includes programming libraries and components for developing 32-bit and 64-bit applications using Windows programming languages. The UL is easy to use, but significant documentation and explanation is still required to help new users get going, and to allow existing users to take advantage of the Library's powerful features.

The Universal Library Help is presented in three parts:

- **Universal Library User's Guide:** The *User's Guide* provides a general description of the Universal Library, offers an overview of the various features and functions of the library, and discusses how you can use the Universal Library features in different operating systems and [languages](#).  
  
The User's Guide also provides board-specific information relating to the features and functions that are included with the Universal Library.
- **Universal Library Function Reference:** The *Function Reference* details the features, usage, and options of the Universal Library functions and methods.
- **Example programs:** The examples programs demonstrate the use of many of the most frequently used functions, and are valuable learning tools. They are written for many popular languages. Each example program is fully functional, and provides an ideal starting place for your own programming effort. You can cut and paste from the example programs to create your own programs. It's easier to cut-and-paste pieces from a known, working program than to start writing everything from scratch.

In addition to the Universal Library Help, refer to the **ReadMe** files shipped with the Universal Library software for the latest information available.

Revision: 12.0

October, 2012

## Your new Measurement Computing product comes with a fantastic extra:

### MANAGEMENT COMMITTED TO YOUR SATISFACTION

Thank you for choosing a Measurement Computing product — and congratulations! You own the finest, and you can now enjoy the protection of the most comprehensive warranties and unmatched phone tech support. It's the embodiment of our mission:

- To provide data acquisition hardware and software that will save time and save money.

Simple installations minimize the time between setting up your system and actually making measurements. We offer quick and simple access to outstanding live FREE technical support to help integrate MCC products into a DAQ system.

### 30-day Money Back Guarantee

Any Measurement Computing Corporation product may be returned within 30 days of purchase for a full refund of the price paid for the product being returned. If you are not satisfied, or chose the wrong product by mistake, you do not have to keep it.

This warranty is in lieu of all other warranties, expressed or implied, including any implied warranty of merchantability or fitness for a particular application. The remedy provided herein is the buyer's sole and exclusive remedy. Neither Measurement Computing Corporation, nor its employees shall be liable for any direct or indirect, special, incidental or consequential damage arising from the use of its products, even if Measurement Computing Corporation has been notified in advance of the possibility of such damages.

## Redistribution, Trademark, and Copyright Information

### Redistributing a custom UL application

Customers can distribute the necessary runtime files (University Library driver files) for any application created using the Universal Library. Customers may not distribute any files that give the end user the ability to develop applications using the Universal Library.

### Trademark and copyright information

TracerDAQ, Universal Library, Measurement Computing Corporation, and the Measurement Computing logo are either trademarks or registered trademarks of Measurement Computing Corporation.

Windows, Microsoft, and Visual Studio are either trademarks or registered trademarks of Microsoft Corporation

LabVIEW is a trademark of National Instruments.

CompactFlash is a registered trademark of SanDisk Corporation.

All other trademarks are the property of their respective owners.

Information furnished by Measurement Computing Corporation is believed to be accurate and reliable. However, no responsibility is assumed by Measurement Computing Corporation neither 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 Measurement Computing Corporation.

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 Measurement Computing Corporation.

**Notice:** Measurement Computing Corporation does not authorize any Measurement Computing Corporation product for use in life support systems and/or devices without prior written consent from Measurement Computing Corporation. Life support devices/systems are devices or systems which, a) are intended for surgical implantation into the body, or b) support or sustain life and whose failure to perform can be reasonably expected to result in injury. Measurement Computing Corporation products are not designed with the components required, and are not subject to the testing required to ensure a level of reliability suitable for the treatment and diagnosis of people.

## About this Document

The online help includes information about the Windows Universal Library API, Universal Library for .NET, and InstaCal. The earliest revisions discussed in this document include changes for product versions 5.20 and later.

## Contact Measurement Computing Corporation

You can contact us via the following methods:

**Address:** Measurement Computing Corporation  
10 Commerce Way  
Norton, MA 02766

**Telephone:** 508-946-5100

**Fax:** 508-946-9500

**Email:** Technical Support: [techsupport@measurementcomputing.com](mailto:techsupport@measurementcomputing.com)  
Sales: [sales@measurementcomputing.com](mailto:sales@measurementcomputing.com)  
All other correspondence: [info@measurementcomputing.com](mailto:info@measurementcomputing.com)

**Visit our Web site:** [www.measurementcomputing.com](http://www.measurementcomputing.com)

# Universal Library Overview

The Universal Library is the software required to write your own programs for use with any Measurement Computing data acquisition and control board. The Universal Library is universal in three ways:

- **Universal across boards:** The library contains high-level functions for all of the common operations for all boards. Although each of the boards has different hardware, the Universal Library hides these differences from your program. For example, a program written for use with one A/D board will work without modification with a different A/D board.
- **Universal across languages:** The Universal Library provides identical sets of functions and arguments for each supported language. If you switch languages, you will not have to learn a new library, with new syntax, and different features.

If you are programming for the .NET Framework, you will find that the Universal Library for .NET has the same "look and feel" as the Universal Library API, and is just as easy to program.

32-bit languages supported by the Universal Library at the time the library was released are listed below:

Microsoft Windows Languages	.NET Languages
Visual Basic	VB .NET
Visual C++	C# .NET
Quick C for Windows	
Microsoft C	

- **Universal across platforms:** The Universal Library provides the same sets of functions for the following operating systems:
  - Windows 8 (32-bit and 64-bit<sup>1</sup>)
  - Windows 7 (32-bit and 64-bit<sup>1</sup>)
  - Windows Vista (32-bit and 64-bit<sup>1</sup>)
  - Windows XP (32-bit (SP2) and 64-bit<sup>1</sup>)

<sup>1</sup> 64-bit operating systems support Measurement Computing USB, WLS, WEB, and most PCI hardware.

**Note:** Visual Studio 2005 or later is required to develop .NET applications and to run the example programs.

## Hardware requirements

Supported Measurement Computing data acquisition hardware:

- 64-bit operating systems: USB, WLS, WEB, and most PCI hardware are supported.
- 32-bit operating systems: Windows XP supports USB, PCI, WLS, WEB, and PC-CARD hardware. Windows 7 and Windows Vista support USB, PCI, WLS, and WEB hardware. Note that ISA, PC104, and PCMCIA devices (PC-CARD and PCM hardware) are not supported for Windows Vista or Windows 7.

## Installing the Universal Library and InstaCal

**Note:** The Microsoft .NET Framework 2.0 be must be installed on the system before you install the Universal Library and InstaCal.

Perform the following procedure to install the Universal Library and InstaCal:

1. Place the *Measurement Computing Data Acquisition Software* CD in your CD drive.

The **MCC DAQ** dialog opens.

2. Select **InstaCal & Universal Library** and click the **Install** button.
3. Follow the installation instructions as prompted.

InstaCal is a powerful installation, test, and calibration software package that is installed with the Universal Library application. Refer to the *Quick Start Guide* that shipped with your hardware for examples of using InstaCal with Measurement Computing's DEMO-BOARD.

## **Universal Library support for .NET**

Universal Library support for .NET requires that the Microsoft .NET Framework 2.0 be installed on the system before you install the Universal Library.



## CB.CFG Configuration File

Board-specific information is stored in the CB.CFG configuration file which is read by Universal Library. The CB.CFG file is created the first time you run InstaCal, and is updated each time you add DAQ hardware or modify device configuration settings.

The Universal Library cannot run without the CB.CFG file. *For this reason, you must use InstaCal to install/remove DAQ hardware and to configure device settings.*

## Redistributing a Custom UL Application

The easiest way to distribute an application written with the Universal Library is to include a copy of Measurement Computing's InstaCal installation package with the application. Instruct the end user to install InstaCal before installing the application.

Some developers may want to integrate the installation of the required Universal Library drivers into the custom application's installation. This should only be attempted by developers experienced in installation development.

Following is an overview of the two methods.

### Distributing InstaCal in addition to your custom UL application

If you create an application using the Universal Library, you may distribute the necessary runtime files (Universal Library driver files) with the application royalty free. These files may be installed from Measurement Computing's InstaCal installation package. To distribute a custom UL application, provide the end user with two CDs or disks:

- One CD or disk that contains Measurement Computing's InstaCal application. InstaCal must be installed before the custom UL application.
- One CD or disk that contains the setup program for their custom VB or C++ application.

You may not distribute any files that give the end user the ability to develop applications using the Universal Library.

### Integrating InstaCal into your custom UL installation CD or disk

For developers who wish to distribute their application on one CD, refer to the *Universal Library Distribution Guide*. This document contains procedures to merge the setup programs for both InstaCal and the custom UL application into one setup program that you can distribute on one CD or disk. The merging process is complicated – only experienced programmers should attempt to do this.

When you install the software, the *Universal Library Redistribution Guide* (ULRedistribution.pdf) is copied to the default installation directory "C:\Program Files\Measurement Computing\DAQ\Documents" on your local drive.

## Getting Started

The Universal Library is callable from Visual Basic and Visual C++. A list of the languages currently supported by the Universal Library is provided in the "[Universal Library Overview](#)" topic. Additionally, the UL is now callable from any language supported by the .NET Framework.

Before starting your application, perform the following:

1. Set up and test your board with InstaCal. The Universal Library will not function until InstaCal has created a configuration file (CB.CFG).
2. Run the example programs for the language in which you program.

## Example programs

You can install example programs for supported languages when you install the Universal Library software. If selected, the example programs are installed into the following installation subdirectories:

- C
- C#
- CWIN
- VB.NET
- VBWIN

On Windows 8, Windows 7, and Windows Vista, the example programs are installed by default to  
\\Users\\Public\\Documents\\Measurement Computing\\DAQ.

On Windows XP, the example programs are installed by default to \\Program Files\\Measurement Computing\\DAQ.

**Note:** Visual Studio 2005 or later is required to run the example programs.

When you install the example programs, an **Examples** shortcut is added to the directory where you installed the Universal Library software. When selected, the directory containing the example programs opens in Windows Explorer.

For a complete list of example programs, refer to the *Universal Library Function Reference*. This document includes tables that list the UL and UL for .NET example programs. Each table contains the name of the sample program and the functions that the program demonstrates. Click on a link below to display each table.

- [UL example programs sorted by program name](#)
- [UL example programs sorted by function call](#)
- [UL for .NET example programs sorted by program name](#)
- [UL for .NET example programs sorted by method call](#)

## Multi-threading

Only one application program that calls the Measurement Computing driver can be running at a time.

For example, when you are running a program created with the Universal Library, you cannot change any hardware configuration settings with the InstaCal program until you first stop the UL program. This is because InstaCal stores hardware configuration settings in a file (cb.cfg) which is read by the Universal Library when you run an application. To change device settings, stop the UL application and run InstaCal.

## Differences between the UL and UL for .NET

The table below lists the main differences between the Universal Library and the Universal Library for .NET.

	Universal Library	Universal Library for .NET
<b>Board Number</b>	The board number is included as a parameter to the board functions.	An MccBoard class is created for each board installed, and the board number is passed to that board class.
<b>Functions</b>	Set of function calls defined in a header.	Set of methods in the MccBoard and MccService classes. To access a method, instantiate a <a href="#">UL for .NET class</a> and call the appropriate method using that class.
<b>Constants</b>	Constants are defined and assigned a value in the "header" file.	Constants are defined as <a href="#">enumerated types</a> .
<b>Return value</b>	The return value is an Error code.	The return value is an <a href="#">ErrorInfo object</a> that contains the error's number and message.

### Board Number

In a .NET application, you can program multiple boards by creating an MccBoard class for each board installed.

```
Board0=new MccBoard(0)
Board1=new MccBoard(1)
Board2=new MccBoard(2)
```

The board number may be passed into the MccBoard class, which eliminates the need to include the board number as a parameter to the board methods.

### MCC classes

To use board-specific Universal Library functions inside a .NET application, you use methods of the appropriate class. UL for .NET classes include:

Universal Library for .NET Class	Description
<a href="#">MccBoard</a>	Accesses board-related Universal Library functions.
<a href="#">ErrorInfo</a>	Utility class for storing and reporting error codes and messages.
<a href="#">BoardConfig</a>	Gets and sets board configuration settings.
<a href="#">CtrConfig</a>	Gets and sets counter configuration settings.
<a href="#">DioConfig</a>	Gets and sets digital I/O configuration settings.
<a href="#">ExpansionConfig</a>	Gets and sets expansion board configuration settings.
<a href="#">GlobalConfig</a>	Gets and sets global configuration settings.
<a href="#">MccService</a>	Accesses utility Universal Library functions.
<a href="#">DataLogger</a>	Reads and converts binary log files.

### Methods

Methods are accessed through the class containing them. The following example demonstrates how to call the AIn() method from within a 32-bit Windows application and also from a .NET application:

VB Application using CBW32.DLL	VB Application using CBW32.DLL
<pre>Dim Board As Integer Dim Channel As Integer Dim Range As Integer Dim ULStat As Integer Dim DataValue As Short  Board = 0 Channel = 0 Range = BIP5VOLTS; ULStat = cbAIn(Board, Channel, Range, DataValue)</pre>	<pre>Dim Board0 as MccBoard Board0 = new MccDaq.MccBoard(0) Dim Channel As Integer Dim Range As MccDaq.Range Dim ULStat As ErrorInfo Dim DataValue As UInt16  Channel = 0 Range = Range.BIP5VOLTS; ULStat = Board0.AIn(Channel, Range, DataValue)</pre>

## Enumerated Types

Instead of using constants such as BIP5VOLTS, the Universal Library for .NET uses enumerated types. An enumerated type takes settings such as range types, scan options or digital port numbers and puts them into logical groups.

Some examples are:

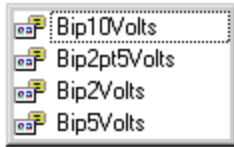
```
Range.Bip5Volts
Range.Bip10Volts
Range.Uni5Volts
Range.Uni10Volts

ScanOptions.Background
ScanOptions.Continuous
ScanOptions.BurstMode

TimeZone.GMT
FileType.Text
```

If you are programming inside of Visual Studio .NET, the types that are available for a particular enumerated value display automatically when you type code:

```
int Gain =Range.
```



## Error Handling

For .NET applications, the return value for the Universal Library functions is an object ([ErrorInfo](#)) rather than a single integer value. The ErrorInfo object contains both the number of the error that occurred, as well as associated error message. Within a .NET application, error checking may be performed as follows:

```
ULStat=Board0.AIn(Channel, Range, DataValue)
'check the numeric value of ULStat
If Not ULStat.Value = ErrorInfo.ErrorCode.NoErrors Then
'if there was an error, then display the error message
MsgBox ULStat.Message
EndIf
```

## Service Methods

You can access other Universal Library functions that are not board-specific through the [MccService class](#). This class contains a set of static methods you can access directly, without having to instantiate an MccService object. The following examples demonstrate Library calls to .NET memory management methods.

```
WindowHandle=MccService.WinBuffAlloc(1000)
MccService.WinBuffFree(WindowHandle)
```

## Configuration Methods

In 32-bit Windows applications, you access board configuration information by calling the cbGetConfig() and cbSetConfig() API functions. In .NET applications, you access board configuration information through separate classes, such as [MccBoard](#) and its [BoardConfig](#), [CtrConfig](#), [DioConfig](#), and [ExpansionConfig](#) properties. Each configuration item has a separate get and set method. Below are some examples of how to access board configuration within a .NET application.

```
MccDaq.ErrorInfo UIStat = Board0.BoardConfig.GetRange(DevNumber, RangeValue)
MccDaq.ErrorInfo UIStat = Board1.DioConfig.GetNumBits(DevNumber, Number)
MccDaq.ErrorInfo UIStat = Board2.CtrConfig.GetCtrType(DevNumber, CounterType)
MccDaq.ErrorInfo UIStat = Board3.BoardConfig.SetClock(DevNumber, ClockSource)
MccDaq.ErrorInfo UIStat = Board4.ExpansionConfig.SetCJCChan(DevNumber, CjcChan)
```

# DataLogger Methods

In 32-bit Windows applications, you access the information contained in binary files logged from MCC hardware by calling the Data Logger API functions. In .NET applications, you access this information by calling the [DataLogger class](#) and its methods.

The following example demonstrates how to retrieve the name of the first binary log file using the `cbLogGetFileName()` function and `GetFileName()` method.

C/C++ application	C# application
<pre>char    filename[50]; char*   path = "C:\\LogData"; int     retval = 0;  retval = cbLogGetFileName (GETFIRST, path, filename);</pre>	<pre>string    filename = new string('\0', 50); char*     path = "C:\\LogData"; ErrorInfo status;  status = DataLogger.GetFileName(MccService.GetFirst, ref path, ref filename);</pre>

## Using the Universal Library in Windows

All 32-bit applications (including console applications) access the 32-bit Windows Dynamic Link Library (DLL) version of the Universal Library (CBW32.DLL). Example programs that illustrate the use of CBW32.DLLs are provided for each supported language.

## Buffer management

The Universal Library contains numerous functions and methods for managing Windows global memory buffers. Click on a link below for a list of the available functions and methods.

- [Windows memory management functions\(\)](#)
- [Windows memory management methods\(\)](#)

## Real-Time Acquisition Under Windows

Real-time acquisition is available for Windows. To operate at full speed in Windows, the A/D board must have an onboard FIFO buffer. All of our advanced designs have FIFO buffers, including our PCI-DAS boards (except for the PCI-DAS08), and many of our CIO- boards, such as the CIO-DAS80x, CIO-DAS160x, CIO-DAS140x, and CIO-DAS16/330x. All of these data acquisition boards will operate at full speed in Windows.

Applying software calibration factors in real time on a per-sample basis eats up machine cycles. If your CPU is slow, or if processing time is at a premium, withhold calibration until after the acquisition run is complete. Turning off real-time software calibration saves CPU time during a high speed acquisition run.

## Multi-threading

Only one application program that calls the Measurement Computing driver can be running at a time.

For example, when you are running a program created with the Universal Library, you cannot change any hardware configuration settings with the InstaCal program until you first stop the UL program. This is because InstaCal stores hardware configuration settings in a file (cb.cfg) which is read by the Universal Library when you run an application. To change device settings, stop the UL application and run InstaCal.

## Processor Speed

Processor speed remains a factor for DMA transfers and for real-time software calibration. Processors of less than 150 megahertz (MHz) Pentium class may impose speed limits below the capability of the board (refer to board-specific information).

If your processor is less than a 150 MHz Pentium and you need an acquisition speed in excess of 200 kilohertz (kHz), use the NOCALIBRATEDATA option to turn off real-time software calibration and save CPU time. After the acquisition is run, calibrate the data with cbACalibrateData().

## Visual Basic for Windows

To use the Universal Library with Visual Basic®, include the Universal Library declaration file CBW.BAS in your program. Include the file as a module in the project, or include it by reference inside those Forms which call into the Universal Library. Once the declarations for the Universal Library have been added to your project, call the library functions from any Form's event handlers.

When using Visual Basic, CBW.BAS references CBW32.DLL to call Universal Library functions. This is accomplished with conditional compile statements.

For Visual Basic 6.0 and older, Windows memory buffers cannot be mapped onto arrays. As a consequence, the [cbWinArrayToBuf\(\)](#) and [cbWinBufToArray\(\)](#) functions must be used to copy data between arrays and Windows buffers.

### Example:

```
Count = 100
MemHandle = cbWinBufAlloc (Count)
cbAInScan (.....,MemHandle,...)
For i = 0 To Count
    Print dataArray(i)
Next i
cbWinBufFree (MemHandle)
```

## CB.CFG Locations with Visual Basic

All programs that use the Universal Library read the CB.CFG configuration file. Include a copy of the CB.CFG configuration file with any compiled stand alone Visual Basic programs that you wish to distribute to another machine or directory.

## Visual Basic Example Programs

A complete set of Visual Basic example programs is included in the VBWIN folder of the Universal Library installation directory.

Each program illustrates the use of a Universal Library function from within a Visual Basic program. The .FRM files contain the programs, and the corresponding .VBP or .MAK files are the project files used to build the programs for Visual Basic.

## **Microsoft Visual C++**

To use the Universal Library with MS Visual C++, include the Universal Library header file CBW.H in your C/C++ program and add the library file CBW32.LIB to your library modules for linking to the CBW32.DLL.

### **Microsoft Visual C++ Example Programs**

The CWIN folder of the Universal Library installation directory contains three sample programs - Wincai01, Wincai02 and Wincai03. Each program is an example of a simple C program that calls a few of the Universal Library functions from a Windows application. Use the .DSP project files to build a 32 bit application.

The non-Windows C examples in the C folder of the installation directory provide a more complete set of examples. You can compile these programs as 32-bit console applications for Windows by using the MAKEMC32.BAT file



# Universal Library Language Interface

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, and especially to the example programs for each language. This manual is brief with respect to details of language use and syntax. Review the [example programs](#) for more detailed information. The example programs for each language are located in the installation directory.

## Function arguments

Each library function takes a list of arguments and most return an [error code](#). Some functions also return data via their arguments. For example, one of the arguments for [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:" section of the function description.

## 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. For example, [cbLogGetPreferences\(\)](#) takes an argument called TimeFormat that must be set to either TIMEFORMAT\_12HOUR or TIMEFORMAT\_24HOUR. These constant names are defined and assigned a value in the "header" 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. 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.

Some options can have an alternative value, such as BACKGROUND and FOREGROUND. If an option can have more than one value, one of the values is designated as the default.

Individual options can be turned on by adding them to the Options argument. 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.

## Error handling

Most library functions return an error code. If no errors occurred during a library call, 0 (or [NOERRORS](#)) is returned as the error code; otherwise, one of the codes listed in [Error Codes](#) is returned.

If a non-zero error code is returned, you can use [cbGetErrMsg\(\)](#) to convert the error code to a specific error message. As an alternative to checking the error code after each function call, you can turn on the library's internal error handling with [cbErrHandling\(\)](#).

## 16-bit values using a signed integer data type

When using functions that require 16-bit values, the data is normally in the range 0 to 65,535. However, some programming languages, such as Visual Basic®, only provide signed data types. When using signed integers, reading values above (32,767) can be confusing.

The number (32,767) is equivalent to (0111 1111 1111 1111) binary. The next increment (1000 0000 0000 0000) binary has a decimal value of (-32,768). The maximum value (1111 1111 1111 1111) binary translates to (-1) decimal. Keep this in mind if you are using languages that don't support unsigned integers.

There is additional information on this topic in the *Universal Library Function Reference*. Also, refer to the documentation supplied with your language compiler.

## 32-bit values using a signed long data type

When using functions that require 32-bit values, the data is normally in the range 0 to 4,294,967,295. However, some programming languages, such as Visual Basic®, only provide signed data types. When using signed integers, reading values above (2,147,483,647) can be confusing. The number (2,147,483,647) is equivalent to (0111 1111 1111 1111 1111 1111 1111 1111) binary. The next increment (1000 0000 0000 0000 0000 0000 0000 0000) binary has a decimal value of (-2,147,483,648). The maximum value (1111 1111 1111 1111 1111 1111 1111 1111) binary translates to (-1) decimal. Keep this in mind if you are using languages that don't support unsigned longs.

## Configuring a UL for .NET Project

Programming the Universal Library API is available through the various languages supported by the Microsoft .NET Framework. All .NET applications access the Universal Library (CBW32.DLL and CBW64.DLL) through the MccDag .NET assembly (MCCDAQ.DLL). The MccDag assembly provides an interface that exposes each Universal Library function that is callable from the .NET language.

The Universal Library for .NET is designed to provide the same "look and feel" as the Universal Library for Windows. This design makes it easier to port over existing data acquisition programs, and minimizes the learning curve for programmers familiar with the Universal Library API.

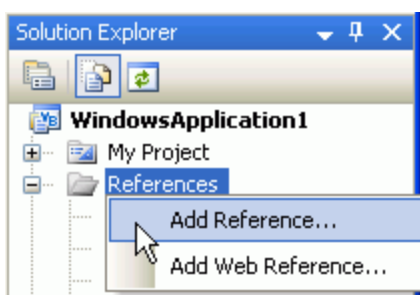
In the Universal Library for .NET, each function is exposed as a class method with virtually the same parameter set as their UL counterparts.

## Referencing the MccDag Namespace in a .NET Project

In a .NET application, there are no header files to include in your project. You define methods and constants by adding the MccDag assembly, or [Namespace](#), as a reference to your project. You access UL for .NET methods through a class that has the Universal Library as a member.

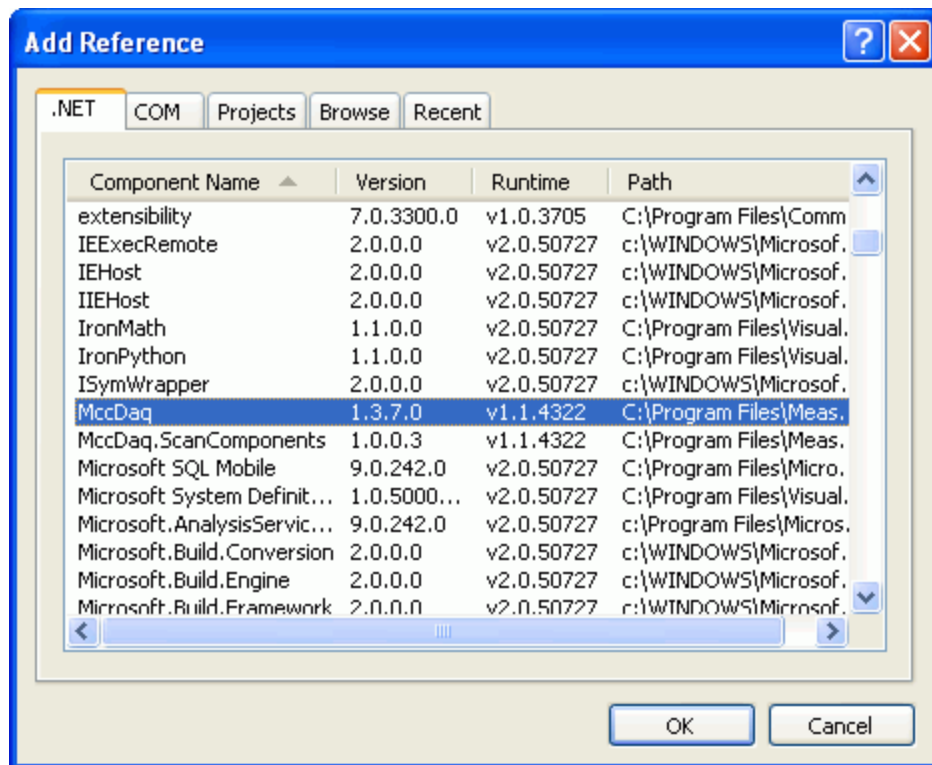
To add the MccDag Namespace as a reference in a Visual Studio .NET project:

1. Start a new Visual Basic or C# project in Visual Studio .NET.
2. From the Visual Studio .NET Solution Explorer window, right-click on **References** and select **Add Reference**.

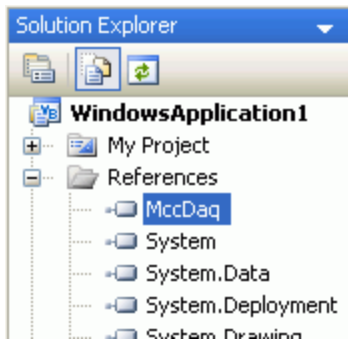


The **Add Reference** window appears.

3. From the .NET tab, select the **MccDag** option from the displayed list of .NET assemblies and click on the **OK** button.



**MccDag** appears under the **References** folder in the Solution Explorer window.



The MccDaq Namespace is now referenced by your Visual Studio .NET project.

# Universal Library for .NET Language Interface

The MccDaq [Namespace](#) provides an interface that exposes each Universal Library for .NET function as a member of a class with virtually the same parameters set as their UL counterparts.

When you develop a .NET application that uses the Universal Library, you add the MccDaq Namespace as a reference to your project. There are no "header" files in a .NET project.

The MccDaq Namespace contains the classes and enumerated values by which your UL for .NET applications can access to the Universal Library data types and functions.

The MccDaq Namespace contains five main classes:

- MccBoard class
- ErrorInfo class
- MccService class
- GlobalConfig class
- DataLogger class

The MccDaq assembly allows you to design *Common Language Specification (CLS)* - compliant programs. A CLS-compliant program contains functions that can be called from any existing or future language developed for the Microsoft .NET Framework. Using CLS-compliant data types ensures future compatibility.

## MccBoard class

The MccBoard class provides access to all of the methods for data acquisition and properties providing board information and configuration for a particular board.

### Class constructors

The MccBoard class provides two constructors – one which accepts a board number argument and one with no arguments.

The following code examples demonstrate how to create a new instance of the MccBoard class using the latter version with a default board number of 0:

Visual Basic

```
Private DaqBoard As MccDaq.MccBoard
New MccDaq.MccBoard(0)
```

C#

```
private MccDaq.MccBoard DaqBoard;
DaqBoard = new Mccdaq.MccBoard(0);
```

The following code example demonstrates how to create a new instance of the MccBoard class with the board number passed to it:

Visual Basic

```
Private DaqBoard As MccDaq.MccBoard
DaqBoard = New MccDaq.MccBoard(BoardNumber)
```

C#

```
private MccDaq.MccBoard DaqBoard;
DaqBoard = new Mccdaq.MccBoard(BoardNumber);
```

### Class properties

The MccBoard class also contains six properties that you can use to examine or change the configuration of your board:

Properties	Description
<a href="#">BoardName</a>	Name of the board associated with an instance of the <a href="#">MccBoard</a> class.
<a href="#">BoardNum</a>	Number of the board associated with an instance of the MccBoard class.
<a href="#">BoardConfig</a>	Gets a reference to a cBoardConfig class object. Use this class reference to get or set various board settings.
<a href="#">CtrConfig</a>	Gets a reference to a cCtrConfig class object. Use this class reference to get or set various counter settings.
<a href="#">DioConfig</a>	Gets a reference to a cDioConfig class object. Use this class reference to get or set various digital I/O settings.
<a href="#">ExpansionConfig</a>	Gets a reference to a cExpansionConfig class object. Use this class reference to get or set various expansion board settings.

The configuration information for all boards is stored in the CB.CFG file, and is loaded from CB.CFG by all programs that use the library.

Class methods

The MccBoard class contains close to 80 methods that are equivalents of the function calls used in the standard Universal Library. The MccBoard class methods have virtually the same parameters set as their UL counterparts.

The following code examples demonstrate how to call the AIn() method of the MccBoard object MccDag:

```
Visual      ULStat = DaqBoard.AIn(Chan, Range, DataValue)
Basic
C#          ULStat = DaqBoard.AIn(Chan, Range, out DataValue);
```

Many of the arguments are [MccDag enumerated values](#). Enumerated values take settings such as range types or scan options and put them into logical groups. For example, to set a range value, reference a value from the [MccDag.Range](#) enumerated type, such as Range.Bip5Volts.

The *Universal Library Function Reference* contains detailed information about all MccBoard class methods.

ErrorInfo class

Most UL methods return ErrorInfo objects. These objects contain two properties that provide information on the status of the method called:

- [MccService.ErrHandling\(\)](#) method sets the manner of reporting and handling errors for all methods.
- [ErrorInfo.Message](#) property gets the text of the error message associated with a specific error code.
- [ErrorInfo.Value](#) property gets the named constant value associated with the ErrorInfo object.

The ErrorInfo class also includes Error Code enumerated values that define the error number and associated message which can be returned when you call a method. Click [here](#) for information about the properties, methods, and enumerated constants in the Error Info class.

MccService class

The [MccService Class](#) contains all members for calling utility UL functions.

The following code examples demonstrate how to call a UL for .NET memory management method from within a Universal Library program:

```
WindowHandle=MccService.WinBuffAlloc(1000)
MccService.WinBuffFree(WindowHandle)
```

GlobalConfig class

The [GlobalConfig class](#) contains members that are used to obtain global configuration information.

DataLogger class

The [DataLogger Class](#) contains all members for reading and converting binary log files.

MccDag enumerations

The MccDag Namespace contains enumerated values which are used by many of the methods available from the MccDag classes. Refer to specific method descriptions in the *Universal Library Function Reference* for the values of each enumerated type.

<a href="#">MccDag.BCDMode</a>	Lists BCD mode options (enabled/disabled).
<a href="#">MccDag.C8254Mode</a>	Lists all of the operating modes for 8254 counters.
<a href="#">MccDag.C8536OutputControl</a>	Lists all of the types of output from an 8536 counters.
<a href="#">MccDag.C8536TriggerType</a>	Lists all of the options for specifying the trigger type for 8536 counters.
<a href="#">MccDag.C9513OutputControl</a>	List all of the types of output from 9513 counters.
<a href="#">MccDag.ChannelType</a>	List all of the options for setting the channel type.
<a href="#">MccDag.CompareValue</a>	List all options for comparing values while configuring a 9513 counter.
<a href="#">MccDag.ConnectionPin</a>	Defines the connector pins to associate with the signal type and direction when calling the SelectSignal function.
<a href="#">MccDag.CountDirection</a>	Defines the count direction when configuring counters.
<a href="#">MccDag.CountEdge</a>	Defines the edge used for counting
<a href="#">MccDag.CounterControl</a>	Defines the possible state of each counter channel (enabled/disabled).
<a href="#">MccDag.CounterDebounceMode</a>	Lists all options for specifying the debounce mode.
<a href="#">MccDag.CounterDebounceTime</a>	Lists all options for specifying the debounce time.
<a href="#">MccDag.CounterEdgeDetection</a>	Lists all options for specifying the edge detection.
<a href="#">MccDag.CounterMode</a>	List all options for specifying the counter mode.

<a href="#"><u>MccDag.CounterRegister</u></a>	Lists all of the register names used to load counters.
<a href="#"><u>MccDag.CounterSource</u></a>	Lists all counter input sources.
<a href="#"><u>MccDag.CountingMode</u></a>	Lists all valid modes for a C7266 counter configuration.
<a href="#"><u>MccDag.CtrlOutput</u></a>	Lists all options for linking counter 1 to counter 2.
<a href="#"><u>MccDag.DACUpdate</u></a>	Lists the available DAC update modes.
<a href="#"><u>MccDag.DataEncoding</u></a>	Lists the format of the data that is returned by a counter.
<a href="#"><u>MccDag.DigitalLogicState</u></a>	Defines all digital logic states.
<a href="#"><u>MccDag.DigitalPortDirection</u></a>	Configures a digital I/O port as input or output.
<a href="#"><u>MccDag.DigitalPortType</u></a>	Defines all digital port types.
<a href="#"><u>MccDag.DTMode</u></a>	Lists all modes to transfer to/from the memory boards.
<a href="#"><u>MccDag.ErrorHandling</u></a>	Defines all error handling options.
<a href="#"><u>MccDag.ErrorReporting</u></a>	Defines all error reporting options.
<a href="#"><u>MccDag.EventParameter</u></a>	Lists all options for latching data.
<a href="#"><u>MccDag.EventType</u></a>	Lists all available event conditions.
<a href="#"><u>MccDag.FieldDelimiter</u></a>	Defines the delimiter character used to separate fields in a converted log file.
<a href="#"><u>MccDag.FileType</u></a>	Defines the file type used to convert a binary log file.
<a href="#"><u>MccDag.FlagPins</u></a>	Lists all signals types that can be routed to the FLG1 and FLG2 pins on the 7266 counters.
<a href="#"><u>MccDag.FunctionType</u></a>	Lists all valid function types used with data acquisition functions.
<a href="#"><u>MccDag.GateControl</u></a>	Lists all of the gating modes for configuring a 9513 counter.
<a href="#"><u>MccDag.IdleState</u></a>	Lists all options for specifying the idle state.
<a href="#"><u>MccDag.IndexMode</u></a>	Lists the actions to take when the index signal is received by a 7266 counter.
<a href="#"><u>MccDag.InfoType</u></a>	Lists the the configuration information to be used with the MccBoard class configuration functions.
<a href="#"><u>MccDag.LoggerUnits</u></a>	Lists the options used to specify the units for analog data in a binary file.
<a href="#"><u>MccDag.OptionState</u></a>	Enables or disables various options.
<a href="#"><u>MccDag.PrimaryBitConfigPortType</u></a>	Defines digital port types for bit level configuration.
<a href="#"><u>MccDag.PrimaryDigitalPortType</u></a>	Defines digital port types for bit level input/output methods.
<a href="#"><u>MccDag.Quadrature</u></a>	Lists all of the resolution multipliers for quadrature input.
<a href="#"><u>MccDag.Range</u></a>	Defines the set of ranges within the UL for A/D and D/A operations.
<a href="#"><u>MccDag.RecycleMode</u></a>	Lists the recycle mode options for 9513 and 8536 counters.
<a href="#"><u>MccDag.Reload</u></a>	Lists the options for reloading the 9513 counter.
<a href="#"><u>MccDag.ScanOptions</u></a>	Lists the available scan options for paced input/output functions.
<a href="#"><u>MccDag.SetpointFlag</u></a>	Lists the options for setting the flag type.
<a href="#"><u>MccDag.SetpointOutput</u></a>	Lists the options for setting the output source.
<a href="#"><u>MccDag.SignalDirection</u></a>	Lists all of the directions available from a specified signal type assigned to a connector pin.
<a href="#"><u>MccDag.SignalPolarity</u></a>	Lists all available polarities for a specified signal.
<a href="#"><u>MccDag.SignalSource</u></a>	List all of the signal sources of the signal from which the frequency will be calculated.
<a href="#"><u>MccDag.SignalType</u></a>	Lists all signal types associated with a connector pin on boards supporting a DAQ-Sync connector.
<a href="#"><u>MccDag.SoftwareTriggerType</u></a>	Defines trigger types for software triggering.
<a href="#"><u>MccDag.StatusBits</u></a>	List all status bits available when reading counter status.
<a href="#"><u>MccDag.TEDSReadOptions</u></a>	Lists the options for reading data from a TEDS sensor into an array.
<a href="#"><u>MccDag.TempScale</u></a>	Lists valid temperature scales that the input can be converted to.
<a href="#"><u>MccDag.ThermocoupleOptions</u></a>	Specifies whether or not to apply smoothing to temperature readings.
<a href="#"><u>MccDag.TimeFormat</u></a>	Defines the formats for displaying time stamp data.
<a href="#"><u>MccDag.TimeOfDay</u></a>	List all time of day options for initializing a 9513 counter.
<a href="#"><u>MccDag.TimeZone</u></a>	Defines the available time zones to store time stamp data.
<a href="#"><u>MccDag.TriggerEvent</u></a>	Lists all options for specifying the trigger event.
<a href="#"><u>MccDag.TriggerSensitivity</u></a>	Lists all options for specifying the trigger sensitivity.
<a href="#"><u>MccDag.TriggerSource</u></a>	Lists all options for specifying the trigger source.
<a href="#"><u>MccDag.TriggerType</u></a>	List all valid trigger types for the <a href="#"><u>MccBoard.SetTrigger</u></a> method.
<a href="#"><u>MccDag.VInOptions</u></a>	Lists all options for specifying voltage input options.
<a href="#"><u>MccDag.VOutOptions</u></a>	Lists all options for specifying voltage output options.

Many of the Universal Library for .NET methods are overloaded to provide for signed or unsigned data types as parameters. The [AConvertData\(\)](#) function is shown below using both signed and unsigned data types.

VB .NET	<pre>Public Function AConvertData(ByVal numPoints As Integer, ByRef adData As Short, ByRef chanTags As Short) As MccDaq.ErrorInfo Member of MccDaq.MccBoard  Public Function AConvertData(ByVal numPoints As Integer, ByRef adData As System.UInt16, ByRef chanTags As System.UInt16) As MccDaq.ErrorInfo Member of MccDaq.MccBoard</pre>
C# .NET	<pre>public MccDaq.ErrorInfo AConvertData (System.Int32 numPoints, System.Int16 adData, System.Int16 chanTags) Member of MccDaq.MccBoard  public MccDaq.ErrorInfo AConvertData(System.Int32 numPoints, System.UInt16 adData, System.UInt16 chanTags) Member of MccDaq.MccBoard</pre>

For most data acquisition applications, unsigned data values are easier to manage. However, since Visual Basic (version 6 and earlier) does not support unsigned data types, it may be easier to port these programs to .NET if the signed (Int16) data types are used for the function parameters. For additional information on using signed data types, refer to [16-bit values using a signed integer data type](#).

The short (Int16) data type is Common Language Specification (CLS) compliant, is supported in VB, and will be included in any future .NET language developed for the .NET Framework. Using CLS-compliant data types ensures future compatibility. Unsigned data types (UInt16) are not CLS-compliant, but are still supported by various .NET languages, such as C#.

## Analog Input Hardware

All devices with analog inputs support the [cbAIn\(\)/ AIn\(\)](#) and [cbAInScan\(\)/ AInScan\(\)](#) functions. If using an expansion device, refer to specific hardware information for information regarding whether [cbAInScan\(\)/AInScan\(\)](#) are supported.

When hardware paced A/D conversion is not supported, [cbAInScan\(\)/ AInScan\(\)](#) loops through software paced conversions. The scan executes at the maximum speed possible. The speed varies with CPU speed. The only valid option in this case is [CONVERTDATA](#).

## Concurrent analog input and output

Concurrent analog input and output scans are supported on devices with both paced analog inputs and outputs. These devices allow operations with analog input functions ([cbAInScan\(\)/ AInScan\(\)](#) and [cbAPretrig\(\)/APretrig\(\)](#)) and analog output functions ([cbAOutScan\(\)/AOutScan\(\)](#)) to overlap without having to call [cbStopBackground\(\)/StopBackground\(\)](#) between the start of the input and output scans.

## Trigger support

### Digital Trigger

If trigger support is "Polled gate" (as opposed to "Hardware"), you implement a trigger by gating the on-board pacer; doing so disables the on-board pacer. The trigger input is then polled continuously until the trigger occurs. When the trigger occurs, the software disables the gate input so that the pacer is not affected when the trigger returns to its original state. Acquisition continues until the requested number of samples is acquired. There are two side effects to this type of trigger:

- The polling portion of the function does not occur in the background, even if the BACKGROUND option is specified (although the actual data acquisition occurs).
- The trigger does not necessarily occur on the rising edge. Acquisition can start at any time after the function is called if the trigger input is at "active" level. For this reason, it is best to use a trigger that goes active for a much shorter time than it is inactive.

### Analog Trigger

Set up the trigger levels for an analog trigger using the [cbSetTrigger\(\)](#) function / [SetTrigger\(\)](#) method, and pass the appropriate values to the HighThreshold and LowThreshold arguments.

For most devices that support analog triggering, you can calculate the HighThreshold and LowThreshold values by passing the required trigger voltage level and the appropriate Range to the [cbFromEngUnits\(\)](#) function/[FromEngUnits\(\)](#) method.

However, for some devices, you must manually calculate HighThreshold and LowThreshold. If a device requires manual calculation, that information is included in the Trigger information for the specific product in this section. The procedure for manually calculating these values is described in the [cbSetTrigger\(\)](#) function/[SetTrigger\(\)](#) method sections of the *Universal Library Function Reference*.

### Pretrigger Implementations

You can implement pretrigger functionality through software or hardware. These two methods have different limitations and requirements. Most Measurement Computing products with pretrigger capability are implemented in hardware.

- When pretrigger functionality is implemented in hardware, the buffer created using [cbWinBufAlloc\(\)](#) must be large enough to hold 512 samples more than the requested TotalCount. The trigger location is tracked by a counter on the device. When the trigger condition is met, data is acquired and the library functions return the actual number of pretrigger points that were acquired. When run in BACKGROUND mode, the [cbGetStatus\(\)](#) function typically shows CurCount rise to the value of PretrigCount and remain there while CurIndex cycles from 0 to TotalCount continuously until the trigger is received.
- When pretrigger functionality is implemented in software, the additional space in the buffer is not required. The trigger location is tracked by software. Any triggers that occur before the number of samples defined by the pretrigger count argument are ignored. When run in BACKGROUND mode, the [cbGetStatus\(\)](#) function will typically show CurCount at a value of 0 and CurIndex at a value of -1 until the trigger is received. They will then rise from PretrigCount to TotalCount.

## Sampling rate using SINGLEIO

When transferring data using SINGLEIO mode, the maximum analog sampling rate is dependent on the speed of the computer in which the device is installed. In general the max rate is between 5 kHz and 50 kHz. An [OVERRUN](#) error will occur if the requested speed cannot be achieved. Data will be returned, but there may be gaps in the data. Some devices only support SINGLEIO mode, so the maximum rate attainable with these devices is system-dependent.



## CIO-DAS08 Series, PCI-DAS08, and PC104-DAS08

The CIO-DAS08 Series includes the following hardware:

- CIO-DAS08
- CIO-DAS08-AOH
- CIO-DAS08-AOL
- CIO-DAS08-AOM
- CIO-DAS08-PGH
- CIO-DAS08-PGL
- CIO-DAS08-PGM

This topic also includes the following hardware:

- PCI-DAS08
- PC104-DAS08

The CIO-DAS08 Series, PCI-DAS08, and PC104-DAS08 support the following UL and UL for .NET features.

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, EXTTRIGGER

### HighChan

7

### Rate

From 63 up to 50000 (refer to the [Introduction: Analog input boards](#) topic regarding SINGLEIO scans).

### Range

The CIO-DAS08, PCI-DAS08, and PC104-DAS08 boards do not have programmable gain, so the Range argument to analog input functions is ignored. However, the following are valid ranges for switch settings on these boards, and can also be used with [cbToEngUnits\(\)/ToEngUnits\(\)](#).

PCI-DAS08:

BIP5VOLTS ( $\pm 5$  volts)

CIO-DAS08 and PC104-DAS08:

BIP10VOLTS ( $\pm 10$  volts)

UNI10VOLTS (0 to 10 volts)

BIP5VOLTS ( $\pm 5$  volts)

The CIO-DAS08-PGH and CIO-DAS08-AOH support the following programmable ranges:

BIP10VOLTS ( $\pm 10$  volts)

UNI10VOLTS (0 to 10 volts)

BIP5VOLTS ( $\pm 5$  volts)

UNI1VOLTS (0 to 1 volts)

BIP1VOLTS ( $\pm 1$  volts)

UNIP1VOLTS (0 to 0.1 volts)

BIPPT5VOLTS ( $\pm 0.5$  volts)

UNIP01VOLTS (0 to 0.01 volts)

BIPPT1VOLTS ( $\pm 0.1$  volts)

BIP1PT05VOLTS ( $\pm 0.05$  volts)

BIP1PT01VOLTS ( $\pm 0.01$  volts)

BIP1PT005VOLTS ( $\pm 0.005$  volts)

The CIO-DAS08-PGL and CIO-DAS08-AOL support the following programmable ranges:

BIP10VOLTS ( $\pm 10$  volts)

UNI10VOLTS (0 to 10 volts)

BIP5VOLTS ( $\pm 5$  volts)

UNI5VOLTS (0 to 5 volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

UNI2PT5VOLTS (0 to 2.5 volts)

BIP1PT25VOLTS ( $\pm 1.25$  volts)

UNI1PT25VOLTS (0 to 1.25 volts)

BIPPT625VOLTS ( $\pm 0.625$  volts)

The CIO-DAS08-PGM and CIO-DAS08-AOM support the following programmable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI1VOLTS (0 to 1 volts)
BIPPT5VOLTS ( $\pm 0.5$ volts)	UNIPT1VOLTS (0 to 0.1 volts)
BIPPT1VOLTS ( $\pm 0.1$ volts)	UNIPT01VOLTS (0 to 0.01 volts)
BIPPT05VOLTS ( $\pm 0.05$ volts)	

## Analog output (CIO-DAS08-AOH, CIO-DAS08-AOL, and CIO-DAS08-AOM only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS

### HighChan

1 max

### Rate

Ignored

### Count

2 max

### Range

Ignored - not programmable; fixed at one of eight switch-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT67VOLTS ( $\pm 1.67$ volts)	UNI1PT67VOLTS (0 to 1.67 volts)

### DataValue

0 to 4,095

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

CIO-DAS08 and CIO-DAS08-AOH, -AOL, and -AOM also support:

UL: [cbDConfigPort\(\)](#) for FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

UL for .NET: [DConfigPort\(\)](#) for FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

### PortNum

AUXPORT

CIO-DAS08 and CIO-DAS08-AOH, -AOL, and -AOM also support these digital ports:

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

### DataValue

0 to 15 using [cbDOut\(\)](#) or [DOut\(\)](#)

0 to 7 using [cbDIn\(\)](#) or [DIn\(\)](#)

CIO-DAS08 and CIO-DAS08-AOH, -AOL, and -AOM also support these values:

0 to 255 using FIRSTPORTA or FIRSTPORTB

0 to 15 using FIRSTPORTCL or FIRSTPORTCH

### BitNum

0 to 3 using [cbDBitOut\(\)](#) or [DBitOut\(\)](#)

0 to 2 using [cbDBitIn\(\)](#) or [DBitIn\(\)](#)

CIO-DAS08 and CIO-DAS08-AOH, -AOL, and -AOM also support these values:

0 to 23 using FIRSTPORTA

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 3

### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Hardware considerations

### Pacing analog input

Hardware pacing, external or internal clock supported.

Before using the [cbAInScan\(\)](#) function or the [AInScan\(\)](#) method for timed analog input with a CIO-DAS08 or PC104-DAS08 series board, the output of counter 1 must be wired to the Interrupt input.

If you have a CIO-DAS08 board revision 3 or higher, a jumper is provided on the board to accomplish this. An interrupt level must have been selected in InstaCal and the CB.CFG file saved.

### Triggering and gating

Polled digital input triggering (TTL) supported. Refer to "[Trigger support](#)" in the "Introduction: Analog input Boards" section for more information.

Use pin 25 as the trigger input.

### Pacing analog output

Software pacing only

### Digital output

Since the channel settings and DOut bits share a register, attempting to change the digital output value during an analog input scan may result in no change or unexpected values in digital output ports.

## CIO-DAS08/JR Series

The CIO-DAS08/JR Series includes the following hardware:

- CIO-DAS08/JR
- CIO-DAS08/JR-AO
- CIO-DAS08JR/16
- CIO-DAS08/JR/16-AO

The CIO-DAS08/JR Series support the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

CONVERTDATA

### HighChan

0 to 7

### Rate

Ignored

### Range

These boards do not have programmable gain, so the Range arguments for analog input functions are ignored.

## Analog Output

Valid if the optional D/A converters are installed on the board.

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS

### HighChan

1 max

### Rate

Ignored

### Count

2 max

### Range

Ignored - not programmable; fixed at BIP5VOLTS ( $\pm 5$  volts)

### DataValue

0 to 4,095

For the **CIO-DAS08/JR/16-AO**, the following argument values are also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortNum

AUXPORT\*

**DataValue**

0 to 255

**BitNum**

0 to 7

\*AUXPORT is not configurable for these boards.

**Counter I/O****Functions**

These boards do not have any counters and do not support any of the counter I/O functions.

**Hardware Considerations****Pacing analog input**

Software pacing only

## CIO-DAS1400 Series and CIO-DAS1600 Series

The CIO-DAS1400 Series includes the following hardware:

- CIO-DAS1401/12
- CIO-DAS1402/12
- CIO-DAS1402/16

The CIO-DAS1600 Series includes the following hardware:

- CIO-DAS1601/12
- CIO-DAS1602/12
- CIO-DAS1602/16

The CIO-DAS1400 Series and CIO-DAS1600 Series support the following UL and UL for .NET features.

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, DMAIO, BURSTMODE, EXTTRIGGER

For the CIO-DAS1601/12, CIO-DAS1602/12, and CIO-DAS1602/16, these argument values are also valid:

DTCONNECT, EXTMEMORY

### HighChan

0 to 15 single-ended mode

0 to 7 differential mode

### Rate

CIO-DAS1401/12, CIO-DAS1402/12, CIO-DAS1601/12, and CIO-DAS1602/16:

160000

CIO-DAS1402/16 and CIO-DAS1602/16:

100000

CIO-DAS1401/12, CIO-DAS1402/12, CIO-DAS1601/12, and CIO-DAS1602/12 to external memory (DT Connect):

330000

### Range

CIO-DAS1402/12, CIO-DAS1402/16, CIO-DAS1602/12 and CIO-DAS1602/16:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)

CIO-DAS1401/12 and CIO-DAS1601/12:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP1VOLTS ( $\pm 1$ volts)	UNI1VOLTS (0 to 1 volts)
BIPPT1VOLTS ( $\pm 0.1$ volts)	UNIPT1VOLTS (0 to 0.1 volts)
BIPPT01VOLTS ( $\pm 0.01$ volts)	UNIPT01VOLTS (0 to 0.01 volts)

## Analog output (CIO-DAS1600 series only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS

## HighChan

1 max

## Count

2 max

## Rate

Ignored

## Pacing

Software pacing only

## Range

Ignored - not programmable; fixed at one of four jumper-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)

## DataValue

0 to 4,095

## Digital I/O

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

The CIO-DAS1600 series also supports [cbDConfigPort\(\)](#) and [DConfigPort\(\)](#)

### PortNum

AUXPORT\*

The CIO-DAS1600 series also supports these digital ports:

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

\* AUXPORT is not configurable for these boards.

### DataValue

0 to 15

The **CIO-DAS1600** series also supports these values.

0 to 255 using FIRSTPORTA or FIRSTPORTB

0 to 15 using FIRSTPORTCL or FIRSTPORTCH

### BitNum

0 to 3

The **CIO-DAS1600** series also supports these bit number settings.

0 to 23 using FIRSTPORTA

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 3

### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Hardware considerations

### Pacing analog input

Hardware pacing, external or internal clock supported. Specifying SINGLEIO while also specifying BURSTMODE is not recommended. If this combination is used, the Count value should be set as low as possible, preferably to the number of channels in the scan. Otherwise, overruns may occur.

When EXTMEMORY is used with the CIO-DAS1600 Series, the [cbGetStatus\(\)](#) function or [GetStatus\(\)](#) method does not return the current count and current index. This is a limitation imposed by maintaining identical registers to the KM-DAS1600.

### Triggering and gating

External digital (TTL) polled gate trigger supported. Refer to the "[Trigger support](#)" section of the "Introduction: Analog input boards" topic.

### Range

The CIO-DAS1400 Series and CIO-DAS1600 Series board's A/D ranges are configured with a combination of a switch (Unipolar/Bipolar) and a programmable gain code. The state of this switch is set in the configuration file using InstaCal. After the UNI/BIP switch setting is selected, only matching ranges can be used in Universal Library programs.



## CIO-DAS16 Series and PC104-DAS16 Series

The CIO-DAS16 Series and PC104-DAS16 Series includes the following hardware:

- CIO-DAS16/330
- CIO-DAS16/330i
- CIO-DAS16/M1
- CIO-DAS16/M1/16
- CIO-DAS16
- CIO-DAS16/F
- CIO-DAS16/JR
- CIO-DAS16/JR/16

The CIO-DAS16 Series and PC104-DAS16 Series includes the following hardware:

- PC104-DAS16JR/16
- PC104-DAS16JR/12

The CIO-DAS16 Series and PC104-DAS16 Series support the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

The DAS16/330, DAS16/330i, DAS16/M1 and DAS16/M1/16 also support the following functions:

UL: [cbAPretrig\(\)](#), [cbFilePretrig\(\)](#)

UL for .NET: [APretrig\(\)](#), [FilePretrig\(\)](#)

The DAS16/330i and DAS16/M1 also support the following functions:

UL: [cbALoadQueue\(\)](#)

UL for .NET: [ALoadQueue\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, EXTTRIGGER

The DAS16/330, DAS16/330i, DAS16/M1 and DAS16/M1/16 also support the following settings:

DTCONNECT, BLOCKIO, EXTMEMORY

The DAS16, DAS16/F, DAS16/JR, DAS16/JR/16 and PC104-DAS16JR series also support the following settings:

SINGLEIO, DMAIO

The DAS16/M1/16 also supports the following setting:

BURSTMODE

### HighChan

0 to 7 (DAS16/M1 and DAS16/M1/16)

0 to 15 in single-ended mode, or 0 to 7 in differential mode (all others)

### Rate

DAS16/M1 and DAS16/M1/16:

Up to 1000000

CIO-DAS16JR:

Up to 130000

DAS16/330 and DAS16/330i:

Up to 330000

DAS16/F and DAS16JR/16:

Up to 100000

DAS16:

Up to 50000

PC104-DAS16JR/12:

Up to 160000

## Range

The CIO-DAS16 and CIO-DAS16/F do not have programmable gain, so the Range argument to analog input functions is ignored.

All other boards in this series support the following ranges:

BIP5VOLTS ( $\pm 5$ volts)	UNI10VOLTS (0 to 10 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
	UNI1PT25VOLTS (0 to 1.25 volts)

For all programmable gain boards in this series except the DAS16/M1/16, this argument value is also valid:

BIP10VOLTS ( $\pm 10$  volts)

For all programmable gain boards in this series except the CIO-DAS16/JR/16 and PC104-DAS16JR/16, this argument value is also valid:

BIPPT625VOLTS ( $\pm 0.625$  volts)

## Analog output(CIO-DAS16 and CIO-DAS16/F only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS

### HighChan

1 maximum

### Rate

Ignored

### Count

2 max

### Range

Ignored - not programmable; fixed at UNI5VOLTS (0 to 5 volts)

### DataValue

0 to 4,095

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

CIO-DAS16 & 16/F, CIO-DAS16/M1 and CIO-DAS16/M1/16 also support:

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

### PortNum

AUXPORT (not configurable for these boards)

CIO-DAS16, 1CIO-DAS16/F, CIO-DAS16/M1 and CIO-DAS16/M1/16 also support:

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

### DataValue

0 to 15

CIO-DAS16, 1CIO-DAS16/F, CIO-DAS16/M1 and CIO-DAS16/M1/16 also support:

0 to 255 for FIRSTPORTA and FIRSTPORTB

0 to 15 for FIRSTPORTCL & FIRSTPORTCH

#### **BitNum**

0 to 3

CIO-DAS16, 1CIO-DAS16/F, CIO-DAS16/M1 and CIO-DAS16/M1/16 also support:

0 to 23 using FIRSTPORTA

## **Counter I/O**

#### **Functions**

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

#### **CounterNum**

1 to 3

CIO-DAS16/M1/16 also supports counter 4 through counter 6, with counter 4 being the only independent user counter.

#### **Config**

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

#### **LoadValue**

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## **Triggering (CIO-DAS16/M1/16 only)**

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

#### **TrigType**

TRIGPOSEDGE, TRIGNEGEDGE, GATEHIGH, GATELOW

#### **Threshold**

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

# CIO-DAS48-PGA, CIO-DAS48-I

The CIO-DAS48-PGA and CIO-DAS48-I supports the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

### Options

CONVERTDATA

### HighChan

47 single-ended, 23 differential

### Rate

Ignored

### Range

The CIO-DAS48-PGA can be configured with a jumper for either voltage or current input. The CIO-DAS48-I is configured for current input only.

Voltage input ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volt)	UNI2PT5VOLTS (0 to 2.5 volt)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)
BIPP625VOLTS ( $\pm 0.625$ volts)	

Current input ranges:

MA4TO20 (4 to 20 mA)	MA1TO5 (1 to 5 mA)
MA2TO10 (2 to 10 mA)	MAP5TO2PT5 (0.5 to 2.5 mA)

## Notes

The CIO-DAS48/PGA and CIO-DAS48-I do not support analog output, digital I/O, or counter I/O functions.

## CIO-DAS800 Series

The CIO-DAS800 Series includes the following hardware:

- CIO-DAS800
- CIO-DAS801
- CIO-DAS802
- CIO-DAS802/16

CIO-DAS800 Series supports the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, BLOCKIO, EXTTRIGGER

### HighChan

0 to 7

### Rate

CIO-DAS802/16:

100,000

All others in the series:

50,000

### Range

CIO-DAS800:

Ignored - Not programmable

CIO-DAS801:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI1VOLTS (0 to 1 volts)
BIP1VOLTS ( $\pm 1$ volts)	UNIPT1VOLTS (0 to 0.1 volts)
BIPP5VOLTS ( $\pm 0.5$ volts)	UNIPT01VOLTS (0 to 0.01 volts)
BIPP05VOLTS ( $\pm 0.05$ volts)	
BIPP01VOLTS ( $\pm 0.01$ volts)	

CIO-DAS802:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)
BIPP625VOLTS ( $\pm 0.625$ volts)	

CIO-DAS802/16:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)

## Analog output

These boards do not have D/A converters, and do not support analog output functions.

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortNum

AUXPORT (not configurable for these boards)

### DataValue

UL: 0 to 15 using [cbDOut\(\)](#), 0 to 7 using [cbDIn\(\)](#)

UL for .NET: 0 to 15 using [DOut\(\)](#), 0 to 7 using [DIn\(\)](#)

### BitNum

0 to 3 using [cbDBitOut\(\)/DBitOut\(\)](#)

0 to 2 using [cbDBitIn\(\)/DBitIn\(\)](#)

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 3

### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Hardware considerations

### Pacing analog input

Hardware pacing, external or internal clock supported.

The packet size is 128 samples.

Note that digital output is not compatible with concurrent [cbAInScan\(\)/AInScan\(\)](#) operation, since the channel multiplexor control shares the register with the digital output control. Writing to this register during a scan may adversely affect the scan.

### Triggering and gating

Digital hardware triggering supported.

## DEMO-BOARD

The DEMO-BOARD is a software simulation of a data acquisition device that simulates analog input and digital I/O operations. The DEMO-BOARD supports the following UL and UL for .NET features.

### Analog input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, SINGLEIO, DMAIO

#### HighChan

7 max

#### Rate

300000

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDInScan\(\)](#), [cbDOut\(\)](#), [cbDBitOut\(\)](#), [cbDOutScan\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DIn\(\)](#), [DBitIn\(\)](#), [DInScan\(\)](#), [DOut\(\)](#), [DBitOut\(\)](#), [DOutScan\(\)](#), [DConfigPort\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB

AUXPORT

### DataValue

0 to 255 using FIRSTPORTA, FIRSTPORTB, or AUXPORT

### BitNum

0 to 15 using FIRSTPORTA

0 to 7 using AUXPORT

## Using the DEMO-BOARD

### Analog input

The DEMO-BOARD simulates eight channels of 16-bit analog input. InstaCal is used to configure the following waveforms on the analog input channels:

- Sine wave
- Square wave
- Saw-tooth
- Ramp
- Damped sine wave
- Input from a data file

The data file is a streamer file, so any data that has been previously saved in a streamer file can be used as a source of demo data by the board. Data files are named **DEMO0.DAT** through **DEMO7.DAT**. When a data file is assigned to a channel, the library tries to extract data for that channel from the streamer file. If data for that channel does not exist, then the first (and possibly only) channel data in the streamer is extracted and used.

For example, DEMO2.DAT is assigned as the data source for channel 5 on the DEMO-BOARD. The library will try to extract data from the file that corresponds to channel 5. If DEMO2.DAT has scan data that corresponds to channels 0 through 15, then channel 5 data is extracted. If DEMO2.DAT only has data for a single channel, the data for that channel is used as the data source for channel 5.

### Digital I/O

The DEMO-BOARD simulates the following:

- One eight-bit AUXPORT configurable digital input/output port. Each bit of the AUXPORT generates a square wave with a different period.
- Two eight-bit configurable digital I/O ports — FIRSTPORTA and FIRSTPORTB — which can be used for high speed scanning. FIRSTPORTA functions like AUXPORT in that it generates square waves. Each bit of FIRSTPORTB generates a pulse with a different frequency.



## miniLAB 1008

The miniLAB 1008 supports the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)\\*](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)\\*](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

\*The channel-gain queues are limited to eight channel-gain pairs.

### Options

BACKGROUND, BURSTIO\*\*, BLOCKIO\*\*\*, CONTINUOUS, EXTTRIGGER, CONVERTDATA, NOCALIBRATEDATA

\*\* BURSTIO cannot be used with the CONTINUOUS option. BURSTIO can only be used with sample count scans of 4096 or less.

\*\*\* The BLOCKIO packet size is 64 samples wide.

### HighChan

0 to 7 in single-ended mode, 0 to 3 in differential mode

### Rate

8000 hertz (Hz) maximum for BURSTIO mode. The rate is 1200 Hz maximum for all other modes.

When using [cbAInScan\(\)](#) or [AInScan\(\)](#), the minimum sample rate is 100 S/s aggregate.

### Range

Single-ended:

BIP10VOLTS ( $\pm 10$  volts)

Differential:

BIP20VOLTS ( $\pm 20$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

BIP10VOLTS ( $\pm 10$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP1PT25VOLTS ( $\pm 1.25$  volts)

BIP4VOLTS ( $\pm 4$  volts)

BIP1VOLTS ( $\pm 1$  volts)

### Pacing

Hardware pacing, internal clock supported.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGHIGH, TRIGLOW

Digital (TTL) hardware triggering supported. The hardware trigger is source selectable via InstaCal (AUXPORT inputs 0 - 3).

## Analog Output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Count

(HighChan-LowChan)+1

### HighChan

1

### Range

Ignored - not programmable; fixed at UNI5VOLTS (0 to 5 V)

### DataValue

0 to 1023

## Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### Configuration Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

AUXPORT\*

### PortType

AUXPORT\*

\* Only AUXPORT is bitwise configurable on this board, and must be configured using [cbDConfigBit\(\)](#) or [cbDConfigPort\(\)](#) (or the UL for .NET functions [DConfigBit\(\)](#) or [DConfigPort\(\)](#)) before use for output.

### Port I/O Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

AUXPORT

### DataValue

0 to 15 for AUXPORT, FIRSTPORTCL, or FIRSTPORTCH

0 to 255 for FIRSTPORTA or FIRSTPORTB

### Bit I/O Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortType

AUXPORT

FIRSTPORTA

### BitNum

0 to 3 on AUXPORT

0 to 23 on FIRSTPORTA

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

### Count

$2^{32}-1$  when reading the counter.

0 when loading the counter

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are used only to reset the counter for this board to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* topic apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

### RegNum

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### EventType

ON\_SCAN\_ERROR (analog input), ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a Measurement Computing USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### Resolution

When configured for single-ended mode, the resolution of the data is 11 bits (data values between 0 and 2,047). However, the Universal Library maps this data to 12-bit values, so the range of data is no different from the differential configuration. Consequently, the data returned contains only even numbers between 0 and 4,094 when the NOCALIBRATEDATA option is used.

### BURSTIO

Allows higher sampling rates (up to 8000 Hz) for sample counts up to 4096. Data is collected into the miniLAB's local FIFO. Data is collected into the USB device's local FIFO. Data transfers to the PC don't occur until the scan completes. For BACKGROUND scans, the Count and Index returned by [cbGetStatus\(\)](#) and [GetStatus\(\)](#) remain 0, and STATUS=RUNNING until the scan finishes. The Count and Index are not updated until the scan is completed. When the scan is complete and the data is retrieved, [cbGetStatus\(\)](#) and [GetStatus\(\)](#) are updated to the current Count and Index, and Count = IDLE.

BURSTIO is the default mode for non-CONTINUOUS fast scans (aggregate sample rates above 1000 Hz) with sample counts up to 4096. BURSTIO mode allows higher sampling rates (up to 8000 Hz) for sample counts up to 4096. Non-BURSTIO scans are limited to a maximum of 1200 Hz. To avoid the BURSTIO default, specify BLOCKIO mode.

### Continuous scans

When running [cbAInScan\(\)](#) with the CONTINUOUS option, you should consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, make the total number of samples an integer multiple of the packet size and the number of channels.

### Concurrent operations

Concurrent operations on a particular USB device are not allowed. If you invoke a UL or UL for .NET function on a USB device while another function is running on that USB device, the [ALREADYACTIVE](#) error is returned.

## PCI-2500 Series

The PCI-2500 Series includes the following hardware:

- PCI-2511
- PCI-2513
- PCI-2515
- PCI-2517

The PCI-2500 Series support the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbFileAInScan\(\)](#), [cbAPretrig\(\)\\*](#), [cbATrig\(\)](#), [cbALoadQueue\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [FileAInScan\(\)](#), [APretrig\(\)\\*](#), [ATrig\(\)](#), [ALoadQueue\(\)](#)

\* Pretrigger capability is implemented in software. PretrigCount must be less than the TotalCount and cannot exceed 100000 samples. TotalCount must be greater than the PretrigCount. If a trigger occurs while the number of collected samples is less than the PretrigCount, that trigger will be ignored. Requires a call to [cbSetTrigger/SetTrigger](#) for the analog trigger type.

### Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK, EXTTRIGGER, HIGHRESRATE

With EXTTRIGGER mode, the first channel in the scan is the analog trigger channel.

### HighChan

PCI-2517, PCI-2515, PCI-2513:

0 to 15 in single-ended mode, 0 to 7 in differential mode

PCI-2511:

0 to 15 in single-ended mode

### Rate

Up to 1 MHz

### Range

PCI-2517, PCI-2515, PCI-2513:

BIP10VOLTS ( $\pm 10$ volts)	BIPPT5VOLTS ( $\pm 0.5$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIPPT2VOLTS ( $\pm 0.2$ volts)
BIP2VOLTS ( $\pm 2$ volts)	BIPPT1VOLTS ( $\pm 0.1$ volts)
BIP1VOLTS ( $\pm 1$ volt)	

PCI-2511:

BIP10VOLTS ( $\pm 10$  volts)

## Analog Output (PCI-2517 and PCI-2515 only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, NONSTREAMEDIO, SIMULTANEOUS

NONSTREAMEDIO can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

### HighChan

PCI-2517: 0 to 3

PCI-2515: 0 to 1

### Rate

1 MHz

### Range

Ignored - not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Pacing

Hardware pacing, external or internal clock supported.

## Digital I/O

### Configuration

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

PortType

FIRSTPORTA

### Port I/O

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDInScan\(\)](#), [cbDOutScan\(\)](#)\*

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DInScan\(\)](#), [DOutScan\(\)](#)\*

\*FIRSTPORTA and FIRSTPORTB must be set for output to use this function. Refer to [DIO PortNum](#) in the *Hardware Considerations* section below for more information.

### Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER, HIGHRESRATE, NONSTREAMEDIO, WORDXFER,

The EXTTRIGGER option can only be used with the [cbDInScan\(\)](#) function. You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).

The WORDXFER option can only be used with FIRSTPORTA.

The NONSTREAMEDIO, ADCCLOCKTRIG, and ADCCLOCK options can only be used with the [cbDOutScan\(\)](#) function.

The NONSTREAMEDIO option can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

Rate

12 MHz

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

DataValue

0 to 255

0 to 65,535 using the WORDXFER option with FIRSTPORTA

### Bit I/O

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 23

## Counter Input

### Functions

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCConfigScan\(\)](#), [cbCInScan\(\)](#), [cbCClear\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CConfigScan\(\)](#), [CInScan\(\)](#), [CClear\(\)](#)

Note: Counters on these boards are zero-based (the first counter number is "0").

### Rate

6 MHz

## CounterNum

0 to 3

## Options

BACKGROUND, CONTINUOUS, EXTTRIGGER

You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).

## LoadValue

0 to 65,535 (Refer to "[16-bit values using a signed integer data type](#)" for information on 16-bit values using unsigned integers.)

## Timer Output

### Functions

UL: [cbTimerOutStart\(\)](#), [cbTimerOutStop\(\)](#)

UL for .NET: [TimerOutStart\(\)](#), [TimerOutStop\(\)](#)

## TimerNum

0 to 1

## Frequency

15.260 Hz to 1.0 MHz

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

## TrigType

TRIGABOVE, TRIGBELOW, TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE

Digital triggering (TRIG\_HIGH, TRIG\_LOW, TRIGPOSEDGE, TRIGNEGEDGE) is not supported for pre-trigger acquisitions ([cbAPretrig\(\)](#) function).

Analog triggering (TRIGABOVE, TRIGBELOW) is not supported for the [cbDInScan\(\)](#) function and the [cbCInScan\(\)](#) function.

## Threshold

Analog hardware triggering, 12-bit resolution: 0 to 4,095 (supported for [cbAInScan\(\)](#) only)

Analog software triggering, 16-bit resolution: 0 to 65,535 (supported for [cbAPretrig\(\)](#) only)

## DAQ Input

### Functions

UL: [cbDagInScan\(\)](#)

UL for .NET: [DagInScan\(\)](#)

## Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK, EXTTRIGGER, HIGHRESRATE

## ChanTypeArray

ANALOG, DIGITAL8, DIGITAL16, CTR16, CTR32LOW, CTR32HIGH, SETPOINTSTATUS

## ChanArray

ANALOG:

- PCI-2517, PCI-2515, PCI-2513: 0 to 15 in single-ended mode, 0 to 7 in differential mode
- PCI-2511: 0 to 15 in single-ended mode

DIGITAL8: FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

DIGITAL16: FIRSTPORTA

CTR16: 0-3 counters

CTR32LOW: 0-3 counters

CTR32HIGH: 0-3 counters

SETPOINTSTATUS: 16-bit port that indicates the current state of the 16 possible setpoints.

ChanTypeArray flag value:

- SETPOINT\_ENABLE: Enables a setpoint. Refer to [Setpoints](#) in the *Hardware Considerations* section below for more information.

## Rate

Analog: Up to 1 MHz.

Digital: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

Counter: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

## GainArray

ANALOG only; ignore for other ChanTypeArray values.

PCI-2517, PCI-2515, PCI-2513:

BIP10VOLTS ( $\pm 10$ volts)	BIPPT5VOLTS ( $\pm 0.5$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIPPT2VOLTS ( $\pm 0.2$ volts)
BIP2VOLTS ( $\pm 2$ volts)	BIPPT1VOLTS ( $\pm 0.1$ volts)
BIP1VOLTS ( $\pm 1$ volt)	

PCI-2511:

Ignored; fixed at BIP10VOLTS ( $\pm 10$  volts)

## ChanCount

Number of elements in ChanArray, ChanTypeArray and GainArray. Up to 512 elements max.

## PretrigCount

100000 max. This argument is ignored if the EXTTRIGGER option is not specified.

# DAQ Triggering

## Functions

UL: [cbDagSetTrigger\(\)](#)

UL for .NET: [DagSetTrigger\(\)](#)

## TrigSource

TRIG\_IMMEDIATE, TRIG\_EXTTTL, TRIG\_ANALOGHW, TRIG\_ANALOGSW, TRIG\_DIGPATTERN, TRIG\_COUNTER, TRIG\_SCANCOUNT

## TrigSense

RISING\_EDGE, FALLING\_EDGE, ABOVE\_LEVEL, BELOW\_LEVEL, EQ\_LEVEL, NE\_LEVEL

## TrigEvent

START\_EVENT, STOP\_EVENT

# DAQ Setpoint

## Functions

UL: [cbDagSetSetpoints\(\)](#)

UL for .NET: [DagSetSetpoints\(\)](#)

## SetpointFlagsArray

SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA, SF\_GREATERTHAN\_LIMITB, SF\_OUTSIDE\_LIMITS, SF\_HYSTERESIS, SF\_UPDATEON\_TRUEONLY, SF\_UPDATEON\_TRUEANDFALSE

## SetpointOutputArray

SO\_NONE, SO\_FIRSTPORTC, SO\_TMR0, SO\_TMR1

Also available for PCI-2515 and PCI-2517:

SO\_DAC0, SO\_DAC1

Also available for PCI-2517:

SO\_DAC2, SO\_DAC3

## LimitAArray

Any value valid for the associated input channel.

Ignored for SF\_GREATERTHAN\_LIMITB

## LimitBArray

Any value valid for the associated input channel and less than LimitA.

Ignored for SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA

### Output#Array

For SetpointOutputArray = SO\_NONE: Ignored

For SetpointOutputArray = SO\_FIRSTPORTC: 0 to 65,535

For SetpointOutputArray = SO\_TMR#: 0 (to disable the timer) or 15.26 to 1000000 (to set the output frequency)

For SetpointOutputArray = SO\_DAC#: Voltage values between -10 and +10

### OutputMask#Array >

For SetpointOutputArray = SO\_FIRSTPORTC: 0 to 65,535

For SetpointOutputArray = all other values: Ignored

### SetpointCount

0 (to disable setpoints) to 16

## DAQ Output (PCI-2517 and PCI-2515 only)

### Functions

UL: [cbDagOutScan\(\)](#)

UL for .NET: [DagOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS, NONSTREAMEDIO, ADCCLOCKTRIG, ADCCLOCK

### ChanType

ANALOG, DIGITAL16

### ChanArray

ANALOG:

PCI-2517: 0 to 3

PCI-2515: 0 to 1

DIGITAL16:

FIRSTPORTA (FIRSTPORTB must be configured as an output)

### Rate

ANALOG: Up to 1 MHz.

DIGITAL16: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

### Range

Ignored

## Hardware considerations

### Channel count

For input and output scans, the count must be set to an integer multiple of the number of channels or a [BADCOUNT](#) error is returned.

### Sampling and update rates

Sampling and update rates are system-dependent. Data overruns/underruns may occur with higher sampling rates when using BACKGROUND and CONTINUOUS modes. To avoid this, use a larger buffer/count size, or use NONSTREAMEDIO mode, if supported.

The minimum size buffer is 256 for cbAOutScan(), cbDInScan(), and cbDOutScan(). Values less than that result in a [BADBUFFERSIZE](#) error.

### Settling time

For most applications, settling time should be left at the default value of 1  $\mu$ s. However, if you are scanning multiple channels and one or more channels are connected to a high impedance source, you may get better results by increasing the settling time. Keep in mind that increasing the settling time reduces the maximum acquisition rate. You can set the time between A/D conversions with the ADC Settling Time option in InstaCal. Select between 1  $\mu$ s, 5  $\mu$ s, 10  $\mu$ s, or 1 ms.

### Setpoints

You enable setpoints with the SETPOINT\_ENABLE flag. This flag must be OR'ed with the ChanTypeArray argument values. You set the setpoint criteria with the [cbDagSetSetpoints\(\)/DagSetSetpoints\(\)](#). The number of channels set with the SETPOINT\_ENABLE flag must match the number of setpoints set by the SetpointCount argument ([cbDagSetSetpoints\(\)/DagSetSetpoints\(\)](#)).



## Output non-streamed data to a DAC output channel

With NONSTREAMEDIO mode, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. The FIFO holds 524288 samples. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample data are transferred or downloaded to the device, the device is responsible for outputting the data. You can't make any changes to the output buffer once the output begins.

## Trigger DAC output operations with the ADC clock

Specify the ADCCLOCKTRIG option to trigger a data output operation upon the start of the ADC clock.

## DIO PortNum

For `cbDOutScan()/DOutScan()` and `cbDaqOutScan()/DaqOutScan()`, `FIRSTPORTA` and `FIRSTPORTB` are treated as one 16-bit port. These functions can only be used with `FIRSTPORTA`. You must configure both `FIRSTPORTA` and `FIRSTPORTB` for output using the `cbDConfigPort()` function.

## Synchronous scanning with multiple boards

You can operate up to four PCI-2500 Series boards synchronously by setting the direction of the A/D and D/A pacer pins (**XAPCR** or **XDPCR**) in InstaCal.

On the board used to pace each device, set the pacer pin that you want to use (XAPCR or XDPCR) for *Output*. On the board(s) that you want to synchronize with this board, set the pacer pin that you want to use (XAPCR or XDPCR) for *Input*.

You set the direction using the InstaCal configuration dialog's **XAPCR Pin Direction** and **XDPCR Pin Direction** settings. If you have an older version of InstaCal, these settings might be labeled "ADC Clock Output" (set to Enabled to configure XAPCR for *output*) or "DAC Clock Output" (set to Enabled to configure XDPCR for *output*).

Wire the pacer pin configured for output to each of the pacer input pins that you want to synchronize.

## Quadrature encoder operations

To configure a counter channel as a multi-axis quadrature encoder, use the [cbCConfigScan\(\)/CConfigScan\(\)](#) Mode argument values to set a specified counter to encoder mode, set the encoder measurement mode to X1, X2, or X4, and then set the count to be latched either by the internal "start of scan" signal (default) or by the signal on the mapped channel.

You can optionally perform the following operations:

- Enable gating, so that the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.
- Enable "latch on Z" to latch counter outputs using the Encoder Z mapped signal.
- Enable "clear on Z" so that the counter is cleared on the rising edge of the mapped (Z) channel. By default, "clear on Z" is disabled, and the counter is not cleared.

## Asynchronous reads

The `CConfigScan()` method's Bit32 counter mode option only affects counter resolution for asynchronous calls ([CIn\(\)](#) and [CIn32\(\)](#)), and only when the counter is configured for `StopAtMax`.

This mode is recommended for use only with `CIn32()`. Using the Bit32 option with `CIn()` is not very useful, since the value returned by `CIn()` is only 16 bits. The effect is that the value returned by `CIn()` rolls over 65,535 times before stopping.

## PCI-DAS1602, PCI-DAS1200 and PCI-DAS1000 Series

The PCI-DAS1602, PCI-DAS1200 and PCI-DAS1000 Series includes the following hardware:

- PCI-DAS1000, PCI-DAS1001, PCI-DAS1002
- PCI-DAS1200, PCI-DAS1200/JR
- PCI-DAS1602/12, PCI-DAS1602/16

The PCI-DAS1602, PCI-DAS1200 and PCI-DAS1000 Series support the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbAPretrig\(\)](#), [cbFileAInScan\(\)](#), [cbFilePretrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [APretrig\(\)](#), [FileAInScan\(\)](#), [FilePretrig\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, BLOCKIO, BURSTMODE, EXTTRIGGER

### HighChan

0 to 15 in single-ended mode, 0 to 7 in differential mode

### Rate

PCI-DAS1602/12, PCI-DAS1200, PCI-DAS1200/JR: Up to 330000

PCI-DAS1000: Up to 250000

PCI-DAS1602/16, PCI-DAS1002: Up to 200000

PCI-DAS1001: Up to 150000

### Range

PCI-DAS1602/12, PCI-DAS1602/16, PCI-DAS1200, PCI-DAS1200JR, PCI-DAS1002, PCI-DAS1000:

indent2

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)

PCI-DAS1001:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP1VOLTS ( $\pm 1$ volts)	UNI1VOLTS (0 to 1 volts)
BIPPT1VOLTS ( $\pm 0.1$ volts)	UNIPT1VOLTS (0 to 0.1 volts)
BIPPT01VOLTS ( $\pm 0.01$ volts)	UNIPT01VOLTS (0 to 0.01 volts)

## Analog Output

Excludes PCI-DAS1200/JR.

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS

For PCI-DAS1602 Series, the following argument values are also valid:

BACKGROUND, CONTINUOUS, EXTCLOCK

### HighChan

0 to 1

### Rate

PCI-DAS1602/16: Up to 100000

PCI-DAS1602/12: Up to 250000

All others: ignored

## Range

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)

## DataValue

0 to 4,095

For **PCI-DAS1602/16**, the following argument values are also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

## Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

## PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

## DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

0 to 15 for FIRSTPORTCL or FIRSTPORTCH

## BitNum

0 to 23 FIRSTPORTA

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

## CounterNum

4 to 6

## Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

## Loadvalue

0 to 65,535

## Triggering

PCI-DAS1602/12 and PCI-DAS1602/16 only

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

## TrigType

TRIGPOSEDGE, TRIGNEGEDGE, TRIGABOVE, TRIGBELOW, GATEHIGH, GATELOW, GATENEGHYS, GATEPOSHYS, GATEABOVE, GATEBELOW, GATEINWINDOW, GATEOUTWINDOW

## Threshold

PCI-DAS1602/16: 0 to 65,535

PCI-DAS1602/12: 0 to 4,095

## Event Notification (PCI versions only)

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

## EventType

ON\_SCAN\_ERROR, ON\_PRETRIGGER, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

PCI-DAS1602/12 and PCI-DAS1602/16 also support: ON\_END\_OF\_AO\_SCAN

## Hardware considerations

### Pacing analog input

Hardware pacing, external or internal clock supported.

The clock edge used to trigger acquisition for the external pacer may be rising or falling and is selectable using InstaCal.

For the PCI-DAS1602/16, the packet size is 256 samples. All others in this series have a packet size of 512 samples.

### Analog input configuration:

The analog input mode is selectable via InstaCal for either 8 channel differential or 16 channel single-ended.

### Triggering and gating - PCI-DAS1602 Series

Digital (TTL) and analog hardware triggering supported.

Analog thresholds are set relative to the  $\pm 10V$  range. For example: a threshold of 0 equates to -10 V. Thresholds of 65,535 and 4,095 correspond to +9.999695 and +9.995116 volts for the 16-bit and 12-bit boards, respectively.

When using analog trigger feature, one or both of the DACs are used to set the threshold and are unavailable for other functions. If the trigger function requires a single reference (GATEABOVE, GATEBELOW, TRIGABOVE, TRIGBELOW) then DAC0 is available. If the trigger function requires two references (GATEINWINDOW, GATE OUTWINDOW, GATENEGHYS, GATEPOSHYS), then neither DAC is available for other functions.

### Triggering and gating - PCI-DAS1200, PCI-DAS1000 Series

Digital (TTL) hardware triggering supported.

### Concurrent operations - PCI-DAS1602 Series

Concurrent analog input and output scans supported. That is, PCI-DAS1602 Series boards allow for operations of analog input functions ([cbAInScan\(\)](#) and [cbAPretrig\(\)](#), or [AInScan\(\)](#) and [APretrig\(\)](#)), and analog output functions ([cbAOutScan\(\)](#) or [AOutScan\(\)](#)) to overlap without having to call [cbStopBackground](#) or [StopBackground\(\)](#) between the start of input and output scans.

### Pacing analog output - PCI-DAS1602 Series

Hardware pacing, external or internal clock supported.

The clock edge used to trigger analog output updates for the external pacer may be rising or falling and is selectable using InstaCal.

### Counters

The source for counter 4 may be internal or external and is selectable using InstaCal.

Although counters 4, 5 and 6 are programmable through the counter functions, the primary purpose for some of these counters may conflict with these functions.

Potential conflicts:

- PCI-DAS1200, PCI-DAS1000 Series: Counters 5 and 6 are always available to the user. Counter 4 is used as a residual counter by some of the analog input functions and methods.
- PCI-DAS1602 Series: Counters 5 and 6 are used as DAC pacers by some analog output functions and methods. Counter 4 is used as a residual counter by some of the analog input functions and methods.

## PCI-DAS4020/12

### Analog Input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbAPretrig\(\)](#), [cbFileAInScan\(\)](#), [cbFilePretrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [APretrig\(\)](#), [FileAInScan\(\)](#), [FilePretrig\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, DMAIO, BLOCKIO\*, EXTTRIGGER

\*The packet size is based on the Options setting:

Options setting	Packet size
BLOCKIO	2,048

#### HighChan

3 max. When scanning multiple channels, the number of channels scanned must be even.

#### Rate

Up to 20000000. The minimum value for Rate is 1000. Note that contiguous memory is no longer required to achieve the maximum sample rate.

#### Range

BIP5VOLTS ( $\pm 5$  volts)

BIP1VOLTS ( $\pm 1$  volts)

### Analog Output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

NONE

#### HighChan

1 max

#### Count

2

#### Rate

Ignored

#### Range

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

#### DataValue

0 to 4,095

#### Pacing

Software only

### Digital I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

#### DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

0 to 15 for FIRSTPORTCL or FIRSTPORTCH

#### BitNum

0 to 23 for FIRSTPORTA

## Counter I/O

### Functions

No counter functions are supported.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE, TRIGNEGEDGE, TRIGABOVE, TRIGBELOW, GATEHIGH, GATELOW, GATENEGHYS, GATEPOSHYS, GATEABOVE, GATEBELOW, GATEINWINDOW, GATEOUTWINDOW

### Threshold

0 to 4,095

## Event Notification

### Functions

UL: [cbDisableEvent\(\)](#), [cbEnableEvent\(\)](#)

UL for .NET: [DisableEvent\(\)](#), [EnableEvent\(\)](#)

### Types

ON\_SCAN\_ERROR, ON\_PRETRIGGER\*, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

\*Note that theEventData for ON\_PRETRIGGER events may not be accurate. In general, this value will be below the actual number of pretrigger samples available in the buffer.

## Hardware Considerations

### EXTCLOCK

An approximation of the rate is used to determine the size of the packets to transfer from the board. When the EXTCLOCK option is used, set the Rate argument to an approximate maximum value.

### Pacing analog input

Hardware pacing, external or internal clock supported. The clock source can be set via InstaCal to either the "Trig/Ext Clk" BNC input or the "A/D External Clock" input on the 40 pin connector (P3). Configuring for the BNC clock input will disable the clock input (pin 10) on the 40-pin connector. When the EXTCLOCK option is used, the clock signal presented to the "Trig/Ext Clk" BNC input or the "A/D External Clock" input is divided by 2 in one or two channel mode, and is divided by 4 in four channel mode. If both EXTCLOCK and EXTTRIGGER are used, both the Trigger BNC and pin 10 on the 40-pin connector require signals. This is further explained in the Triggering section below.

When using EXTCLOCK, the Rate argument *is used* by the Universal Library to calculate the appropriate chain size; set the Rate argument to the approximate rate that the external clock will be pacing acquisitions.

When executing [cbAInScan\(\)/AInScan\(\)](#) with the EXTCLOCK option, the first three clock pulses are used to set up the PCI-DAS4020/12, and the first sample is actually taken on the fourth clock pulse.

### Triggering and gating

Digital (TTL) hardware triggering supported. The trigger source can be set via InstaCal to either the "Trig/Ext Clk" BNC input, the "A/D Start Trigger" input on the 40-pin connector (P3) or the "A/D Stop Trigger" input on the 40-pin connector (P3). Use the A/D Start Trigger input for the [cbAInScan\(\)](#) and [cbFileAInScan\(\)](#) functions, and [AInScan\(\)](#) and [FileAInScan\(\)](#) methods. For the [cbAPretrig\(\)](#) or [cbFilePretrig\(\)](#) functions, and the [APretrig\(\)](#) or [FilePretrig\(\)](#) methods, use the A/D Stop Trigger input.

When using both EXTCLOCK and EXTTRIGGER options, one of the signals (either clock or trigger) must be assigned to the Trig/Ext Clk BNC input. The function of the Trigger BNC is determined by the setting of "Trig/Ext Clock Mode" in InstaCal. The Trig/Ext Clock BNC can be set to function as either the trigger ("A/D Start Trigger") or the clock ("A/D External Clock"). Pin 10 on the 40-pin connector then assumes the opposite function.

Analog hardware triggering supported. The trigger source can be set via InstaCal to any of the analog BNC inputs. [cbSetTrigger\(\)/SetTrigger\(\)](#) is supported for TRIGBELOW and TRIGABOVE trigger types. Analog thresholds are set relative to the voltage range set in the scan. For example, using a range of BIP1VOLTS during a [cbAInScan\(\)/AInScan\(\)](#), (0) corresponds to -1V and 4,095 corresponds to +1V.

When using the [cbAPretrig\(\)](#) function or the [APretrig\(\)](#) method, use either the TRIGGER BNC or pin 8 of the 40 pin connector. To use the BNC, set the InstaCal option "Trig/Ext Clock Mode" to A/D Stop Trigger; otherwise, if not set to this selection, pin 8 of the 40-pin connector is used.

When using `cbAPretrig()/APretrig()` with EXTCLOCK, the two inputs are required. The TRIGGER BNC can be set to function as either the pacer clock or the trigger. For the BNC to be setup as the pacer clock, set the InstaCal option "Trig/Ext Clk Mode" to A/D External Clock. To use the BNC as the trigger, set this InstaCal option to A/D Stop Trigger. If neither of these selections are used, the 40-pin connector will be used for both inputs; pin 8 will be input for A/D Stop Trigger, and pin 10 will be input for the pacer clock signal.

Digital (TTL) hardware gating supported. The gate source can be set via InstaCal to either the "Trig/Ext Clk" BNC input or the "A/D Pacer Gate" input on the 40-pin connector (P3).

Analog hardware gating supported. Analog thresholds are set relative to the voltage range set in the scan. For example, using a range of BIP1VOLTS during a `cbAInScan()/AInScan()`, (0) corresponds to (-1V) and 4,095 corresponds to +1V.

The gate must be in the active (enabled) state before starting an acquisition.

For EXTCLOCK or EXTTRIGGER (digital triggering) using the BNC connector, InstaCal provides a configuration setting for thresholds. The selections available are either 0 volts (V) or 2.5 volts (V). Use 0 volts if the incoming signal is BIPOLAR. Use the 2.5 volts option if the signal is UNIPOLAR, such as standard TTL.

When using both EXTCLOCK and EXTTRIGGER options, one of the signals (either clock or trigger) must be assigned to the Trig/Ext Clk BNC input.

### Sample Size Requirements

With the following functions and methods, be aware of packet size, and adjust the number of samples acquired accordingly:

- [`cbAPretrig\(\)`](#), [`APretrig\(\)`](#)
- [`cbAInScan\(\)`](#), [`AInScan\(\)`](#) with the CONTINUOUS scan option

These functions and methods use a circular buffer. Align the data by packets in the buffer. The total number of samples must be greater than one packet, and must be an integer multiple of packet size; refer to the following table. The minimum value for contiguous memory is calculated as:

$$(\text{\# of KB}) = \{(\text{\# of samples}) \div 512\}$$

For example, to run `cbAInScan()` on one channel at 18 MHz with the CONTINUOUS option set, the minimum sample size from the table below is 262,144, since the Rate is between 14 and 20 MHz. The minimum contiguous memory is then calculated as:

$$(262,144 \div 512) = 512 \text{ KB}$$

# of channels	Rate in MHz	Packet size in samples	Minimum sample size (2 packets)
1	$20 \geq \text{Rate} \geq 13.3$	131,072	262,144
	$13.3 > \text{Rate} > 4$	65,536	131,072
	$4 \geq \text{Rate} \geq 2$	4,096	8,192
	$2 > \text{Rate}$	2,048	4,096
2	$20 \geq \text{Rate} \geq 6.6$	131,072	262,144
	$6.6 > \text{Rate} \geq 2$	65,536	131,072
	$2 > \text{Rate} \geq 1$	4,096	8,192
	$1 > \text{Rate}$	2,048	4,096
4	$10 \geq \text{Rate} \geq 3.3$	131,072	262,144
	$3.3 > \text{Rate} \geq 1$	65,536	131,072
	$1 > \text{Rate} \geq 0.5$	4,096	8,192
	$0.5 > \text{Rate}$	2,048	4,096

PCI-DAS6000 Series

The PCI-DAS6000 Series includes the following hardware:

- PCI-DAS6013, PCI-DAS6014
- PCI-DAS6023, PCI-DAS6025
- PCI-DAS6030, PCI-DAS6031, PCI-DAS6032, PCI-DAS6033, PCI-DAS6034, PCI-DAS6035, PCI-DAS6036
- PCI-DAS6040
- PCI-DAS6052
- PCI-DAS6070, PCI-DAS6071

The PCI-DAS6000 Series support the following UL and UL for .NET features.

Analog Input

Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbAPretrig\(\)](#), [cbFileAInScan\(\)](#), [cbFilePretrig\(\)](#), [cbALoadQueue\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [APretrig\(\)](#), [FileAInScan\(\)](#), [FilePretrig\(\)](#), [ALoadQueue\(\)](#)

Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, DMAIO, BLOCKIO, BURSTMODE, EXTTRIGGER

Packet size is 512 for all PCI-6000 Series in most configurations. The exceptions are shown below.

Device	Aggregate rate	Packet size
PCI-DAS6040	400 kHz to 800 kHz	1,024
PCI-DAS6070	>800 kHz	2,048
PCI-DAS6071		

HighChan

0 to 15 in single-ended mode. 0 to 7 in differential mode.

For the PCI-DAS6031, PCI-DAS6033 and PCI-DAS6071, the following additional argument values are also valid:

16 to 63 in single-ended mode, 8 to 31 in differential mode

Rate

PCI-DAS6030, PCI-DAS6031, PCI-DAS6032, PCI-DAS6033:

Up to 100,000

PCI-DAS6013, PCI-DAS6014, PCI-DAS6023, PCI-DAS6025, PCI-DAS6034, PCI-DAS6035, PCI-DAS6036:

Up to 200,000

PCI-DAS6040:

Single-channel: Up to 500,000

Multi-channel: Up to 250,000

PCI-DAS6052:

Up to 333,000

PCI-DAS6070, PCI-DAS6071:

Up to 1,250,000

Range

PCI-DAS6013\*, PCI-DAS6014\*, PCI-DAS6023, PCI-DAS6025, PCI-DAS6034\*, PCI-DAS6035\*, and PCI-DAS6036\*:

BIP10VOLTS (±10 volts)

BIP5VOLTS (±5 volts)

BIPPT5VOLTS (±0.5 volts)

BIPPT05VOLTS (±0.05 volts)

\* Note: Mixing high gains (BipPt05Volts, BipPt5Volts) with low gains (Bip5Volts, Bip10Volts) within an AInScan() function is not supported.



PCI-DAS6030, PCI-DAS6031, PCI-DAS6032 and PCI-DAS6033:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2VOLTS ( $\pm 2$ volts)	UNI2VOLTS (0 to 2 volts)
BIP1VOLTS ( $\pm 1$ volt)	UNI1VOLTS (0 to 1 volt)
BIPPT5VOLTS ( $\pm 0.5$ volts)	UNIPT5VOLTS (0 to 0.5 volts)
BIPPT2VOLTS ( $\pm 0.2$ volts)	UNIPT2VOLTS (0 to 0.2 volts)
BIPPT1VOLTS ( $\pm 0.1$ volts)	UNIPT1VOLTS (0 to 0.1 volts)

PCI-DAS6040, PCI-DAS6052, PCI-DAS6070 and PCI-DAS6071:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2VOLTS (0 to 2 volts)
BIP1VOLTS ( $\pm 1$ volt)	UNI1VOLTS (0 to 1 volt)
BIPPT5VOLTS ( $\pm 0.5$ volts)	UNIPT5VOLTS (0 to 0.5 volts)
BIPPT25VOLTS ( $\pm 0.25$ volts)	UNIPT2VOLTS (0 to 0.2 volts)
BIPPT1VOLTS ( $\pm 0.1$ volts)	UNIPT1VOLTS (0 to 0.1 volts)
BIPPT05VOLTS ( $\pm 0.05$ volts)	

## Analog Output

PCI-DAS6014, PCI-DAS6025, PCI-DAS6030, PCI-DAS6031, PCI-DAS6035, PCI-DAS6036, PCI-DAS6040, PCI-DAS6052, PCI-DAS6070 and PCI-DAS6071 only.

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS, BACKGROUND, EXTCLOCK, CONTINUOUS (packet size = 512)

### HighChan

0 to 1

### Rate

PCI-DAS6014, PCI-DAS6025, PCI-DAS6035, PCI-DAS6036:

10 kHz

PCI-DAS6030, PCI-DAS6031:

100 kHz

PCI-DAS6040:

Single-channel: 1.0 MHz

Multi-channel: 500 kHz

PCI-DAS6052:

333 kHz

PCI-DAS6070, PCI-DAS6071:

1.0 MHz

### Range

PCI-DAS6014, PCI-DAS6025, PCI-DAS6035 and PCI-DAS6036:

Ignored - Not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

PCI-DAS6030, PCI-DAS6031, PCI-DAS6040, PCI-DAS6052, PCI-DAS6070 and PCI-DAS6071:

BIP10VOLTS ( $\pm 10$  volts)

UNI10VOLTS (0 to 10 volts)

### DataValue

0 to 4,095

For the PCI-DAS6014, PCI-DAS6030, PCI-DAS6031, PCI-DAS6036 and PCI-DAS6052, the following additional argument value is valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Pacing

Hardware pacing, external or internal clock supported.

## Digital I/O

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigBit\(\)](#), [DConfigPort\(\)](#), [GetDInMask\(\)](#), [GetDOutMask\(\)](#)

### PortNum

AUXPORT\*

### DataValue


0 to 255

### BitNum

0 to 7

\*AUXPORT is bitwise configurable for these boards, and must be configured using [cbDConfigBit\(\)/DConfigBit\(\)](#) or [cbDConfigPort\(\)/DConfigPort\(\)](#) before use.

### PCI-DAS6025:

The **PCI-DAS6025** is designed with the 82C55 chip, and the following additional ports are also available on this board. Click  here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

### DataValue

0-15 for FIRSTPORTCL or FIRSTPORTCH

0-255 for FIRSTPORTA or FIRSTPORTB

### BitNum

0-23 for FIRSTPORTA

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 2

### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### RegNum

LOADREG1, LOADREG2

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE, TRIGNEGEDGE, GATEHIGH, GATELOW

For the PCI-DAS6030, PCI-DAS6031, PCI-DAS6032, PCI-DAS6033, PCI-DAS6040, PCI-DAS6052, PCI-DAS6070, and PCI-DAS6071, the following additional argument values are valid:

TRIGABOVE, TRIGBELOW, GATENEGHYS, GATEPOSHYS, GATEABOVE, GATEBELOW, GATEINWINDOW, GATEOUTWINDOW

## Threshold

PCI-DAS6040, PCI-DAS6070 and PCI-DAS6071:

0 to 255

PCI-DAS6030, PCI-DAS6031, PCI-DAS6032, PCI-DAS6033, PCI-DAS6052:

0 to 4,095

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### EventType

ON\_SCAN\_ERROR, ON\_PRETRIGGER\*, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_END\_OF\_AO\_SCAN\*\*

\*Note that the EventData for ON\_PRETRIGGER events may not be accurate. In general, this value will be below the actual number of pretrigger samples available in the buffer.

\*\*Not supported for PCI-DAS6013, PCI-DAS6023, PCI-DAS6032, PCI-DAS6033 and PCI-DAS6034.

## Hardware Considerations

### Advanced timing and control configuration

You can access the advanced features provided by the Auxiliary Input/Output and DAQ-Sync interfaces through the board configuration page of InstaCal and the UL functions [cbGetSignal\(\)](#) and [cbSelectSignal\(\)](#)\*, or the UL for .NET methods [GetSignal\(\)](#) and [SelectSignal\(\)](#)\*.

ADC\_TB\_SRC and DAC\_TB\_SRC are intended to synchronize the timebase of the analog input and output pacers across two or more boards. Internal calculations of sampling and update rates assume that the external timebase has the same frequency as its internal clock. Adjust sample rates to compensate for differences in clock frequencies.

For example, if the external timebase has a frequency of 10 MHz on a board that has an internal clock frequency of 40 MHz, the scan function samples or updates at a rate of about 1/4 the rate entered. However, while compensating for differences in the external timebase and internal clock frequency, if the rate entered results in an invalid pacer count, the function returns a [BADRATE](#) error.

\*Although the PCI-DAS6013 and PCI-DAS6014 both support cbSelectSignal/SelectSignal(), these boards do not support DAQ-Sync. Therefore:

- Using the DS\_CONNECT option with the Connection argument for the cbSelectSignal() function generates a [BADCONNECTION](#) error.
- Using the DsConnector option with the connectionPin parameter for the SelectSignal() method generates a BADCONNECTION error.

### Pacing analog input

Hardware pacing, external or internal clock supported. The clock edge is selectable through InstaCal and [cbSelectSignal\(\)/SelectSignal\(\)](#).

When using EXT\_CLOCK and BURSTMODE together, do not use the A/D External Pacer to supply the clock. Use the A/D Start Trigger input instead. Since BURSTMODE is actually paced by the internal burst clock, specifying EXT\_CLOCK when using BURSTMODE is equivalent to specifying EXT\_TRIGGER.

Except for SINGLEIO transfers, CONTINUOUS mode scans require enough memory for two packets, or 1,024 samples. The packet size is 512 samples.

### Analog input configuration

- **16-channel boards:** The analog input mode may be 8 channel differential, 16 channel single-ended referenced to ground, or 16 channel single-ended non-referenced and may be selected using InstaCal.
- **64-channel boards:** The analog input mode may be 32 channel differential, 64 channel single-ended referenced to ground, or 64 channel single-ended non-referenced and may be selected using InstaCal.

### Triggering and Gating

Digital (TTL) hardware triggering is supported for the entire series. [cbSetTrigger\(\)/SetTrigger\(\)](#) is supported for GATEHIGH, GATELOW, TRIGPOSEDGE, TRIGNEGEDGE.

The A/D PACER GATE input is used for gating with GATEHIGH or GATELOW. The A/D START TRIGGER input is used for triggering with TRIGPOSEDGE and TRIGNEGEDGE.

When using [cbAPretrig\(\)/APretrig\(\)](#) or [cbFilePretrig\(\)/FilePretrig\(\)](#), use the A/D Stop Trigger input to supply the trigger.

For the PCI-DAS6030, PCI-DAS6031, PCI-DAS6032, PCI-DAS6033, PCI-DAS6040, PCI-DAS6052, PCI-DAS6070 and PCI-DAS6071: Analog hardware triggering and gating are supported. [cbSetTrigger\(\)/SetTrigger\(\)](#) is supported for TRIGABOVE, TRIGBELOW, GATENEGHYS, GATEPOSHYS, GATEABOVE, GATEBELOW, GATEINWINDOW, GATEOUTWINDOW.

The analog trigger source may be set via InstaCal as either the ATRIG input (pin #43 on the I/O connector), or as the first channel in the scan (CH# IN). To use the ATRIG input as the trigger source, set the InstaCal Analog input Trig Source to *Analog Trigger Pin*. To use the first scanned channel as the trigger source, set InstaCal to *1st Chan in the Scan*.

**Note:** When using analog gating features, we strongly recommend setting the ATRIG input as the trigger source.

### Using the ATRIG input as the trigger input

When the trigger source is set to *Analog Trigger Pin*, analog thresholds are set relative to the  $\pm 10V$  range.

### Using the "First Channel in Scan" as the trigger input

When the trigger source is set to *1st Chan in Scan*, the range used for the thresholds is the same as the A/D channel. When using analog gating features with 1st Channel in Scan as the trigger source, be careful to only scan a single channel.

### Calculating analog trigger thresholds

Analog thresholds for the PCI-DAS6030, PCI-DAS6031, PCI-DAS6032, PCI-DAS6033, and PCI-DAS6052 are 12-bit values. For example: a threshold value of 0 equates to  $-10$  volts, while a threshold value of 4,095 equates to  $+9.9976$  volts. Analog thresholds for the PCI-DAS6040, PCI-DAS6070, and PCI-DAS6071 are 8-bit values. For example: a threshold value of 0 equates to  $-10$  volts, while a threshold value of 255 equates to  $+9.92188$  volts.

You need to manually calculate trigger threshold values for these PCI-DAS6000 Series boards. For information on calculating thresholds, refer to the *Notes* section in the [cbSetTrigger\(\)](#) or [SetTrigger\(\)](#) topics in the *Universal Library Function Reference*.

### Channel-Gain Queue

When using [cbALoadQueue\(\)/ALoadQueue\(\)](#), up to 8k elements may be loaded into the queue.

For PCI-DAS6013, PCI-DAS6014, PCI-DAS6034, PCI-DAS6035, and PCI-DAS6036: Mixing high gains (BipPt05Volts, BipPt5Volts) with low gains (Bip5Volts, Bip10Volts) within an [AInScan\(\)](#) function is not supported.

### Analog Output

Using [cbAOutScan\(\)](#) / [AOutScan\(\)](#) in CONTINUOUS mode requires a minimum sample size of two packets. A packet is 512 samples.

### Digital I/O Configuration

AUXPORT is bitwise configurable for these boards, and must be configured using [cbDConfigBit\(\)](#) or [cbDConfigPort\(\)/DConfigBit\(\)](#) or [DConfigPort\(\)](#) before use.

### Counters

The source for counters 1 and 2 may be internal 10 MHz, internal 100 kHz, or external, and is selectable using InstaCal.

## PCI-DAS64/M1/16, PCI-DAS64/M2/16

The PCI-DAS64/M1/16 and PCI-DAS64/M2/16 support the following UL and UL for .NET features.

### Analog Input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbAPretrig\(\)](#), [cbFileAInScan\(\)](#), [cbFilePretrig\(\)](#), [cbALoadQueue\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [APretrig\(\)](#), [FileAInScan\(\)](#), [FilePretrig\(\)](#), [ALoadQueue\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, DMAIO, BLOCKIO, BURSTMODE, EXTTRIGGER

#### HighChan

0 to 63 in single-ended mode; 0 to 31 in differential mode.

#### Rate

PCI-DAS64/M2/16:

Single-channel, Single-range: Up to 2000000

Multi-channel, Single-range: Up to 1500000

Channel/Gain Queue: Up to 750000

PCI-DAS64/M1/16:

Single-channel, Single-range: Up to 1000000

Multi-channel, Single-range: Up to 1000000

Channel/Gain Queue: Up to 750000

#### Range

BIP5VOLTS ( $\pm 5$  volts)

UNI5VOLTS (0 to 5 volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

UNI2PT5VOLTS (0 to 2.5 volts)

BIP1PT25VOLTS ( $\pm 1.25$  volt)

UNI1PT25VOLTS (0 to 1.25 volt)

BIPPPT625VOLTS ( $\pm 0.625$  volts)

When using [cbALoadQueue\(\)](#) or [ALoadQueue\(\)](#), up to 8k elements may be loaded into the queue.

### Analog Output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS

#### HighChan

1 max

#### Count

2

#### Rate

up to 100000

#### Range

Ignored - Not programmable; fixed at BIP5VOLTS ( $\pm 5$  volts)

#### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Digital I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

## PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCH, FIRSTPORTCL, AUXPORT\*

## DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

0 to 15 for FIRSTPORTCL or FIRSTPORTCH or AUXPORT\*

## BitNum

0 to 23 for FIRSTPORTA

0 to 3 for AUXPORT\*

\*AUXPORT is not configurable for these boards.

# Counter I/O

## Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

## CounterNum

1

## Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

## Loadvalue

0 to 65,535

Refer to "[16-bit values using a signed integer data type](#)" for information on 16-bit values using unsigned integers.

# Triggering

## Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

## TrigType

TRIGPOSEDGE, TRIGNEGEDGE, TRIGABOVE, TRIGBELOW, GATEHIGH, GATELOW, GATENEGHYS, GATEPOSHYS, GATEABOVE, GATEBELOW, GATEINWINDOW, GATEOUTWINDOW

## Threshold

0 to 65,535

Refer to "[16-bit values using a signed integer data type](#)" for information on 16-bit values using unsigned integers.

# Event Notification

## Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

## EventType

ON\_SCAN\_ERROR, ON\_PRETRIGGER\*, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_END\_OF\_AO\_SCAN

# Hardware Considerations

## Pacing analog input

- Hardware pacing, external or internal clock supported.
- The clock edge used to trigger acquisition for the external pacer may be rising or falling and is selectable using InstaCal.
- The packet size is 512 samples.

## Options

Except for SINGLEIO transfers, CONTINUOUS mode scans require enough memory for half FIFO of memory.

## Analog Input configuration

The analog input mode may be 32 channel differential or 64 channel single-ended and may be selected using InstaCal.

## Triggering and gating

Digital (TTL) hardware triggering supported. Use the A/D Start Trigger Input (pin 55) for triggering and gating with [cbAInScan\(\)](#) and

[cbFileAInScan\(\)](#) / [AInScan\(\)](#) and [FileAInScan\(\)](#). Use the A/D Stop Trigger Input (pin 54) for [cbAPretrig\(\)](#) and [cbFilePretrig\(\)](#) / [APretrig\(\)](#) and [FilePretrig\(\)](#).

Analog hardware triggering and gating are supported. [cbSetTrigger\(\)](#) / [SetTrigger\(\)](#) are supported for TRIGABOVE, TRIGBELOW, GATENEGHYS, GATEPOSHYS, GATEABOVE, GATEBELOW, GATEINWINDOW, GATEOUTWINDOW. Use the Analog Trigger Input (pin 56) for analog triggering. Analog thresholds are set relative to the  $\pm 5V$  range. For example: a threshold of 0 equates to  $-5$  volts, a threshold of 65,535 equates to  $+4.999847$  volts.

When running BURSTMODE scans with the EXTCLOCK option for [cbAInScan\(\)](#) / [AInScan\(\)](#), connect the clock source to the A/D Start Trigger Input (pin 55). Since the trigger input is used as the clock signal, the EXTTRIGGER option cannot be combined with EXTCLOCK BURSTMODE scans. Since BURSTMODE is actually paced by the internal burst clock, specifying EXTCLOCK when using BURSTMODE is equivalent to specifying EXTTRIGGER.

When using the analog trigger feature, one or both of the DACs are used to set the threshold and are unavailable for other functions. If the trigger function requires a single reference (GATEABOVE, GATEBELOW, TRIGABOVE, TRIGBELOW) then DAC0 is available. If the trigger function requires two references (GATEINWINDOW, GATEOUTWINDOW, GATENEGHYS, GATEPOSHYS) then neither DAC is available for other functions.

**Caution!** Gating should NOT be used with BURSTMODE scans.

### **Pacing analog output**

Hardware pacing, external or internal clock supported.

The clock edge used to trigger analog output updates for the external pacer may be rising or falling and is selectable using InstaCal.

EventData for ON\_PRETRIGGER events may not be accurate. In general, this value will be below the actual number of pretrigger samples available in the buffer.

These boards support concurrent analog input and output scans. That is, these boards allow for operations of analog input functions and methods ([cbAInScan\(\)](#) and [cbAPretrig\(\)](#) / [AInScan\(\)](#) and [APretrig\(\)](#)) and analog output functions and methods ([cbAOutScan\(\)](#) / [AOutScan\(\)](#)) to overlap without having to call [cbStopBackground\(\)](#) / [StopBackground\(\)](#) between the start of input and output scans.

### **Output pin 59 configuration**

Pin 59 may be configured as the DAC Pacer Output, SSH Output with hold configured as high level or SSH Output with hold configured as low level. These options are selected via InstaCal

## PCI-DAS6402/16, CIO-DAS6402 Series, and PCI-DAS3202/16

The CIO-DAS6402 Series includes the following hardware:

- CIO-DAS6402/12, CIO-DAS6402/16

This topic also includes the PCI-DAS6402/16 and PCI-DAS3202/16.

The PCI-DAS6402/16, CIO-DAS6402 Series, and PCI-DAS3202/16 support the following UL and UL for .NET features.

### Analog Input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbAPretrig\(\)](#), [cbFileAInScan\(\)](#), [cbFilePretrig\(\)](#)

For PCI-Versions, the following function also applies:

[cbALoadQueue\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [APretrig\(\)](#), [FileAInScan\(\)](#), [FilePretrig\(\)](#)

For PCI-Versions, the following method also applies:

[ALoadQueue\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, BLOCKIO\*, BURSTMODE, EXTTRIGGER

\*Packet size: 512 for both CIO- and PCI- boards.

#### HighChan

PCI-DAS6402 and CIO-DAS6402:

0 to 63 in single-ended mode, 0 to 31 in differential mode

PCI-DAS3202/16:

0 to 31

#### Rate

CIO-DAS6402/12:

Up to 33000 kHz

PCI-DAS3202/16 and PCI-DAS6402/16:

Up to 200000

CIO-DAS6402/16:

Up to 100000

#### Range

PCI versions, CIO-DAS6402/12:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volt)	UNI1PT25VOLTS (0 to 1.25 volt)

CIO-DAS6402/16:

Ignored - not programmable; fixed at one of six switch-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)

### Analog Output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)



## Options

SIMULTANEOUS

For PCI versions, the following argument values are also valid:

BACKGROUND, EXTCLOCK, CONTINUOUS

## HighChan

1 max

## Rate

CIO versions: Ignored

PCI versions: Up to 100000

## Range

PCI versions, CIO-DAS6402/12:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)

CIO-DAS6402/16:

Ignored - not programmable; fixed at one of six switch-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)

## Data Value

0 to 4,095

For PCI-DAS6402/16, PCI-DAS3202/16, CIO-DAS6402/16, the following additional argument values are also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Digital I/O

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#).

For PCI- Versions, the following additional function is also valid:

[cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

For PCI- Versions, the following additional method is also valid:

[DConfigPort\(\)](#)

### PortNum

AUXPORT\*

### DataValue

0 to 15

### BitNum

0 to 3

\*AUXPORT is not configurable for these boards.

### PCI versions:

For PCI-versions, the following additional argument values are also valid. Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

### DataValue

0-15 for FIRSTPORTCL or FIRSTPORTCH

0 -255 for FIRSTPORTA or FIRSTPORTB

### BitNum

0-23 for FIRSTPORTA

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1

### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

### Loadvalue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE, TRIGNEGEDGE, GATEHIGH, GATELOW

For PCI- versions, the following additional argument values are also valid:

TRIGABOVE, TRIGBELOW, GATENEGHYS, GATEPOSHYS, GATEABOVE, GATEBELOW, GATEINWINDOW, GATEOUTWINDOW

### Threshold

0 to 4,095

For /16 versions the following argument values are also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Event Notification (PCI- version only)

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event Types

ON\_SCAN\_ERROR, ON\_PRETRIGGER\*, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_END\_OF\_AO\_SCAN

## Hardware considerations

### Pacing Analog input

Hardware pacing, external or internal clock supported.

### Triggering and gating

Digital (TTL) hardware triggering supported. The PCI version also supports analog hardware triggering. Analog thresholds are set relative to the  $\pm 10V$  range. For example, a threshold of 0 equates to  $-10$  volts (V), and a threshold of 65,535 equates to  $+9.999695$  volts.

When using the [cbAPretrig\(\)](#) or [cbFilePretrig\(\)](#) functions or the [APretrig\(\)](#) or [FilePretrig\(\)](#) methods on the PCI-DAS6402/16 or PCI-DAS3202/16, use the A/D Stop Trigger In (pin 47) input to supply the trigger.

When using both EXTCLK and BURSTMODE on the PCI-DAS6402/16 or PCI-DAS3202/16, use the A/D Start Trigger In (pin 45) input to supply the clock and not the A/D External Pacer (pin 42). Since BURSTMODE is actually paced by the internal burst clock, specifying EXTCLK when using BURSTMODE is equivalent to specifying EXTTRIGGER.

When using analog trigger feature, one or both of the DACs are used to set the threshold and are unavailable for other functions. If the trigger function requires a single reference (GATEABOVE, GATEBELOW, TRIGABOVE, TRIGBELOW) then DAC0 is available. If the trigger function requires two references (GATEINWINDOW, GATE OUTWINDOW, GATENEGHYS, GATEPOSHYS), then neither DAC is available for other functions.

**Caution!** Gating should NOT be used with BURSTMODE scans.

### Gain queue

When using [cbALoadQueue\(\)](#) or [ALoadQueue\(\)](#) with the PCI version, up to 8k elements may be loaded into the queue.

### Pacing analog output

- CIO Version: Software only
- PCI Version: Hardware pacing, external or internal clock supported.

### Output pin 49 configuration

On the PCI version, pin 49 may be configured as the DAC Pacer Output, SSH Output with hold configured as high level or SSH Output with hold configured as low level. These options are selected via InstaCal.

### Event Notification

The **PCI- version** of these boards support concurrent analog input and output scans. That is, these boards allow for operations of analog input functions [cbAInScan\(\)](#) and [cbAPretrig\(\)](#) or methods [AInScan\(\)](#) and [APretrig\(\)](#), and the analog output function [cbAOutScan\(\)](#) or method [AOutScan\(\)](#) to overlap without having to call [cbStopBackground](#) or [StopBackground\(\)](#) between the start of input and output scans.

## PCIe-DAS1602/16

The PCIe-DAS1602/16 supports the following UL and UL for .NET features.

### Analog Input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, BLOCKIO, BURSTMODE, EXTTRIGGER

#### Mode

Single-ended and differential

#### HighChan

0 to 15 in single-ended mode

0 to 7 in differential mode

#### Rate

Up to 100 kS/s

#### Range

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)

### Analog Output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

Ignored

#### HighChan

1 max

#### Count

2

#### Rate

Ignored

#### Range

Ignored - not programmable; fixed at one of four jumper-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)

#### DataValue

0 to 4,095

### Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

#### PortNum

AUXPORT\*, FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

### DataValue

0 to 15 for FIRSTPORTCL, FIRSTPORTCH or AUXPORT\*

0 to 255 for FIRSTPORTA or FIRSTPORTB

### BitNum

0 to 23 for FIRSTPORTA

0 to 3 for AUXPORT\*

\* AUXPORT is not configurable.

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 3

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### EventType

ON\_SCAN\_ERROR, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE, TRIGNEGEDGE, GATEHIGH, GATELOW

### Threshold

0 to 65,535

## Hardware considerations

### Pacing analog input

Hardware pacing, external or internal clock supported.

### Analog input ranges

The A/D ranges are configured with a combination of a switch (Unipolar / Bipolar) and a programmable gain code. The state of this switch is set in the configuration file using InstaCal. After the UNI/BIP switch setting is selected, only matching ranges can be used in Universal Library programs.

### Triggering and gating

Digital (TTL) hardware triggering supported.

### Pacing analog output

Software pacing only

## PCIM-DAS1602/16, PCIM-DAS16JR/16

The PCIM-DAS1602/16 and PCIM-DAS16JR/16 support the following UL and UL for .NET features.

### Analog Input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, BLOCKIO, BURSTMODE, EXTTRIGGER

#### HighChan

0 to 15 in single-ended mode, 0 to 7 in differential mode

#### Rate

Up to 100000

#### Range

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)

### Analog Output (PCIM-DAS1602/16 only)

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

Ignored

#### HighChan

1 max

#### Count

2

#### Rate

Ignored

#### Range

Ignored - not programmable; fixed at one of four jumper-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)

#### DataValue

0 to 4,095

### Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

The PCIM-DAS1602/16 also supports:

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

### PortNum

AUXPORT\*

The PCIM-DAS1602/16 also supports:

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

### DataValue

0 to 15 for FIRSTPORTCL, FIRSTPORTCH or AUXPORT\*

0 to 255 for FIRSTPORTA or FIRSTPORTB

### BitNum

0 to 23 for FIRSTPORTA

0 to 3 for AUXPORT\*

\* AUXPORT is not configurable for these boards.

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 3

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### EventType

ON\_SCAN\_ERROR, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE, TRIGNEGEDGE, GATEHIGH, GATELOW

### Threshold

0 to 65,535

## Hardware considerations

### Pacing analog input

Hardware pacing, external or internal clock supported.

### Analog input ranges

For the PCIM-DAS1602/16, the A/D ranges are configured with a combination of a switch (Unipolar / Bipolar) and a programmable gain code. The state of this switch is set in the configuration file using InstaCal. After the UNI/BIP switch setting is selected, only matching ranges can be used in Universal Library programs.

### Triggering and gating

Digital (TTL) hardware triggering supported.

### Pacing analog output

Software pacing only

## PCM-DAS08

The PCM-DAS08 supports the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, NOTODINTS, EXTTRIGGER, NOCALIBRATEDATA

For information on real time software calibration, refer to the note listed in the function [cbACalibrateData\(\)/ACalibrateData\(\)](#).

### HighChan

7

### Rate

25000 max. For other restrictions, refer to the *PCM-DAS08 User's Manual* and the *Maximizing sampling rates* discussion below.)

### Range

This board does not have programmable gain, so the Range argument to analog input functions is ignored.

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortNum

AUXPORT

### DataValue

0 to 7

### BitNum

0 to 2

## Hardware Considerations

### Pacing analog input

Internal or external clock

### Maximizing sampling rates

When paced by the onboard clock, the rate is set by an onboard oscillator running at 25 kilohertz (kHz). The oscillator output may be divided by 2, 4 or 8, resulting in rates of 12.5 kHz, 6.25 kHz, or 3.13 kHz. When pacing a single channel from the onboard clock, these are the four choices of rate available. When a rate is requested within the range of 3000 to 25000, the library selects the closest of the four available rates.

Scanning more than one channel has the effect of dividing the rate requested among the number of channels requested. Therefore, the maximum rate when scanning eight channels is 3130 (25000 divided by eight channels).



## PCM-DAS16 Series and PC-CARD-DAS16 Series

The PCM-DAS16 Series and PC-CARD-DAS16 Series includes the following hardware:

- PCM-DAS16D/12
- PPCM-DAS16D/12AO
- PPCM-DAS16D/16
- PPCM-DAS16S/12
- PPCM-DAS16S/16
- PPCM-DAS16S/330

The PCM-DAS16 Series and PC-CARD-DAS16 Series includes the following hardware:

- PC-CARD-DAS16/12
- PC-CARD-DAS16/12AO
- PC-CARD-DAS16/16
- PC-CARD-DAS16/16AO
- PC-CARD-DAS16/330

The PCM-DAS16 Series and PC-CARD-DAS16 Series support the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, SINGLEIO, BLOCKIO, EXTTRIGGER, NOTODINTS, NOCALIBRATEDATA

The PC-CARD-DAS16 series also supports BURSTMODE.

### HighChan

DAS16/S and DAS16/330: 0 to 15

DAS16/D: 0 to 7

### Rate

DAS16/330: 330000

PC-CARD-DAS16/16: 200000

All others in this series: 100000

### Range

DAS16x/12:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	UNI1PT25VOLTS (0 to 1.25 volts)

DAS16x/16:

BIP10VOLTS ( $\pm 10$  volts)  
BIP5VOLTS ( $\pm 5$  volts)  
BIP2PT5VOLTS ( $\pm 2.5$  volts)  
BIP1PT25VOLTS ( $\pm 1.25$  volts)

DAS16/330:

BIP10VOLTS ( $\pm 10$  volts)  
BIP5VOLTS ( $\pm 5$  volts)

## Analog Output

PCM-DAS16D/12AO and PC-CARD-DAS16/xxAO only.

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS (PCM version only)

### HighChan

1 max

### Rate

Ignored

### Count

2 max

### Range

PC-CARD-DAS16/12AO and PCM-DAS16D/12AO:

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

All others in this series:

Ignored - not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

### DataValue

0 to 4,095

For PC-CARD-DAS16/16AO, the following argument values is also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Pacing

Software pacing only

## Digital I/O

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

### PortNum

PC-CARD-DAS16/xxAO:

FIRSTPORTA

All others in this series:

FIRSTPORTA, FIRSTPORTB

### DataValue

PC-CARD-DAS16/xxAO:

0 to 15 for FIRSTPORTA

All others in this series

0 to 15 for FIRSTPORTA or FIRSTPORTB

### BitNum

PC-CARD-DAS16/xxAO:

0 to 3 for FIRSTPORTA

All others in this series:

0 to 7 for FIRSTPORTA

## Counter I/O

### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 3

### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### RegNum

LOADREG1, LOADREG2, LOADREG3

## Triggering

PC-CARD only

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE, TRIGNEGEDGE, GATEHIGH, GATELOW

All at External Trigger input.

## Hardware considerations

### Pacing analog input

- Internal or external clock
- The packet size is 256 samples for PCM boards; 2,048 samples for PC-CARD boards.
- For CONTINUOUS mode scans, the sample count should be at least one packet size ( $\geq 2,048$  samples) for the PC-CARD boards.

These cards do not have residual counters, so BLOCKIO transfers must acquire integer multiples of the packet size before completing the scan. This can be lengthy for the PC-CARDS which must acquire 2,048 samples between interrupts for BLOCKIO transfers. In general, it is best to allow the library to determine the best transfer mode (SINGLEIO vs. BLOCKIO) for these boards.

### Triggering and gating

- External digital (TTL) polled gate trigger supported on PCM versions. Refer to the "[Trigger support](#)" section of the "Introduction: Analog input Boards" topic.
- External digital (TTL) hardware trigger supported on PC-CARD versions.

## PPIO-AI08

The PPIO-AI08 supports the following UL and UL for .NET features.

### Analog Input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

#### Options

CONVERTDATA

#### HighChan

0 to 7

#### Rate

Ignored

#### Range

This board does not have programmable gain, so the Range argument to analog input functions is ignored.

### Digital I/O

#### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

#### PortNum

AUXPORT\*

#### DataValue

0 to 15 using [cbDOut\(\)](#) or [DOut\(\)](#)

0 to 7 using [cbDIn\(\)](#) or [DIn\(\)](#)

#### BitNum

0 to 3 using [cbDBitOut\(\)](#) or [DBitOut\(\)](#)

0 to 2 using [cbDBitIn\(\)](#) or [DBitIn\(\)](#)

\* AUXPORT is not configurable for this board.

### Hardware Considerations

#### Pacing analog input

Software pacing only

## USB-1208FS and USB-1408FS

The USB-1208FS and USB-1408FS supports the following UL and UL for .NET features.

### Analog input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

#### Options

BACKGROUND, BLOCKIO\*, CONTINUOUS, EXTCLOCK, EXTTRIGGER, NOCALIBRATEDATA, RETRIGMODE\*\*, SINGLEIO

\* The USB-1208FS packet size is based on the Options setting:

BLOCKIO: 31

SINGLEIO: 1

\*\* RETRIGMODE can only be used with [cbAInScan\(\)](#)/[AInScan\(\)](#).

#### Mode

Single-ended and differential

#### HighChan

0 to 7 in single-ended mode

0 to 3 in differential mode

#### Count

In CONTINUOUS mode, Count must be an integer multiple of the packet size.

#### Rate

USB-1208FS: 50 kHz maximum for BLOCKIO mode.

USB-1408FS: 48 kHz maximum for BLOCKIO mode.

The throughput depends on the system being used. Most systems can achieve 40 kHz aggregate.

When using [cbAInScan\(\)](#)/[AInScan\(\)](#), the minimum sample rate is 1 Hz.

#### Range

Single-ended:

BIP10VOLTS ( $\pm 10$  volts)

Differential:

BIP20VOLTS ( $\pm 20$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

BIP10VOLTS ( $\pm 10$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP1PT25VOLTS ( $\pm 1.25$  volts)

BIP4VOLTS ( $\pm 4$  volts)

BIP1VOLTS ( $\pm 1$  volts)

#### Pacing

Hardware pacing, internal clock supported.

External clock supported via the SYNC pin.

### Triggering

#### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

#### TrigType

TRIGPOSEDGE, TRIGNEGEDGE

Both devices support external digital (TTL) hardware triggering. Use the TRIG\_IN input for the external trigger signal.

### Analog Output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

## Options

BACKGROUND, CONTINUOUS

The number of samples (Count) in a CONTINUOUS scan must be an integer multiple of the packet size (32).

## HighChan

0 to 1

## Count

Count must be an integer multiple of the number of channels in the scan.

In a CONTINUOUS scan, Count must be an integer multiple of the packet size (32).

## Rate

Up to 10 kHz maximum for a single channel

Up to 5 kHz maximum for two channels

## Range

Ignored - not programmable; fixed at UNI4VOLTS (0 to 4 V, nominal. Actual range is 0 to 4.096 V.)

## DataValue

0 to 4,095

## Digital I/O

- ▶ [Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.](#)

## Configuration Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

## Port I/O Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

## Bit I/O Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 15 for on FIRSTPORTA

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#)/[CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#)/[CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

## CounterNum

1

## Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

cbCLoad()/CLoad() and cbCLoad32()/CLoad32() are only used to reset the counter for this device to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* section apply when using cbCIn() or CIn() for values greater than 32,767 and when using cbCIn32() or CIn32() for values greater than 2,147,483,647.

## RegNum

LOADREG1

## Event notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event types

ON\_SCAN\_ERROR (analog input and analog output), ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_END\_OF\_AO\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### Acquisition rate

Since the maximum data acquisition rate depends on the system connected to the device, it is possible to "lose" data points when scanning at higher rates. The Universal Library cannot always detect this data loss.

Most systems can sustain rates of 40 kS/s aggregate in BLOCKIO mode, and 1 kS/s aggregate in SINGLEIO mode.

### Channel-gain queue

When using cbALoadQueue()/ALoadQueue(), the channel gain queue is limited to 16 elements. The queue accepts any combination of valid channels and gains in each element.

### EXTCLOCK

By default, the SYNC pin is configured for pacer output and provides the internal pacer A/D clock signal. To configure the pin for pacer input, use the EXTCLOCK option.

If you use the EXTCLOCK option, make sure that you disconnect from the external clock source when you test or calibrate the device with InstaCal, as the SYNC pin drives the output.

### Repetitive trigger events

Use [RETRIGMODE](#) with cbAInScan() to set up repetitive trigger events. Use the ConfigItem option BIADTRIGCOUNT with [cbSetConfig\(\)](#) to set the A/D trigger count, and the ConfigItem option BIDACTRIGCOUNT to set the D/A trigger count.

### Resolution

When configured for single-ended mode, the resolution of the data is 11 bits (data values between 0 and 2,047). However, the Universal Library maps this data to 12-bit values, so the range of data is no different from the differential configuration. Consequently, the data returned contains only even numbers between 0 and 4,094 when the NOCALIBRATEDATA option is used.

### Continuous scans

When running [cbAInScan\(\)](#)/ [AInScan\(\)](#) with the CONTINUOUS option, consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, set the total number of samples to be an integer multiple of the packet size and the number of channels in the scan.

### Concurrent operations

The USB-1208FS and USB-1408FS supports these concurrent operations:

UL function/method	Can be run with:
<a href="#">cbAOutScan()</a> / <a href="#">AOutScan()</a> (BACKGROUND mode)	<ul style="list-style-type: none"><li>■ <a href="#">cbDOut()</a> / <a href="#">DOut()</a></li><li>■ <a href="#">cbCLoad()</a> / <a href="#">CLoad()</a></li></ul>

	<ul style="list-style-type: none"> <li>▪ <a href="#">cbCLoad32()</a> / <a href="#">CLoad32()</a></li> </ul>
<a href="#">cbAInScan()</a> / <a href="#">AInScan()</a> (BACKGROUND mode)	<ul style="list-style-type: none"> <li>▪ <a href="#">cbAOut()</a> / <a href="#">AOut()</a></li> <li>▪ <a href="#">cbDIn()</a> / <a href="#">DIn()</a></li> <li>▪ <a href="#">cbDBitIn()</a> / <a href="#">DBitIn()</a></li> <li>▪ <a href="#">cbDOut()</a> / <a href="#">DOut()</a></li> <li>▪ <a href="#">cbDBitOut()</a> / <a href="#">DBitOut()</a></li> <li>▪ <a href="#">cbDConfigPort()</a> / <a href="#">DConfigPort()</a></li> <li>▪ <a href="#">cbCIn()</a> / <a href="#">CIn()</a></li> <li>▪ <a href="#">cbCIn32()</a> / <a href="#">CIn32()</a></li> <li>▪ <a href="#">cbCLoad()</a> / <a href="#">CLoad()</a></li> <li>▪ <a href="#">cbCLoad32()</a> / <a href="#">CLoad32()</a></li> </ul>

### Analog output

When you include both analog output channels in [cbAOutScan\(\)](#)/[AOutScan\(\)](#), the two channels are updated simultaneously.



## USB-1208FS-Plus and USB-1408FS-Plus

The USB-1208FS-Plus and USB-1408FS-Plus support the following UL and UL for .NET features.

### Analog input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

#### Options

BACKGROUND, BLOCKIO, CONTINUOUS, EXTCLOCK, EXTTRIGGER, HIGHRESRATE, NOCALIBRATEDATA, RETRIGMODE\*, SINGLEIO

\* RETRIGMODE can only be used with [cbAInScan\(\)](#)/[AInScan\(\)](#).

#### Mode

Single-ended and differential

#### HighChan

0 to 7 in single-ended mode

0 to 3 in differential mode

#### Count

Count must be an integer multiple of the number of channels in the scan.

#### Packet size

Rate dependent. The default packet size is 32 samples. At higher rates, the packet size increases by a multiple of 32.

#### Rate

USB-1208FS: 0.014 Hz to 51.993 kHz for BLOCKIO mode.

USB-1408FS: 0.14 Hz to 48 kHz maximum for BLOCKIO mode.

The throughput depends on the system being used. Most systems can achieve 40 kHz aggregate.

When using [cbAInScan\(\)](#)/[AInScan\(\)](#), the minimum sample rate is 1 Hz.

#### Range

Single-ended:

BIP10VOLTS ( $\pm 10$  volts)

Differential:

BIP20VOLTS ( $\pm 20$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

BIP10VOLTS ( $\pm 10$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP1PT25VOLTS ( $\pm 1.25$  volts)

BIP4VOLTS ( $\pm 4$  volts)

BIP1VOLTS ( $\pm 1$  volts)

#### Pacing

Hardware pacing, internal clock supported.

External clock supported via the SYNC pin.

### Triggering

#### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

#### TrigType

TRIGPOSEDGE, TRIGNEGEDGE, TRIGHIGH, TRIGLOW

Both devices support external digital (TTL) hardware triggering. Use the TRIG\_IN input for the external trigger signal.

### Analog Output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

## Options

BACKGROUND, CONTINUOUS

## HighChan

0 to 1

## Count

Count must be an integer multiple of the number of channels in the scan.

## Packet size

Rate dependent.

## Rate

Up to 10 kHz maximum for a single channel

Up to 5 kHz maximum for two channels

## Range

Ignored - not programmable; fixed at UNI5VOLTS (0 to 5 V, nominal. Actual range is 0 to 4.096 V.)

## DataValue

0 to 4,095

## Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

## Configuration Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

## Port I/O Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

## Bit I/O Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 15 for on FIRSTPORTA

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#)/[CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#)/[CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

## CounterNum

1

## Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

cbCLoad()/CLoad() and cbCLoad32()/CLoad32() are only used to reset the counter for this device to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* section apply when using cbCIn() or CIn() for values greater than 32,767 and when using cbCIn32() or CIn32() for values greater than 2,147,483,647.

## RegNum

LOADREG1

## Event notification

### Functions

UL: [cbEnableEvent\(\),cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\),DisableEvent\(\)](#)

### Event types

ON\_SCAN\_ERROR (analog input and analog output), ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_END\_OF\_AO\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### Channel-gain queue

The channel-gain queue is limited to 8 elements in single-ended mode, and 4 elements in differential mode. The channels specified must be unique and listed in increasing order. The gains may be any valid value.

### Acquisition rate

Since the maximum data acquisition rate depends on the system connected to the device, it is possible to "lose" data points when scanning at higher rates. The Universal Library cannot always detect this data loss.

Most systems can sustain rates of 40 kS/s aggregate in BLOCKIO mode, and 1 kS/s aggregate in SINGLEIO mode.

### HIGHRESRATE

Specify the HIGHRESRATE scan option to acquire data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel".

### EXTCLOCK

By default, the SYNC pin is configured for pacer output and provides the internal pacer A/D clock signal. To configure the pin for pacer input, use the EXTCLOCK option.

If you use the EXTCLOCK option, make sure that you disconnect from the external clock source when you test or calibrate the device with InstaCal, as the SYNC pin drives the output.

### Retriggering

Use [RETRIGMODE](#) with cbAInScan() to set up repetitive trigger events. Use the ConfigItem option BIADTRIGCOUNT with [cbSetConfig\(\)](#) to set the A/D trigger count, and the ConfigItem option BIDACTRIGCOUNT to set the D/A trigger count.

### Resolution

When configured for single-ended mode, the resolution of the data is 11 bits (data values between 0 and 2,047). However, the Universal Library maps this data to 12-bit values, so the range of data is no different from the differential configuration. Consequently, the data returned contains only even numbers between 0 and 4,094 when the NOCALIBRATEDATA option is used.

### Continuous scans

When running [cbAInScan\(\)](#)/ [AInScan\(\)](#) with the CONTINUOUS option, set the count to be an integer multiple of the number of channels in the scan in order to keep the data aligned properly in the array.

### Analog output

When you include both analog output channels in cbAOutScan()/AOutScan(), the two channels are updated simultaneously.

## USB-1208HS Series

The USB-1208HS Series includes the following devices:

- USB-1208HS
- USB-1208HS-2AO
- USB-1208HS-4AO

The USB-1208HS Series supports the following UL and UL for .NET features:

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

BACKGROUND, BLOCKIO, BURSTMODE, CONTINUOUS, EXTCLOCK, EXTTRIGGER, NOCALIBRATEDATA, RETRIGMODE\*, SINGLEIO

### Packet size

Rate dependent. The default packet size is 256 samples. At higher rates, the packet size increases by a multiple of 256.

### Count

Count must be an integer multiple of the number of channels in the scan.

### Mode

Single-ended or differential

### HighChan

0 to 7 in single-ended mode

0 to 3 in differential mode

### Rate

1 MS/s, maximum

### Range

Single-ended mode:

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

UNI10VOLTS (0 to 10 volts)

Differential mode:

BIP20VOLTS ( $\pm 20$  volts)

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TrigPosEdge, TrigNegEdge, TrigHigh, TrigLow

## Analog output (USB-1208HS-2AO and USB-1208HS-4AO only)

### Functions

UL: [cbAOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [AOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER, RETRIGMODE

### Range

BIP10VOLTS ( $\pm 10$  Volts)

### HighChan

USB-1208HS-2AO: 0 to 1

USB-1208HS-4AO: 0 to 3

### Packet size

Rate dependent.

### Count

Count must be an integer multiple of the number of channels in the scan.

### Rate

1 MHz per channel, maximum

## Digital I/O

### Configuration

Functions

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AuxPort

PortType

AuxPort

### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AuxPort

DataValue

0 to 65,535

### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortType

AuxPort

BitNum

0 to 15 on AuxPort

## Counter I/O

### Functions

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCLoad\(\)](#), [cbCLoad32\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CLoad\(\)](#), [CLoad32\(\)](#)

Note: Counters on these boards are zero-based (the first counter number is "0").

### CounterNum

0 to 1

### Count

$2^{32}$  when reading the counter.

**LoadValue**

0 when loading the counter.

cbCLoad() and cbCLoad32() CLoad() and CLoad32() are only used to reset the counter to 0. No other values are valid.

Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.

**RegNum**

LOADREG0, LOADREG1

**Timer****Functions**

UL: [cbPulseOutStart\(\)](#), [cbPulseOutStop\(\)](#)

UL for .NET: [PulseOutStart\(\)](#), [PulseOutStop\(\)](#)

**TimerNum**

0

**Frequency**

0.0094 Hz to 20 MHz

**DutyCycle**

0 to 1, non-inclusive

**InitialDelay**

0 sec to 107.37 sec

**IdleState**

IDLE\_LOW, IDLE\_HIGH

**PulseCount**

0 to  $2^{32} - 1$  (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

Set to 0 to continuously generate pulses until PulseOutStop() is called.

**Configuration****Functions**

UL: [cbSetConfig\(\)](#)

**InfoType**

BOARDINFO

**ConfigItem**

BIADTRIGCOUNT, BIDACTRIGCOUNT

**Device number**

0

**String Configuration****Functions**

UL: [cbGetConfigString\(\)](#), [cbSetConfigString\(\)](#)

**InfoType**

BOARDINFO

**ConfigItem**

BINODEID

**maxConfigLen**

At least 64 for BINODEID

**Event Notification****Functions**

UL: [cbEnableEvent\(\)](#) [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#) [DisableEvent\(\)](#)

## Event types

UL: [ON\\_DATA\\_AVAILABLE](#), [ON\\_END\\_OF\\_AI\\_SCAN](#), [ON\\_SCAN\\_ERROR](#), [ON\\_END\\_OF\\_AO\\_SCAN](#)

UL for .NET: [OnDataAvailable](#), [OnEndOfAiScan](#), [OnScanError](#), [OnEndOfAoScan](#)

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a Measurement Computing USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware Considerations

### Retriggering

Use [RETRIGMODE](#) with [cbAInScan\(\)](#) to set up repetitive trigger events. Use the ConfigItem option [BIADTRIGCOUNT](#) with [cbSetConfig\(\)](#) to set the A/D trigger count, and the ConfigItem option [BIDACTRIGCOUNT](#) to set the D/A trigger count.

When using [RETRIGMODE](#), set the values for the Count argument ([cbAInScan\(\)/AInScan\(\)](#)) and the [BIADTRIGCOUNT](#) argument ([cbSetConfig\(\)/SetAdRetrigCount\(\)](#)) to an integer multiple of the number of channels in the scan. so that the entire buffer, or the portion of the buffer defined by [BIADTRIGCOUNT](#), will contain updated data.

### Device identifier

You can enter up to 64 characters for the value of the device's text identifier using the ConfigItem option [BINODEID](#) with [cbSetConfigString\(\)](#).

# USB-1208LS

The USB-1208LS supports the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

### Options

BACKGROUND, BLOCKIO\*, BURSTIO\*\*, CONTINUOUS, EXTTRIGGER, NOCALIBRATEDATA, CONVERTDATA

\*The packet size is based on the Options setting. When set to BLOCKIO, the packet size is 64 samples.

\*\* BURSTIO can only be used with the number of samples (Count) set equal to the size of the FIFO or less. The USB-1208LS FIFO holds 4096 samples. BURSTIO cannot be used with the CONTINUOUS option.

### HighChan

0 to 7 in single-ended mode

0 to 3 in differential mode

### Count

In CONTINUOUS mode, Count must be an integer multiple of the packet size.

### Rate

8000 Hz maximum for BURSTIO mode.

The maximum rate is 1200 Hz for all other modes.

When using [cbAInScan\(\)](#) or [AInScan\(\)](#), the minimum sample rate is 100 Hz.

### Range

Single-ended:

BIP10VOLTS ( $\pm 10$  volts)

Differential

BIP20VOLTS ( $\pm 20$  volts)

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP4VOLTS ( $\pm 4$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP1PT25VOLTS ( $\pm 1.25$  volts)

BIP1VOLTS ( $\pm 1$  volts)

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGHIGH, TRIGLOW

External digital (TTL) hardware triggering is supported. Use the TRIG\_IN input (pin # 18 on the screw terminal) for the external trigger signal.

## Analog Output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

All settings are ignored

### HighChan

0 to 1

### Count

(HighChan - LowChan) + 1



## Rate

All settings are ignored

## Range

Ignore - Not programmable; fixed at UNI5VOLTS (0 to 5 volts)

## DataValue

0 to 1023

## Digital I/O

- This board has an 82C55 chip. Click [here](#) to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

## Configuration

Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

## Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

## Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 15 for on FIRSTPORTA

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

### Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#)/[CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter for this device to 0. No other values are valid.

The [Basic signed integers](#) guidelines apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

### RegNum

## Event notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event types

ON\_SCAN\_ERROR (analog input), ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### BURSTIO

BURSTIO mode allows higher sampling rates for sample counts up to the size of the FIFO. The USB-1208LS FIFO holds 4096 samples. Data is collected into the device's local FIFO. Data transfers to the PC don't occur until the scan completes. For BACKGROUND scans, the Count and Index returned by [cbGetStatus\(\)](#) and [GetStatus\(\)](#) remain 0, and Status=RUNNING until the scan finishes. The Count and Index are not updated until the scan is completed. When the scan is complete and the data is retrieved, [cbGetStatus\(\)](#) and [GetStatus\(\)](#) are updated to the current Count and Index, and Status = IDLE.

The USB-1208LS uses BURSTIO as the default mode for non-CONTINUOUS fast scans with sample counts up to the size of the FIFO (4096 samples). BURSTIO mode allows higher sampling rates for sample counts up to the size of the FIFO. Maximum Rate values of non-BURSTIO scans are limited (refer to the [Rate](#) option above). To avoid the BURSTIO default, specify BLOCKIO mode.

### Resolution

When configured for single-ended mode, the resolution of the data is 11 bits (data values between 0 and 2,047). However, the Universal Library maps this data to 12-bit values, so the range of data is no different from the differential configuration. Consequently, the data returned contains only even numbers between 0 and 4,094 when the NOCALIBRATEDATA option is used.

### Continuous scans

When running [cbAInScan\(\)/AInScan\(\)](#) with the CONTINUOUS option, consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, set the total number of samples to be an integer multiple of the packet size and the number of channels in the scan.

### Concurrent operations

Concurrent operations are not allowed. If you invoke a UL or UL for .NET function on a USB-1208LS while another function is running on that same unit, the ALREADYACTIVE error is returned.

### Channel-gain queue

When using [cbALoadQueue\(\)/ALoadQueue\(\)](#), the channel gain queue is limited to eight elements.

### Analog output

When you include both analog output channels in [cbAOutScan\(\)/AOutScan\(\)](#), the two channels are updated simultaneously.

## USB-1602HS Series

The USB-1602HS Series includes the following devices:

- USB-1602HS
- USB-1602HS-2AO

The USB-1602HS Series supports the following UL and UL for .NET features.

### Analog Input

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbAPretrig\(\)\\*](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [APretrig\(\)\\*](#), [ATrig\(\)](#)

\*Pretrigger capability is implemented in software. PretrigCount must be less than the TotalCount and cannot exceed 100000 samples. If a trigger occurs while the number of collected samples is less than the PretrigCount, that trigger will be ignored. Requires a call to [cbSetTrigger](#)/ [SetTrigger](#) for the analog trigger type.

#### Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK\*, EXTTRIGGER, HIGHRESRATE

\* With EXTTRIGGER mode, the first channel in the scan is the analog trigger channel.

#### HighChan

0 to 1

#### Count

Count must be an integer multiple of the number of channels in the scan.

#### Rate

Up to 2 MHz

#### Range

BIP10VOLTS ( $\pm 10$  volts)  
BIP2PT5VOLTS ( $\pm 2.5$  volts)  
BIPP5VOLTS ( $\pm 0.5$  volts)

## Analog Output (USB-1602HS-2AO only)

UL: [cbAOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [AOutScan\(\)](#)

#### Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS

NONSTREAMEDIO can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

#### HighChan

0 to 1

#### Rate

1 MHz each channel

#### Count

Count must be an integer multiple of the number of channels in the scan.

#### Range

Ignored - Not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

#### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

#### Pacing

Hardware pacing, external or internal clock supported.

## Digital I/O

#### Port I/O

## Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDInScan\(\)](#), [cbDOutScan\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DInScan\(\)](#), [DOutScan\(\)](#)

## Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER, HIGHRESRATE, NONSTREAMEDIO

- The EXTTRIGGER option can only be used with the [cbDInScan\(\)](#) function. You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).
- The NONSTREAMEDIO, ADCCLOCKTRIG, and ADCCLOCK options can only be used with the [cbDOutScan\(\)](#) function.
- The HIGHRESRATE option can only be used with the [cbDInScan\(\)](#) function.

## Rate

8 MHz

## PortNum

AUXPORT

## DataValue

0 to 65,535

## Bit I/O

### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortType

AUXPORT

### BitNum

0 to 15

## Counter Input

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCConfigScan\(\)](#), [cbCInScan\(\)](#), [cbCClear\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CConfigScan\(\)](#), [CInScan\(\)](#), [CClear\(\)](#)

Note: Counters on these devices are zero-based (the first counter number is "0").

## Rate

4 MHz

## CounterNum

0 to 3

## Options

BACKGROUND, CONTINUOUS, EXTTRIGGER, HIGHRESRATE

You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).

## LoadValue

0 to 65,535

The [Visual Basic signed integers](#) guidelines apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

## Timers

UL: [cbPulseOutStart\(\)](#), [cbPulseOutStop\(\)](#)

UL for .NET: [PulseOutStart\(\)](#), [PulseOutStop\(\)](#)

## TimerNum

0 to 1

## Frequency

0.112 Hz to 24 MHz

## Duty cycle

0 to 1, non-inclusive

## PulseCount

Ignored

## Triggering

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

## TrigType

TRIGABOVE, TRIGBELOW, TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE

- Digital triggering (TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE) is not supported for pre-trigger acquisitions ([cbAPretrig\(\)](#) function).
- Analog triggering (TRIGABOVE, TRIGBELOW) is not supported for the [cbDInScan\(\)](#) function and the [cbCInScan\(\)](#) function.

## Threshold

Analog hardware triggering, 12-bit resolution: 0 to 4,095 (supported for [cbAInScan\(\)](#) only)

Analog software triggering, 16-bit resolution: 0 to 65,535 (supported for [cbAPretrig\(\)](#) only)

## DAQ Input

UL: [cbDagInScan\(\)](#)

UL for .NET: [DagInScan\(\)](#)

## Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK, EXTTRIGGER, HIGHRESRATE

## ChanTypeArray

ANALOG, DIGITAL16, CTR16, CTR32LOW, CTR32HIGH

- When mixing ANALOG channel types with any other input types, the ANALOG channels should be the first in the list.

## ChanArray

ANALOG: 0 to 3

DIGITAL16: AUXPORT

CTR16: 0-3 counters

CTR32LOW: 0-3 counters

CTR32HIGH: 0-3 counters

SETPOINTSTATUS: 16-bit port that indicates the current state of the 16 possible setpoints.

ChanTypeArray flag value:

- SETPOINT\_ENABLE: Enables a setpoint. Refer to [Setpoints](#) in "Hardware Considerations" below for more information.

## Rate

Analog: Up to 2 MHz

Digital: Up to 8 MHz if no analog channel is selected. Otherwise up to 2 MHz.

Counter: Up to 8 MHz if no analog channel is selected. Otherwise up to 2 MHz.

## GainArray

ANALOG only; ignore for other ChanTypeArray values.

BIP10VOLTS ( $\pm 10$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

BIPPT5VOLTS ( $\pm 0.5$  volt)

## PretrigCount

100000 max

## DAQ Triggering

UL: [cbDagSetTrigger\(\)](#)

UL for .NET: [DagSetTrigger\(\)](#)

## TrigSource

TRIG\_IMMEDIATE, TRIG\_EXTTTL, TRIG\_ANALOGHW, TRIG\_ANALOGSW, TRIG\_DIGPATTERN, TRIG\_COUNTER, TRIG\_SCANCOUNT

### TrigSense

RISING\_EDGE, FALLING\_EDGE, ABOVE\_LEVEL, BELOW\_LEVEL, EQ\_LEVEL, NE\_LEVEL

### TrigEvent

START\_EVENT, STOP\_EVENT

## DAQ Setpoint

UL: [cbDagSetSetpoints\(\)](#)

UL for .NET: [DagSetSetpoints\(\)](#)

### SetpointFlagsArray

SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA, SF\_GREATERTHAN\_LIMITB, SF\_OUTSIDE\_LIMITS, SF\_HYSTERESIS, SF\_UPDATEON\_TRUEONLY, SF\_UPDATEON\_TRUEANDFALSE

### SetpointOutputArray

SO\_NONE, SO\_DIGITALPORT

Also available for USB-1602HS-2AO:

SO\_DAC0, SO\_DAC1

### LimitAArray

Any value valid for the associated input channel.  
Ignored for SF\_GREATERTHAN\_LIMITB

### LimitBArray

Any value valid for the associated input channel and less than LimitA.  
Ignored for SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA

### Output#Array

For SetpointOutputArray = SO\_NONE: Ignored

For SetpointOutputArray = SO\_DIGITALPORT: 0 to 65,535

For SetpointOutputArray = SO\_DAC#: Voltage values between -10 and +10

### OutputMask#Array

For SetpointOutputArray = SO\_DIGITALPORT: 0 to 65,535

For SetpointOutputArray = all other values: Ignored

### SetpointCount

0 (to disable setpoints) to 16

## DAQ Output (USB-1602HS-2AO only)

UL: [cbDagOutScan\(\)](#)

UL for .NET: [DagOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS, ADCCLOCKTRIG, ADCCLOCK

### ChanType

ANALOG, DIGITAL16

### ChanArray

ANALOG: 0 to 1

DIGITAL16: AUXPORT

### Rate

Analog: Up to 2 MHz

Digital16: Up to 8 MHz (system-dependent) if no analog channel is selected. Otherwise up to 2 MHz.

### Range

BIP10VOLTS ( $\pm 10$  volts)

## Hardware considerations

## Channel count

For input and output scans, the count must be set to an integer multiple of the number of channels in the scan or a [BADCOUNT](#) error is returned.

## Sampling and update rates

Sampling and update rates are system-dependent. Data overruns/underruns may occur with higher sampling rates when using BACKGROUND and CONTINUOUS modes. To avoid this, use a larger buffer/count size, or use NONSTREAMEDIO mode, if supported.

The minimum size buffer is 256 for [cbAOutScan\(\)](#). Values less than that result in a [BADBUFFERSIZE](#) error.

## Settling time

For most applications, settling time should be left at the default value of 1  $\mu$ s. However, if you are scanning multiple channels and one or more channels are connected to a high impedance source, you may get better results by increasing the settling time. Keep in mind that increasing the settling time reduces the maximum acquisition rate. You can set the time between A/D conversions with the ADC Settling Time option in InstaCal. Select between 1  $\mu$ s, 5  $\mu$ s, 10  $\mu$ s, or 1 ms.

## Setpoints

You enable setpoints with the SETPOINT\_ENABLE flag. This flag must be OR'ed with the ChanTypeArray argument values. You set the setpoint criteria with cbDaqSetSetpoints()/DaqSetSetpoints(). The number of channels set with the SETPOINT\_ENABLE flag must match the number of setpoints set by the SetpointCount argument (cbDaqSetSetpoints()/DaqSetSetpoints()).

## Output non-streamed data to a DAC output channel

With NONSTREAMEDIO mode, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. The FIFO holds 524288 samples. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample updates are transferred or downloaded to the device, the device is responsible for outputting the data. You can't change the output buffer once the output begins.

## Trigger DAC output operations with the ADC clock

Specify the ADCCLOCK option to trigger a data output operation upon the start of the ADC clock.

## Quadrature encoder operations

To configure a counter channel as a multi-axis quadrature encoder, use the [cbCConfigScan\(\)/ CConfigScan\(\)](#) Mode argument values to set a specified counter to encoder mode, set the encoder measurement mode to X1, X2, or X4, and then set the count to be latched either by the internal "start of scan" signal (default) or by the signal on the mapped channel.

You can optionally perform the following operations:

- Enable gating, so that the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.
- Enable "latch on Z" to latch counter outputs using the Encoder Z mapped signal.
- Enable "clear on Z" so that the counter is cleared on the rising edge of the mapped (Z) channel. By default, "clear on Z" is disabled, and the counter is not cleared.

## Asynchronous reads

The CConfigScan() method's Bit32 counter mode option only affects counter resolution for asynchronous calls ([CIn\(\)](#) and [CIn32\(\)](#)), and only when the counter is configured for StopAtMax.

This mode is recommended for use only with CIn32(). Using the Bit32 option with CIn() is not very useful, since the value returned by CIn() is only 16 bits. The effect is that the value returned by CIn() rolls over 65,535 times before stopping.

## USB-1604HS Series

The USB-1604HS Series includes the following devices:

- USB-1604HS
- USB-1604HS-2AO

The USB-1604HS Series supports the following UL and UL for .NET features.

### Analog Input

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbAPretrig\(\)\\*](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [APretrig\(\)\\*](#), [ATrig\(\)](#)

\*Pretrigger capability is implemented in software. PretrigCount must be less than the TotalCount and cannot exceed 100000 samples. If a trigger occurs while the number of collected samples is less than the PretrigCount, that trigger will be ignored. Requires a call to [cbSetTrigger/ SetTrigger](#) for the analog trigger type.

#### Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK\*, EXTTRIGGER, HIGHRESRATE

\* With EXTTRIGGER mode, the first channel in the scan is the analog trigger channel.

#### HighChan

0 to 3

#### Count

Count must be an integer multiple of the number of channels in the scan.

#### Rate

Up to 1.33 MHz

#### Range

BIP10VOLTS ( $\pm 10$  volts)  
BIP2PT5VOLTS ( $\pm 2.5$  volts)  
BIPP5VOLTS ( $\pm 0.5$  volts)

## Analog Output (USB-1604HS-2AO only)

UL: [cbAOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [AOutScan\(\)](#)

#### Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS

NONSTREAMEDIO can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

#### HighChan

0 to 1

#### Rate

1 MHz each channel

#### Count

Count must be an integer multiple of the number of channels in the scan.

#### Range

Ignored - Not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

#### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

#### Pacing

Hardware pacing, external or internal clock supported.

## Digital I/O

#### Port I/O



## Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDInScan\(\)](#), [cbDOutScan\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DInScan\(\)](#), [DOutScan\(\)](#)

## Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER, HIGHRESRATE, NONSTREAMEDIO

- The EXTTRIGGER option can only be used with the [cbDInScan\(\)](#) function. You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).
- The NONSTREAMEDIO, ADCCLOCKTRIG, and ADCCLOCK options can only be used with the [cbDOutScan\(\)](#) function.
- The HIGHRESRATE option can only be used with the [cbDInScan\(\)](#) function.

## Rate

8 MHz

## PortNum

AUXPORT

## DataValue

0 to 65,535

## Bit I/O

### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortType

AUXPORT

### BitNum

0 to 15

## Counter Input

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCConfigScan\(\)](#), [cbCInScan\(\)](#), [cbCClear\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CConfigScan\(\)](#), [CInScan\(\)](#), [CClear\(\)](#)

Note: Counters on these devices are zero-based (the first counter number is "0").

## Rate

4 MHz

## CounterNum

0 to 3

## Options

BACKGROUND, CONTINUOUS, EXTTRIGGER, HIGHRESRATE

You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).

## LoadValue

0 to 65,535

The [Visual Basic signed integers](#) guidelines apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

## Timers

UL: [cbPulseOutStart\(\)](#), [cbPulseOutStop\(\)](#)

UL for .NET: [PulseOutStart\(\)](#), [PulseOutStop\(\)](#)

## TimerNum

0 to 1

## Frequency

0.112 Hz to 24 MHz

## Duty cycle

0 to 1, non-inclusive

## PulseCount

Ignored

## Triggering

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

## TrigType

TRIGABOVE, TRIGBELOW, TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE

- Digital triggering (TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE) is not supported for pre-trigger acquisitions ([cbAPretrig\(\)](#) function).
- Analog triggering (TRIGABOVE, TRIGBELOW) is not supported for the [cbDInScan\(\)](#) function and the [cbCInScan\(\)](#) function.

## Threshold

Analog hardware triggering, 12-bit resolution: 0 to 4,095 (supported for [cbAInScan\(\)](#) only)

Analog software triggering, 16-bit resolution: 0 to 65,535 (supported for [cbAPretrig\(\)](#) only)

## DAQ Input

UL: [cbDaqInScan\(\)](#)

UL for .NET: [DaqInScan\(\)](#)

## Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK, EXTTRIGGER, HIGHRESRATE

## ChanTypeArray

ANALOG, DIGITAL16, CTR16, CTR32LOW, CTR32HIGH

- When mixing ANALOG channel types with any other input types, the ANALOG channels should be the first in the list.

## ChanArray

ANALOG: 0 to 3

DIGITAL16: AUXPORT

CTR16: 0-3 counters

CTR32LOW: 0-3 counters

CTR32HIGH: 0-3 counters

SETPOINTSTATUS: 16-bit port that indicates the current state of the 16 possible setpoints.

ChanTypeArray flag value:

- SETPOINT\_ENABLE: Enables a setpoint. Refer to [Setpoints](#) in "Hardware Considerations" below for more information.

## Rate

Analog: Up to 1.33 MHz

Digital: Up to 8 MHz if no analog channel is selected. Otherwise up to 1.33 MHz.

Counter: Up to 8 MHz if no analog channel is selected. Otherwise up to 1.33 MHz.

## GainArray

ANALOG only; ignore for other ChanTypeArray values.

BIP10VOLTS ( $\pm 10$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

BIPPT5VOLTS ( $\pm 0.5$  volt)

## PretrigCount

100000 max

## DAQ Triggering

UL: [cbDaqSetTrigger\(\)](#)

UL for .NET: [DaqSetTrigger\(\)](#)

## TrigSource

TRIG\_IMMEDIATE, TRIG\_EXTTTL, TRIG\_ANALOGHW, TRIG\_ANALOGSW, TRIG\_DIGPATTERN, TRIG\_COUNTER, TRIG\_SCANCOUNT

### TrigSense

RISING\_EDGE, FALLING\_EDGE, ABOVE\_LEVEL, BELOW\_LEVEL, EQ\_LEVEL, NE\_LEVEL

### TrigEvent

START\_EVENT, STOP\_EVENT

## DAQ Setpoint

UL: [cbDagSetSetpoints\(\)](#)

UL for .NET: [DagSetSetpoints\(\)](#)

### SetpointFlagsArray

SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA, SF\_GREATERTHAN\_LIMITB, SF\_OUTSIDE\_LIMITS, SF\_HYSTERESIS, SF\_UPDATEON\_TRUEONLY, SF\_UPDATEON\_TRUEANDFALSE

### SetpointOutputArray

SO\_NONE, SO\_DIGITALPORT

Also available for USB-1604HS-2AO:

SO\_DAC0, SO\_DAC1

### LimitAArray

Any value valid for the associated input channel.  
Ignored for SF\_GREATERTHAN\_LIMITB

### LimitBArray

Any value valid for the associated input channel and less than LimitA.  
Ignored for SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA

### Output#Array

For SetpointOutputArray = SO\_NONE: Ignored

For SetpointOutputArray = SO\_DIGITALPORT: 0 to 65,535

For SetpointOutputArray = SO\_DAC#: Voltage values between -10 and +10

### OutputMask#Array

For SetpointOutputArray = SO\_DIGITALPORT: 0 to 65,535

For SetpointOutputArray = all other values: Ignored

### SetpointCount

0 (to disable setpoints) to 16

## DAQ Output (USB-1604HS-2AO only)

UL: [cbDagOutScan\(\)](#)

UL for .NET: [DagOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS, ADCCLOCKTRIG, ADCCLOCK

### ChanType

ANALOG, DIGITAL16

### ChanArray

ANALOG: 0 to 1

DIGITAL16: AUXPORT

### Rate

Analog: Up to 1 MHz

Digital16: Up to 8 MHz (system-dependent) if no analog channel is selected. Otherwise up to 1 MHz.

### Range

BIP10VOLTS ( $\pm 10$  volts)

## Hardware considerations

## Channel count

For input and output scans, the count must be set to an integer multiple of the number of channels in the scan or a [BADCOUNT](#) error is returned.

## Sampling and update rates

Sampling and update rates are system-dependent. Data overruns/underruns may occur with higher sampling rates when using BACKGROUND and CONTINUOUS modes. To avoid this, use a larger buffer/count size, or use NONSTREAMEDIO mode, if supported.

The minimum size buffer is 256 for [cbAOutScan\(\)](#). Values less than that result in a [BADBUFFERSIZE](#) error.

## Settling time

For most applications, settling time should be left at the default value of 1  $\mu$ s. However, if you are scanning multiple channels and one or more channels are connected to a high impedance source, you may get better results by increasing the settling time. Keep in mind that increasing the settling time reduces the maximum acquisition rate. You can set the time between A/D conversions with the ADC Settling Time option in InstaCal. Select between 1  $\mu$ s, 5  $\mu$ s, 10  $\mu$ s, or 1 ms.

## Setpoints

You enable setpoints with the SETPOINT\_ENABLE flag. This flag must be OR'ed with the ChanTypeArray argument values. You set the setpoint criteria with cbDaqSetSetpoints()/DaqSetSetpoints(). The number of channels set with the SETPOINT\_ENABLE flag must match the number of setpoints set by the SetpointCount argument (cbDaqSetSetpoints()/DaqSetSetpoints()).

## Output non-streamed data to a DAC output channel

With NONSTREAMEDIO mode, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. The FIFO holds 524288 samples. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample updates are transferred or downloaded to the device, the device is responsible for outputting the data. You can't change the output buffer once the output begins.

## Trigger DAC output operations with the ADC clock

Specify the ADCCLOCK option to trigger a data output operation upon the start of the ADC clock.

## Quadrature encoder operations

To configure a counter channel as a multi-axis quadrature encoder, use the [cbCConfigScan\(\)](#)/ [CConfigScan\(\)](#) Mode argument values to set a specified counter to encoder mode, set the encoder measurement mode to X1, X2, or X4, and then set the count to be latched either by the internal "start of scan" signal (default) or by the signal on the mapped channel.

You can optionally perform the following operations:

- Enable gating, so that the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.
- Enable "latch on Z" to latch counter outputs using the Encoder Z mapped signal.
- Enable "clear on Z" so that the counter is cleared on the rising edge of the mapped (Z) channel. By default, "clear on Z" is disabled, and the counter is not cleared.

## Asynchronous reads

The CConfigScan() method's Bit32 counter mode option only affects counter resolution for asynchronous calls ([CIn\(\)](#) and [CIn32\(\)](#)), and only when the counter is configured for StopAtMax.

This mode is recommended for use only with CIn32(). Using the Bit32 option with CIn() is not very useful, since the value returned by CIn() is only 16 bits. The effect is that the value returned by CIn() rolls over 65,535 times before stopping.

# USB-1608FS and USB-1608FS-Plus

The USB-1608FS and USB-1608FS-Plus support the following UL and UL for .NET features.

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

### Options

BACKGROUND, BLOCKIO\*, BURSTIO\*\*, CONTINUOUS, CONVERTDATA, EXTCLOCK, EXTTRIGGER, HIGHRESRATE\*\*\*, NOCALIBRATEDATA, and SINGLEIO\*

\* The packet size is based on the Options setting:

BLOCKIO: packet size = 31 samples

SINGLEIO: packet size = the number of channels being sampled.

\*\* BURSTIO can only be used with the number of samples (Count) set equal to the size of the FIFO or less. The device FIFO holds 32,768 samples. BURSTIO cannot be used with the CONTINUOUS option.

\*\*\* HIGHRESRATE is supported by the USB-1608FS-Plus only.

### Mode

Single-ended

### HighChan

0 to 7

### Count

In BURSTIO mode, Count *must* be an integer multiple of the number of channels in the scan:

- For one-, two-, four-, and eight-channel scans, the maximum Count is 32,768 samples.
- For three- and six-channel scans, the maximum Count is 32,766 samples.
- For five-channel scans, the maximum Count is 32,765 samples.
- For seven-channel scans, the maximum Count is 32,767 samples.

### Rate

USB-1608FS:

200 kS/s maximum for BURSTIO mode (50 kS/s for any one channel).

The maximum rate is 100 kS/s for all other modes (50 kS/s for any one channel).

When using [cbAInScan\(\)](#) or [AInScan\(\)](#), the minimum sample rate is 1 S/s. In BURSTIO mode, the minimum sample rate is 20 S/s per channel.

USB-1608FS-Plus:

800 kS/s maximum for BURSTIO mode (100 kS/s for any one channel).

For all other modes the rate is 400 kS/s nominal (100 kS/s for any one channel). Sampling at rates >400 kS/s may result in a data overrun on some systems. If errors occur, and the application requires fewer than 32,768 samples, consider a finite scan with BURSTIO mode enabled.

When using [cbAInScan\(\)](#) or [AInScan\(\)](#), the minimum sample rate is 0.01 S/s (using HIGHRESRATE with Rate set to 10.)

### Range

BIP10VOLTS (±10 volts)

BIP5VOLTS (±5 volts)

BIP2VOLTS (±2 volts)

BIP1VOLTS (±1 volts)

### Pacing

Hardware pacing, internal clock supported. External clock supported via the SYNC pin.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

Digital triggering: TRIGPOSEDGE, TRIGNEGEDGE

The USB-1608FS-Plus also supports TRIGHIGH and TRIGLOW

External digital (TTL) hardware triggering supported. Set the hardware trigger source with the Trig\_In input.

## Digital I/O

### Configuration

Functions

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

### Port I/O

Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#)

PortNum

AUXPORT (eight bits, bit-configurable)

DataValue

0 to 255 for AUXPORT

### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

### Count

$2^{32}-1$  when reading the counter.

### LoadValue

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* topic apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

### RegNum

LOADREG1

## Event Notification

## Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

## Event types

UL: ON\_SCAN\_ERROR, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

UL for .NET: OnScanError(), OnDataAvailable(), OnEndOfAiScan()

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on the USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware Considerations

### Channel-gain queue

The channel-gain queue is limited to eight elements. The gains may be any valid value. The channels specified must be unique and listed in increasing order. The gains may be any valid value.

- USB-1608FS: The channels specified in the queue must be contiguous and in increasing order, except when wrapping around from channel 7 to channel 0.
- USB-1608FS-Plus: The channels specified in the queue must be in increasing order.

### Acquisition rate

Since the maximum data acquisition rate depends on the system connected to the device, it is possible to "lose" data points when scanning at higher rates. If the requested speed cannot be sustained, an [OVERRUN](#) error will occur.

### HIGHRESRATE

USB-1608FS-Plus only. Specify the HIGHRESRATE scan option to acquire data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel".

### Continuous scans

When running [cbAInScan\(\)](#)/ [AInScan\(\)](#) with the CONTINUOUS option, consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, make the total number of samples an integer multiple of the packet size and the number of channels.

### EXTCLOCK

You can set the SYNC pin as a pacer input or a pacer output from InstaCal. By default, this pin is set for pacer input. If set for output when using the [cbAInScan\(\)](#)/[AInScan\(\)](#) option, EXTCLOCK results in a [BADOPTION](#) error.

### BURSTIO mode

BURSTIO mode allows higher sampling rates for sample counts up to the size of the FIFO. The device FIFO holds 32,768 samples. Data is collected into the device local FIFO. Data transfers to the computer don't occur until the scan completes. For BACKGROUND scans, the Count and Index returned by [cbGetStatus\(\)](#) and [GetStatus\(\)](#) remain 0, and Status = RUNNING until the scan finishes. The Count and Index are not updated until the scan is completed. When the scan is complete and the data is retrieved, [cbGetStatus\(\)](#) and [GetStatus\(\)](#) are updated to the current Count and Index, and Status = IDLE.

BURSTIO is required for aggregate Rate settings above 100 kHz, but Count is limited to sample counts up to the size of the FIFO (32,768 samples).

## USB-1608G Series

The USB-1608G Series includes the following devices:

- USB-1608G
- USB-1608GX
- USB-1608GX-2AO

The USB-1608G Series supports the following UL and UL for .NET features:

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

### Options

BACKGROUND, BLOCKIO, BURSTMODE, CONTINUOUS, EXTCLOCK, EXTTRIGGER, HIGHRESRATE, NOCALIBRATEDATA, RETRIGMODE, SCALEDATA, SINGLEIO

### Packet size

Rate dependent. The default packet size is 256 samples. At higher rates, the packet size increases by a multiple of 256.

### Mode

Single-ended and differential

### HighChan

0 to 15 in single-ended mode

0 to 7 in differential mode

### Count

Count must be an integer multiple of the number of channels in the scan.

### Rate

500 kS/s max

### Range

BIP10VOLTS ( $\pm 10$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP1VOLTS ( $\pm 1$  volts)

### Pacing

Hardware pacing, internal clock supported.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

Digital triggering: TRIGPOSEDGE, TRIGNEGEDGE, TRIGHIGH, TRIGLOW

External digital (TTL) hardware triggering supported. Set the hardware trigger source with the TRIG input.

## Analog output (USB-1608GX-2AO only)

### Functions

UL: [cbAOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [AOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER

### Range

BIP10VOLTS ( $\pm 10$  Volts)



## HighChan

0 to 1

## Count

Count must be an integer multiple of the number of channels in the scan.

## Rate

500 kS/s per channel, maximum

## Packet size

Rate dependent.

## Digital I/O

### Configuration

Functions

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AuxPort

PortType

AuxPort

### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AuxPort (eight bits, bit-configurable)

DataValue

0 to 255

### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortType

AuxPort

BitNum

0 to 7 on AuxPort

## Counter I/O

### Functions

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCLoad\(\)](#), [cbCLoad32\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CLoad\(\)](#), [CLoad32\(\)](#)

Note: Counters on these boards are zero-based (the first counter number is "0").

### CounterNum

0 to 1

### Count

$2^{32}$  when reading the counter.

### LoadValue

0 when loading the counter.

cbCLoad() and cbCLoad32() CLoad() and CLoad32() are only used to reset the counter to 0. No other values are valid.

The [Visual Basic signed integers](#) guidelines apply when using cbCIn() or CIn() for values greater than 32,767 and when using cbCIn32() or CIn32() for values greater than 2,147,483,647.

## RegNum

LOADREG0, LOADREG1

## Timer

### Functions

UL: [cbPulseOutStart\(\)](#), [cbPulseOutStop\(\)](#)

UL for .NET: [PulseOutStart\(\)](#), [PulseOutStop\(\)](#)

### TimerNum

0

### Frequency

0.0149 Hz to 32 MHz

### DutyCycle

0 to 1, non-inclusive

### InitialDelay

0 sec to 67.11 sec

### IdleState

IDLE\_LOW, IDLE\_HIGH

### PulseCount

0 to  $2^{32} - 1$  (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

Set to 0 to continuously generate pulses until PulseOutStop() is called.

## Configuration

### Functions

UL: [cbSetConfig\(\)](#)

### InfoType

BOARDINFO

### ConfigItem

BIADTRIGCOUNT, BIDACTRIGCOUNT

### Device number

0

## String Configuration

### Functions

UL: [cbGetConfigString\(\)](#), [cbSetConfigString\(\)](#)

### InfoType

BOARDINFO

### ConfigItem

BINODEID

### maxConfigLen

At least 64 for BINODEID

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#) [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#) [DisableEvent\(\)](#)

### Event types

UL: ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_SCAN\_ERROR

UL for .NET: OnDataAvailable, OnEndOfAiScan, OnScanError

The USB-1608GX-2AO also supports ON\_END\_OF\_AO\_SCAN/OnEndOfAoScan.

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a Measurement Computing USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware Considerations

### Channel gain queue

The channel-gain queue is limited to 16 elements. The channel gains may be any valid value. The channels can be listed in any order, and can include duplicate channels for sampling at different ranges.

### Retriggering

Use [RETRIGMODE](#) with [cbAInScan\(\)](#) to set up repetitive trigger events. Use the ConfigItem option BIADTRIGCOUNT with [cbSetConfig\(\)](#) to set the A/D trigger count.

When using RETRIGMODE, set the values for the Count argument ([cbAInScan\(\)](#)/[AInScan\(\)](#)) and the BIADTRIGCOUNT argument ([cbSetConfig\(\)](#)/[SetAdRetrigCount\(\)](#)) to an integer multiple of the number of channels in the scan. That way, the entire buffer, or the portion of the buffer defined by BIADTRIGCOUNT, will contain updated data.

### Device identifier

You can enter up to 64 characters for the value of the device's text identifier using the ConfigItem option BINODEID with [cbSetConfigString\(\)](#).

### Output scan restriction

You cannot access [cbSetTrigger\(\)](#)/ [SetTrigger\(\)](#) or call BINODEID while an analog output scan is in progress.

## USB-1608HS Series

The USB-1608HS Series includes the following devices:

- USB-1608HS
- USB-1608HS-2AO

The USB-1608HS Series supports the following UL and UL for .NET features.

## Analog Input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)\\*](#), [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)\\*](#), [ATrig\(\)](#), [FileAInScan\(\)](#)

\* The channel-gain queue is limited to eight elements. The channels specified in the queue must be contiguous and in increasing order, except when wrapping around from channel 7 to channel 0. The gains may be any valid value.

### Options

BACKGROUND, BLOCKIO, SINGLEIO, CONTINUOUS, EXTTRIGGER, CONVERTDATA, NOCALIBRATEDATA, RETRIGMODE, and EXTCLOCK

### Packet size

Rate dependent. The default packet size is 256 samples. At higher rates, the packet size increases by a multiple of 256.

### Mode

Single-ended and differential

### HighChan

0 to 7 in single-ended and differential mode

### Rate

250 kHz per channel

### Count

Count must be an integer multiple of the number of channels in the scan.

### Range

BIP10VOLTS ( $\pm 10$ volts)	BIP2VOLTS ( $\pm 2$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIP1VOLTS ( $\pm 1$ volts)

### Pacing

Hardware pacing, internal clock supported. External clock supported via the SYNC\_IN pin.

## Analog Output (USB-1608HS-2AO only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS

### HighChan

0 to 1

### Rate

70 kHz for one channel  
47 kHz for two channels

### Count

Count must be an integer multiple of the number of channels in the scan.

### Range

BIP10VOLTS ( $\pm 10$  volts)

### Packet size

512 samples

## Data Value

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Pacing

Hardware pacing, internal clock supported.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

Analog triggering: TRIGABOVE, TRIGBELOW

Digital triggering: TRIGPOSEDGE, TRIGNEGEDGE, TRIGHIGH, TRIGLOW

External digital (TTL) hardware triggering supported. Set the hardware trigger source with the Trig\_In input.

### Threshold

0 to 65,535 (BIP10VOLTS)

Hardware actually has a 12-bit resolution, but the library uses a 16-bit value so that [cbFromEngUnits\(\)](#) can be used to obtain the trigger value.

## Digital I/O

### Port I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#)

#### PortNum

AUXPORT

#### DataValue

0 to 255 for AUXPORT

### Bit I/O

#### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

#### PortType

AUXPORT

#### BitNum

0 to 7 on AUXPORT

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

### Count

$2^{32} - 1$  when reading the counter.

## LoadValue

0 when loading the counter.

cbCLoad() and cbCLoad32() / CLoad() and CLoad32() are only used to reset the counter to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* topic apply when using cbCIn() or CIn() for values greater than 32,767 and when using cbCIn32() or CIn32() for values greater than 2,147,483,647.

## RegNum

LOADREG1

## Configuration

### Functions

UL: [cbGetConfig\(\)](#), [cbSetConfig\(\)](#), [cbGetConfigString\(\)](#), [cbSetConfigString\(\)](#)

### ConfigItem

BIADTRIGCOUNT, BINODEID

### Device number

0

### maxConfigLen

At least 64 for BINODEID

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event types

UL: ON\_SCAN\_ERROR, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

UL for .NET: OnScanError, OnDataAvailable, OnEndOfAiScan

The USB-1208HS-2AO also supports ON\_END\_OF\_AO\_SCAN/OnEndOfAoScan

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a Measurement Computing USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware Considerations

### Acquisition rate

Since the maximum data acquisition rate depends on the system connected to the device, it is possible to "lose" data points when scanning at higher rates. If the requested speed cannot be sustained, an [OVERRUN](#) error will occur.

### Continuous scans

When running [cbAInScan\(\)/AInScan\(\)](#) with the CONTINUOUS option, make the count an integer multiple of the number of channels in the scan in order to keep the data aligned properly in the array.

### Device identifier

You can enter up to 64 characters for the value of the device's text identifier using the ConfigItem option BINODEID with [cbSetConfigString\(\)](#).

### Output scan restriction

You cannot access [cbSetTrigger\(\)/ SetTrigger\(\)](#) or call BINODEID while an analog output scan is in progress.

### Analog triggering

When using [cbAInScan\(\)/AInScan\(\)](#) with EXTTRIGGER, the value entered to [cbSetTrigger\(\)](#) threshold arguments for analog trigger modes should be a 16 bit value. The resolution of the circuitry is actually 12 bits, but the library uses a 16 bit value so that [cbFromEngUnits\(\)](#) can be used to obtain the trigger value.

### Retriggering

Use [RETRIGMODE](#) with cbAInScan() to set up repetitive trigger events. Use the ConfigItem option BIADTRIGCOUNT with

[cbSetConfig\(\)](#) to set the A/D trigger count, and the ConfigItem option BIDACTRIGCOUNT to set the D/A trigger count.

When using RETRIGMODE, set the values for the Count argument (cbAInScan()/AInScan()) and the BIADTRIGCOUNT argument ([cbSetConfig\(\)/SetAdRetrigCount\(\)](#)) to an integer multiple of the packet size (and the number of channels if using CONTINUOUS). That way, the entire buffer, or the portion of the buffer defined by BIADTRIGCOUNT, will contain updated data.

### **Remote sensing (USB-1608HS-2AO)**

You can enable remote sensing for each of the two analog outputs on the USB-1608HS-2AO with InstaCal.

The remote sensing feature compensates for the voltage drop error that occurs in applications where the USB-1608HS-2AO's analog outputs are connected to its load through a long wire or cable type interconnect.

The remote sensing feature can compensate for  $I \cdot R$  induced voltage losses up to 750 mV, and for any series resistance up to 75  $\Omega$  between its remote sensing terminal pins and its output load.

- To configure the remote sensing connection, connect two separate output wires – one from the VDACC<sub>n</sub>\_F (force) output terminal, and one from the VDACC<sub>n</sub>\_S (sense) output terminal – to the high side or positive input terminal of the field device (load).
- If you are not using the remote sensing feature, simply connect a single output wire or cable from the VDACC<sub>n</sub>\_F (force) output terminal to the load, and leave the VDACC<sub>n</sub>\_S (sense) terminal unconnected.

Refer to the *USB-1608HS-2AO User's Guide* for more information about remote sensing.

## USB-1616FS

The USB-1616FS supports the following UL and UL for .NET features.

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)\\*](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)\\*](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

\*The channel-gain queue is limited to 16 elements. The USB-1616FS accepts only unique contiguous channels in each element, but the gains may be any valid value.

### Options

BACKGROUND, BLOCKIO\*\*, SINGLEIO\*\*, BURSTIO\*\*\*, CONTINUOUS, EXTTRIGGER, and EXTCLOCK

\*\* The packet size is based on the Options setting. When set to BLOCKIO, the packet size is 62 samples. When set to SINGLEIO, the packet size equals the number of channels being sampled.

\*\*\* BURSTIO can only be used with the number of samples (Count) set equal to the size of the FIFO or less. The USB-1616FS FIFO holds 32,768 samples. Also, BURSTIO cannot be used with the CONTINUOUS option.

### HighChan

0 to 15 in single-ended mode

### Count

In BURSTIO mode, Count needs to be an integer multiple of the number of channels in the scan.

- For one-, two-, four-, eight-, and 16-channel scans, the maximum Count is 32,768 samples.
- For three- and six-channel scans, the maximum Count is 32,766 samples.
- For five-channel scans, the maximum Count is 32,765 samples.
- For seven-channel scans, the maximum Count is 32,767 samples.
- For 9-, 10-, 12-, 13-, 14-, and 15-channel scans, the maximum Count is 32,760 samples.
- For 11-channel scans, the maximum Count is 32,758 samples.

### Rate

200 kHz maximum for BURSTIO mode (50 kHz for any one channel).

For all other modes, the maximum rate per channel depends on the number of channels being scanned.

No. of channels in the scan	Maximum rate
1 or 2	50 kHz
3	36 kHz
4	30 kHz
5	25 kHz
6	22 kHz
7	19 kHz
8	17 kHz
9	15 kHz
10	14 kHz
11	12.5 kHz
12	12 kHz
13	11.25 kHz
14	10.5 kHz
15	10 kHz
16	9.5 kHz

When using [cbAInScan\(\)](#) or [AInScan\(\)](#), the minimum sample rate is 1 Hz. In BURSTIO mode, the minimum sample rate is 20 Hz/channel.



## Range

Single-ended:

BIP10VOLTS ( $\pm 10$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP1VOLTS ( $\pm 1$  volts)

## Pacing

Hardware pacing, internal clock supported.

External clock supported via the SYNC terminal.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE

TRIGNEGEDGE

External digital (TTL) hardware triggering supported. You set the hardware trigger source with the TRIG\_IN input terminal.

## Digital I/O

### Configuration

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

### Port I/O

UL: [cbDOut\(\)](#), [cbDIn\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#)

PortNum

AUXPORT

DataValue

0 to 255 for AUXPORT

### Bit I/O

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

## Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

cbCLoad() and cbCLoad32() / CLoad() and CLoad32() are only used to reset the counter for this board to 0. No other values are valid.

The [Basic signed integers](#) guidelines apply when using cbCIn() or CIn() for values greater than 32,767 and when using cbCIn32() or CIn32() for values greater than 2,147,483,647.

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event Types

ON\_SCAN\_ERROR (analog input), ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the USB LED on a Measurement Computing USB module to blink.

When you have several modules connected to the computer, use these functions to identify a particular module by making its LED blink.

## Hardware considerations

### Acquisition Rate

Since the maximum data acquisition rate depends on the system connected to the USB-1616FS, it is possible to "lose" data points when scanning at higher rates. The Universal Library cannot always detect this data loss.

Most systems can sustain rates of 80 kS/s aggregate. If you need to sample at higher rates than this, consider using the [BURSTIO](#) option explained below.

### EXTCLOCK

You can set the SYNC terminal as a pacer input or a pacer output from InstaCal. By default, this terminal is set for pacer input. If set for output, using the cbAInScan()/AInScan() option EXTCLOCK results in a BADOPTION error.

### BURSTIO

Allows higher sampling rates up to the size of the FIFO. The USB-1616FS FIFO holds 32,768 samples. Data is collected into the USB device's local FIFO. Data transfers to the PC don't occur until the scan completes. For BACKGROUND scans, the Count and Index returned by [cbGetStatus\(\)](#) and [GetStatus\(\)](#) remain 0, and STATUS=RUNNING until the scan finishes. The Count and Index are not updated until the scan is completed. When the scan is complete and the data is retrieved, cbGetStatus() and GetStatus() are updated to the current Count and Index, and STATUS=IDLE.

BURSTIO is required for aggregate Rate settings above 100 kHz, but Count is limited to sample counts up to the size of the FIFO (32,768 samples). Count settings must be an integer multiple of the number of channels in the scan (see [Count](#) above).

### Continuous scans

When running [cbAInScan\(\)/AInScan\(\)](#) with the CONTINUOUS option, you should consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, make the total number of samples an integer multiple of the packet size and the number of channels.

When running cbAInScan()/AInScan() with the CONTINUOUS option, you **must** set the count to an integer multiple of the packet size (62) and the number of channels in the scan.

## USB-1616HS Series

The USB-1616HS Series includes the following hardware:

- USB-1616HS
- USB-1616HS-2
- USB-1616HS-4

The USB-1616HS Series support the following UL and UL for .NET features.

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [>cbAPretrig\(\)](#)\*

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [APretrig\(\)](#)\*

\* Pretrigger capability is implemented in software. PretrigCount must be less than the TotalCount and cannot exceed 100000 samples. TotalCount must be greater than the PretrigCount. If a trigger occurs while the number of collected samples is less than the PretrigCount, that trigger will be ignored. Requires a call to [cbSetTrigger/SetTrigger](#) for the analog trigger type.

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, DMAIO, BLOCKIO, EXTTRIGGER\*

\* With EXTTRIGGER mode, the first channel in the scan is the analog trigger channel.

### HighChan

0 to 15 in single-ended mode (0 to 63 single-ended if the [AI-EXP48](#) expansion board is installed.)

0 to 7 in differential mode. (0 to 31 differential if the AI-EXP48 expansion board is installed.)

### Rate

Up to 1 MHz

### Range

BIP10VOLTS ( $\pm 10$ volts)	BIPPT5VOLTS ( $\pm 0.5$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIPPT2VOLTS ( $\pm 0.2$ volts)
BIP2VOLTS ( $\pm 2$ volts)	BIPPT1VOLTS ( $\pm 0.1$ volts)
BIP1VOLTS ( $\pm 1$ volts)	

## Analog output (USB-1616HS-4 and USB-1616HS-2 only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS, NONSTREAMEDIO, ADCCLOCKTRIG, ADCCLOCK

NONSTREAMEDIO can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

### HighChan

USB-1616HS-4: 0 to 3

USB-1616HS-2: 0 to 1

### Rate

1 MHz

### Range

Ignored - not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Pacing

Hardware pacing, external or internal clock supported.

## Digital I/O

► Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions.

### Configuration

#### Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

### Port I/O

#### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDInScan\(\)](#), [cbDOutScan\(\)](#)\*

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DInScan\(\)](#), [DOutScan\(\)](#)\*

\*FIRSTPORTA and FIRSTPORTB must be set for output to use this function. Refer to [DIO PortNum](#) in the *Hardware Considerations* section for more information.

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER, WORDXFER, NONSTREAMEDIO, ADCCLOCKTRIG, ADCCLOCK

- The EXTTRIGGER option can only be used with the [cbDInScan\(\)](#) function. You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).
- The WORDXFER option can only be used with FIRSTPORTA.
- The NONSTREAMEDIO, ADCCLOCKTRIG, and ADCCLOCK options can only be used with the [cbDOutScan\(\)](#) function.
- The NONSTREAMEDIO option can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

#### Rate

12 MHz

#### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

#### DataValue

0 to 255

0 to 65,535 using the WORDXFER option with FIRSTPORTA

### Bit I/O

#### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

#### PortType

FIRSTPORTA

#### BitNum

0 to 23

## Counter Input

### Functions

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCConfigScan\(\)](#), [cbCInScan\(\)](#), [cbCClear\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CConfigScan\(\)](#), [CInScan\(\)](#), [CClear\(\)](#)

Note: Counters on these boards are zero-based (the first counter number is "0").

#### Rate

6 MHz

#### CounterNum

0 to 3

#### Options

BACKGROUND, CONTINUOUS, EXTTRIGGER

You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) in the *Universal Library Description and Use* section for information on 16-bit values using unsigned integers.)

## Timer Output

### Functions

UL: [cbTimerOutStart\(\)](#), [cbTimerOutStop\(\)](#)

UL for .NET: [TimerOutStart\(\)](#), [TimerOutStop\(\)](#)

### TimerNum

0 to 1

### Frequency

15.260 Hz to 1.0 MHz

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGABOVE, TRIGBELOW, TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE

Digital triggering (TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE) is not supported for pre-trigger acquisitions ([cbAPretrig\(\)](#) function).

Analog triggering (TRIGABOVE, TRIGBELOW) is not supported for the [cbDInScan\(\)](#) function and the [cbCInScan\(\)](#) function.

### Threshold

Analog hardware triggering, 12-bit resolution: 0 to 4,095 (supported for [cbAInScan\(\)](#) only)

Analog software triggering, 16-bit resolution: 0 to 65,535 (supported for [cbAPretrig\(\)](#) only)

## Temperature Input

### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#), [cbGetTCValues\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#), [GetTCValues\(\)](#)

### Options

NOFILTER

### Scale

CELSIUS, FAHRENHEIT, KELVIN

### HighChan

0 to 7 (0 to 31 if the AI-EXP48 expansion board is installed.)

## DAQ Input

### Functions

UL: [cbDagInScan\(\)](#)

UL for .NET: [DagInScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, DMAIO, BLOCKIO, EXTTRIGGER

### ChanTypeArray

ANALOG, DIGITAL8, DIGITAL16, CTR16, CTR32LOW, CTR32HIGH, CJC, TC, SETPOINTSTATUS

When mixing the ANALOG channel type with any other input types, the ANALOG channels should be first in the list.

**Note:** For information on associating CJC channels with TC channels, refer to the [Associating CJC channels with TC channels](#) discussion in the *Hardware considerations* section below.

## ChanArray

ANALOG:

- 0 to 15 in single-ended mode
- 0 to 7 in differential mode (0 to 63 single-ended, 0 to 31 differential if the AI-EXP48 expansion board is installed.)

DIGITAL8: FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

DIGITAL16: FIRSTPORTA

CTR16: 0-3 counters

CTR32LOW: 0-3 counters

CTR32HIGH: 0-3 counters

CJC: 0 to 5 (0 to 11 if the AI-EXP48 is installed.)

TC: 0 to 7 (0 to 31 if the AI-EXP48 is installed.)

SETPOINTSTATUS: 16-bit port that indicates the current state of the 16 possible setpoints.

ChanTypeArray flag value:

- SETPOINT\_ENABLE: Enables a setpoint. Refer to the [Setpoints](#) discussion in the *Hardware considerations* section below for more information.

## Rate

Analog: Up to 1 MHz.

Digital: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

Counter: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

## GainArray

ANALOG only; ignore for other ChanTypeArray values.

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP1VOLTS ( $\pm 1$  volts)

BIPPT5VOLTS ( $\pm 0.5$  volts)

BIPPT2VOLTS ( $\pm 0.2$  volts)

BIPPT1VOLTS ( $\pm 0.1$  volts)

## PretrigCount

100000 max

## DAQ Triggering

### Functions

UL: [cbDaqSetTrigger\(\)](#)

UL for .NET: [DaqSetTrigger\(\)](#)

### TrigSource

TRIG\_IMMEDIATE, TRIG\_EXTTTL, TRIG\_ANALOGHW, TRIG\_ANALOGSW, TRIG\_DIGPATTERN, TRIG\_COUNTER, TRIG\_SCANCOUNT

### TrigSense

RISING\_EDGE, FALLING\_EDGE, ABOVE\_LEVEL, BELOW\_LEVEL, EQ\_LEVEL, NE\_LEVEL

### TrigEvent

START\_EVENT, STOP\_EVENT

## DAQ Setpoint

### Functions

UL: [cbDaqSetSetpoints\(\)](#)

UL for .NET: [DaqSetSetpoints\(\)](#)

### SetpointFlagsArray

SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA, SF\_GREATERTHAN\_LIMITB, SF\_OUTSIDE\_LIMITS, SF\_HYSTERESIS,

SF\_UPDATEON\_TRUEONLY, SF\_UPDATEON\_TRUEANDFALSE

### SetpointOutputArray

SO\_NONE, SO\_FIRSTPORTC, SO\_TMR0, SO\_TMR1

Also available for USB-1616HS-2:

SO\_DAC0, SO\_DAC1

Also available for USB-1616HS-4:

SO\_DAC0, SO\_DAC1, SO\_DAC2, SO\_DAC3

### LimitAArray

Any value valid for the associated input channel.

Ignored for SF\_GREATERTHAN\_LIMITB

### LimitBArray

Any value valid for the associated input channel and less than LimitA.

Ignored for SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA

### Output#Array

For SetpointOutputArray = SO\_NONE: Ignored

For SetpointOutputArray = SO\_FIRSTPORTC: 0 to 65,535

For SetpointOutputArray = SO\_TMR#: 0 (to disable the timer) or 15.26 to 1000000 (to set the output frequency)

For SetpointOutputArray = SO\_DAC#: Voltage values between -10 and +10

### OutputMask#Array

For SetpointOutputArray = SO\_FIRSTPORTC: 0 to 65,535

For SetpointOutputArray = all other values: Ignored

### SetpointCount

0 (to disable setpoints) to 16

## DAQ Output (USB-1616HS-4 and USB-1616HS-2 only)

### Functions

UL: [cbDagOutScan\(\)](#)

UL for .NET: [DagOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS, NONSTREAMEDIO, ADCCLOCKTRIG, ADCCLOCK

### ChanType

ANALOG, DIGITAL16

### ChanArray

ANALOG:

- USB-1616HS-4: 0 to 3
- USB-1616HS-2: 0 to 1

DIGITAL16:

- FIRSTPORTA (FIRSTPORTB must be configured as an output)

### Rate

ANALOG: Up to 1 MHz

DIGITAL16: Up to 12 MHz (system-dependent) if no analog channel is selected. Otherwise up to 1 MHz.

### Range

BIP10VOLTS ( $\pm 10$  volts)

## Hardware considerations

### Associating CJC channels with TC channels

The TC channels must immediately follow their associated CJC channels in the channel array. For accurate thermocouple readings, associate CJC channels with the TC channels as listed in the following table:

CJC channels	TC channels
--------------	-------------

CJC0	TC0
CJC1	TC1 and TC2
CJC2	TC3
CJC3	TC4
CJC4	TC5 and TC6
CJC5	TC7
<b>When the AI-EXP48 board is installed:</b>	
CJC6	TC8 through TC11
CJC7	TC12 through TC15
CJC8	TC16 through TC19
CJC9	TC20 through TC23
CJC10	TC24 through TC27
CJC11	TC28 through TC31

The board must be configured for differential inputs when using thermocouples. TC inputs are supported by differential mode configuration only.

### Channel count

For input and output scans, the count must be set to an integer multiple of the number of channels or a [BADCOUNT](#) error is returned.

### Sampling and update rates

Sampling and update rates are system-dependent. Data overruns/underruns may occur with higher sampling rates when using BACKGROUND and CONTINUOUS modes. To avoid this, use a larger buffer/count size, or use NONSTREAMEDIO mode, if supported.

The minimum size buffer is 256 for [cbAOutScan\(\)](#). Values less than that result in a [BADBUFFERSIZE](#) error.

### Settling time

For most applications, settling time should be left at the default value of 1  $\mu$ s. However, if you are scanning multiple channels and one or more channels are connected to a high impedance source, you may get better results by increasing the settling time. Keep in mind that increasing the settling time reduces the maximum acquisition rate. You can set the time between A/D conversions with the ADC Settling Time option in InstaCal. Select between 1  $\mu$ s, 5  $\mu$ s, 10  $\mu$ s, or 1 ms.

### Setpoints

You enable setpoints with the SETPOINT\_ENABLE flag. This flag must be OR'ed with the ChanTypeArray argument values. You set the setpoint criteria with the [cbDaqSetSetpoints\(\)/DaqSetSetpoints\(\)](#). The number of channels set with the SETPOINT\_ENABLE flag must match the number of setpoints set by the SetpointCount argument (cbDaqSetSetpoints()/DaqSetSetpoints()).

### Output non-streamed data to a DAC output channel

With NONSTREAMEDIO mode, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample updates are transferred or downloaded to the device, the device is responsible for outputting the data. You can't make any changes to the output buffer once the output begins.

### Trigger DAC output operations with the ADC clock

Specify the ADCCLOCK option to trigger a data output operation upon the start of the ADC clock.

### DIO PortNum

For cbDOutScan()/DOutScan() and cbDaqOutScan()/DaqOutScan(), FIRSTPORTA and FIRSTPORTB are treated as one 16-bit port. These functions can only be used with FIRSTPORTA. You must configure both FIRSTPORTA and FIRSTPORTB for output using the cbDConfigPort() function.

### Synchronous scanning with multiple boards

You can operate up to four USB-1616HS Series boards synchronously by setting the direction of the A/D and D/A pacer pins (**APR** or **DPR**) in InstaCal.

On the board used to pace each device, set the pacer pin that you want to use (APR or DPR) for *Output*. On the board(s) that you want to synchronize with this board, set the pacer pin that you want to use (APR or DPR) for *Input*.

You set the direction using the InstaCal configuration dialog's **APR Pin Direction** and **DPR Pin Direction** settings.

Wire the pacer pin configured for output to each of the pacer input pins that you want to synchronize.

### Quadrature encoder operations

To configure a counter channel as a multi-axis quadrature encoder, use the [cbCConfigScan\(\)/CConfigScan\(\)](#) Mode argument values to set a specified counter to encoder mode, set the encoder measurement mode to X1, X2, or X4, and then set the count to be latched either by the internal "start of scan" signal (default) or by the signal on the mapped channel.

You can optionally perform the following operations:

- Enable gating, so that the counter is enabled when the mapped channel to gate the counter is high. When the mapped



channel is low, the counter is disabled but holds the count value.

- Enable "latch on Z" to latch counter outputs using the Encoder Z mapped signal.
- Enable "clear on Z" so that the counter is cleared on the rising edge of the mapped (Z) channel. By default, "clear on Z" is disabled, and the counter is not cleared.

### **Asynchronous reads**

The CConfigScan() method's Bit32 counter mode option only affects counter resolution for asynchronous calls ([CIn\(\)](#) and [CIn32\(\)](#)), and only when the counter is configured for StopAtMax.

This mode is recommended for use only with CIn32(). Using the Bit32 option with CIn() is not very useful, since the value returned by CIn() is only 16 bits. The effect is that the value returned by CIn() rolls over 65,535 times before stopping.

## USB-1616HS-BNC

The USB-1616HS-BNC support the following UL and UL for .NET features.

### Analog input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbAPretrig\(\)](#)\*

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [APretrig\(\)](#)\*

\* Pretrigger capability is implemented in software. PretrigCount must be less than the TotalCount and cannot exceed 100000 samples. TotalCount must be greater than the PretrigCount. If a trigger occurs while the number of collected samples is less than the PretrigCount, that trigger will be ignored. Requires a call to [cbSetTrigger/SetTrigger](#) for the analog trigger type.

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, CONVERTDATA, DMAIO, BLOCKIO, EXTTRIGGER\*

\* With EXTTRIGGER mode, the first channel in the scan is the analog trigger channel.

#### HighChan

0 to 15 (only differential mode is available)

#### Rate

Up to 1 MHz

#### Range

BIP10VOLTS ( $\pm 10$ volts)	BIPPT5VOLTS ( $\pm 0.5$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIPPT2VOLTS ( $\pm 0.2$ volts)
BIP2VOLTS ( $\pm 2$ volts)	BIPPT1VOLTS ( $\pm 0.1$ volts)
BIP1VOLTS ( $\pm 1$ volts)	

### Analog output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS, NONSTREAMEDIO, ADCCLOCKTRIG, ADCCLOCK

NONSTREAMEDIO can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

#### HighChan

0 to 1

#### Rate

1 MHz

#### Range

Ignored - not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

#### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

#### Pacing

Hardware pacing, external or internal clock supported.

### Digital I/O

► Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions.

#### Configuration

##### Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

##### PortNum

FIRSTPORTA, FIRSTPORTB

## Port I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDInScan\(\)](#), [cbDOutScan\(\)](#)\*

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DInScan\(\)](#), [DOutScan\(\)](#)\*

\*FIRSTPORTA and FIRSTPORTB must be set for output to use this function. Refer to [DIO PortNum](#) in the *Hardware Considerations* section for more information.

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER, WORDXFER, NONSTREAMEDIO, ADCCLOCKTRIG, ADCCLOCK

- The EXTCLOCK option can only be used with [cbDInScan\(\)/DInScan\(\)](#).
- The EXTTRIGGER option can only be used with [cbDInScan\(\)/DInScan\(\)](#). You can use [cbSetTrigger\(\)](#) to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).
- The WORDXFER option can only be used with FIRSTPORTA.
- The NONSTREAMEDIO, ADCCLOCKTRIG, and ADCCLOCK options can only be used with [cbDOutScan\(\)/DOutScan\(\)](#).
- The NONSTREAMEDIO option can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

### Rate

12 MHz

### PortNum

FIRSTPORTA, FIRSTPORTB

### DataValue

0 to 255

0 to 65,535 using the WORDXFER option with FIRSTPORTA

## Bit I/O

### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortType

FIRSTPORTA

### BitNum

0 to 23

## Counter Input

### Functions

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCConfigScan\(\)](#), [cbCInScan\(\)](#), [cbCClear\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CConfigScan](#), [CInScan\(\)](#), [CClear\(\)](#)

Note: Counters on these boards are zero-based (the first counter number is "0").

### Rate

6 MHz

### CounterNum

0 to 3

### Options

BACKGROUND, CONTINUOUS, EXTTRIGGER

You can use [cbSetTrigger\(\)/SetTrigger\(\)](#) to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) in the *Universal Library Description and Use* section for information on 16-bit values using unsigned integers.)

## Timer Output

## Functions

UL: [cbTimerOutStart\(\)](#), [cbTimerOutStop\(\)](#)

UL for .NET: [TimerOutStart\(\)](#), [TimerOutStop\(\)](#)

## TimerNum

0 to 1

## Frequency

15.260 Hz to 1.0 MHz

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGABOVE, TRIGBELOW, TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE

Digital triggering (TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE) is not supported for pre-trigger acquisitions ([cbAPretrig\(\)](#) function).

Analog triggering (TRIGABOVE, TRIGBELOW) is not supported for [cbDInScan\(\)](#) and [cbCInScan\(\)](#).

### Threshold

Analog hardware triggering, 12-bit resolution: 0 to 4,095 (supported for [cbAInScan\(\)](#)/[AInScan](#) only).

Analog software triggering, 16-bit resolution: 0 to 65,535 (supported for [cbAPretrig\(\)](#)/[APreTrig\(\)](#) only)

## DAQ Input

### Functions

UL: [cbDaqInScan\(\)](#)

UL for .NET: [DaqInScan\(\)](#)

### Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK, EXTTRIGGER

### ChanTypeArray

ANALOG, DIGITAL8, DIGITAL16, CTR16, CTR32LOW, CTR32HIGH, CJC, TC, SETPOINTSTATUS

When mixing the ANALOG channel type with any other input types, the ANALOG channels should be first in the list.

### ChanArray

ANALOG:

- 0 to 15 (only differential mode available)

DIGITAL8: FIRSTPORTA, FIRSTPORTB

DIGITAL16: FIRSTPORTA

CTR16: 0-3 counters

CTR32LOW: 0-3 counters

CTR32HIGH: 0-3 counters

CJC: 0 to 5 (0 to 11 if the AI-EXP48 is installed.)

TC: 0 to 7 (0 to 31 if the AI-EXP48 is installed.)

SETPOINTSTATUS: 16-bit port that indicates the current state of the 16 possible setpoints.

ChanTypeArray flag value:

- SETPOINT\_ENABLE: Enables a setpoint. Refer to the [Setpoints](#) discussion in the *Hardware considerations* section below for more information.

### Rate

Analog: Up to 1 MHz.

Digital: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

Counter: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

### GainArray

ANALOG only; ignore for other ChanTypeArray values.

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP1VOLTS ( $\pm 1$  volts)

BIPPT5VOLTS ( $\pm 0.5$  volts)

BIPPT2VOLTS ( $\pm 0.2$  volts)

BIPPT1VOLTS ( $\pm 0.1$  volts)

### **PretrigCount**

100000 max

## **DAQ Triggering**

### **Functions**

UL: [cbDagSetTrigger\(\)](#)

UL for .NET: [DagSetTrigger\(\)](#)

### **TrigSource**

TRIG\_IMMEDIATE, TRIG\_EXTTTL, TRIG\_ANALOGHW, TRIG\_ANALOGSW, TRIG\_DIGPATTERN, TRIG\_COUNTER, TRIG\_SCANCOUNT

### **TrigSense**

RISING\_EDGE, FALLING\_EDGE, ABOVE\_LEVEL, BELOW\_LEVEL, EQ\_LEVEL, NE\_LEVEL

### **TrigEvent**

START\_EVENT, STOP\_EVENT

## **DAQ Setpoint**

### **Functions**

UL: [cbDagSetSetpoints\(\)](#)

UL for .NET: [DagSetSetpoints\(\)](#)

### **SetpointFlagsArray**

SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA, SF\_GREATERTHAN\_LIMITB, SF\_OUTSIDE\_LIMITS, SF\_HYSTERESIS, SF\_UPDATEON\_TRUEONLY, SF\_UPDATEON\_TRUEANDFALSE

### **SetpointOutputArray**

SO\_NONE, SO\_FIRSTPORTC, SO\_TMR0, SO\_TMR1, SO\_DAC0, SO\_DAC1

### **LimitAArray**

Any value valid for the associated input channel.

Ignored for SF\_GREATERTHAN\_LIMITB

### **LimitBArray**

Any value valid for the associated input channel and less than LimitA.

Ignored for SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA

### **Output#Array**

For SetpointOutputArray = SO\_NONE: Ignored

For SetpointOutputArray = SO\_TMR#: 0 (to disable the timer) or 15.26 to 1000000 (to set the output frequency)

For SetpointOutputArray = SO\_DAC#: Voltage values between -10 and +10

### **OutputMask#Array**

Ignored

### **SetpointCount**

0 (to disable setpoints) to 16

## **DAQ Output (USB-1616HS-4 and USB-1616HS-2 only)**

### **Functions**

UL: [cbDagOutScan\(\)](#)

UL for .NET: [DagOutScan\(\)](#)

## Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, NONSTREAMEDIO, SIMULTANEOUS

## ChanType

ANALOG, DIGITAL16

## ChanArray

ANALOG: 0 to 1

DIGITAL16: FIRSTPORTA (FIRSTPORTB must be configured as an output)

## Rate

ANALOG: Up to 1 MHz

DIGITAL16: Up to 12 MHz (system-dependent) if no analog channel is selected. Otherwise up to 1 MHz.

## Range

BIP10VOLTS ( $\pm 10$  volts)

# Hardware considerations

## Channel count

For input and output scans, the count must be set to an integer multiple of the number of channels or a [BADCOUNT](#) error is returned.

## Sampling and update rates

Sampling and update rates are system-dependent. Data overruns/underruns may occur with higher sampling rates when using BACKGROUND and CONTINUOUS modes. To avoid this, use a larger buffer/count size, or use NONSTREAMEDIO mode, if supported.

The minimum size buffer is 256 for cbAOutScan()/AOutScan(). Values less than that result in a [BADBUFFERSIZE](#) error.

## Settling time

For most applications, settling time should be left at the default value of 1  $\mu$ s. However, if you are scanning multiple channels and one or more channels are connected to a high impedance source, you may get better results by increasing the settling time. Keep in mind that increasing the settling time reduces the maximum acquisition rate. You can set the time between A/D conversions with the ADC Settling Time option in InstaCal. Select between 1  $\mu$ s, 5  $\mu$ s, 10  $\mu$ s, or 1 ms.

## Setpoints

You enable setpoints with the SETPOINT\_ENABLE flag. This flag must be OR'ed with the ChanTypeArray argument values. Set the setpoint criteria with cbDaqSetSetpoints()/DaqSetSetpoints(). The number of channels set with the SETPOINT\_ENABLE flag must match the number of setpoints set by the SetpointCount argument (cbDaqSetSetpoints()/DaqSetSetpoints()).

## Output non-streamed data to a DAC output channel

With NONSTREAMEDIO mode, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. The FIFO holds 524288 samples. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample updates are transferred or downloaded to the device, the device is responsible for outputting the data. You can't make any changes to the output buffer once the output begins.

## Trigger DAC output operations with the ADC clock

Specify the ADCCLOCK option to trigger a data output operation upon the start of the ADC clock.

## DIO PortNum

For cbDOutScan()/DOutScan() and cbDaqOutScan()/DaqOutScan(), FIRSTPORTA and FIRSTPORTB are treated as one 16-bit port. These functions can only be used with FIRSTPORTA. You must configure both FIRSTPORTA and FIRSTPORTB for output using cbDConfigPort()/DConfigPort().

## Quadrature encoder operations

To configure a counter channel as a multi-axis quadrature encoder, use the cbCConfigScan()/CConfigScan() Mode argument values to set a specified counter to encoder mode, set the encoder measurement mode to X1, X2, or X4, and then set the count to be latched either by the internal "start of scan" signal (default) or by the signal on the mapped channel.

You can optionally perform the following operations:

- Enable gating, so that the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.
- Enable "latch on Z" to latch counter outputs using the Encoder Z mapped signal.
- Enable "clear on Z" so that the counter is cleared on the rising edge of the mapped (Z) channel. By default, "clear on Z" is disabled, and the counter is not cleared.

## USB-204, USB-201

The USB-204 and USB-201 supports the following UL and UL for .NET features.

### Analog input

#### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

#### Options

BACKGROUND, BLOCKIO\*, CONTINUOUS, EXTCLOCK, EXTTRIGGER, RETRIGMODE\*\*, NOCALIBRATEDATA, SINGLEIO

\*The **USB-204** packet size is based on the Options setting:

BLOCKIO: 31

SINGLEIO: 1

\*\*RETRIGMODE can only be used with [cbAInScan\(\)](#)/[AInScan\(\)](#).

#### HighChan

0 to 7 in single-ended mode

0 to 3 in differential mode

#### Count

In CONTINUOUS mode, Count must be an integer multiple of the packet size.

#### Rate

USB-204: 1 MS/s

USB-201: 100 kS/s

The throughput depends on the system being used. Most systems can achieve 40 kHz aggregate.

When using [cbAInScan\(\)](#)/[AInScan\(\)](#), the minimum sample rate is 1 Hz.

#### Range

BIP10VOLTS ( $\pm 10$  volts)

#### Pacing

Hardware pacing, internal clock supported.

External clock supported via the PACER\_IN pin.

### Triggering

#### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

#### TrigType

TRIGPOSEDGE

TRIGNEGEDGE

Both products support external digital (TTL) hardware triggering. Use the DTRIG input (pin # 18 on the screw terminal) for the external trigger signal.

### Digital I/O

► Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Configuration Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

#### Port I/O Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

## Bit I/O Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 15 for on FIRSTPORTA

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although cbCIn()/CIn() are valid for use with this counter, cbCIn32()/CIn32() may be more appropriate, since the values returned may be greater than the data types used by cbCIn() and CIn() can handle.

\*\*cbCLoad(), cbCLoad32(), CLoad() and CLoad32() only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

### Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

cbCLoad()/CLoad() and cbCLoad32()/CLoad32() are only used to reset the counter for this device to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* section apply when using cbCIn() or CIn() for values greater than 32,767 and when using cbCIn32() or CIn32() for values greater than 2,147,483,647.

### RegNum

LOADREG1

## Event notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event types

ON\_SCAN\_ERROR (analog input and analog output), ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_END\_OF\_AO\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### Acquisition rate

Since the maximum data acquisition rate depends on the system connected to the device, it is possible to "lose" data points when scanning at higher rates. The Universal Library cannot always detect this data loss.

Most systems can sustain rates of 40 kS/s aggregate in BLOCKIO mode, and 1 kS/s aggregate in SINGLEIO mode.



## EXTCLOCK

By default, the SYNC pin is configured for pacer output and provides the internal pacer A/D clock signal. To configure the pin for pacer input, use the EXTCLOCK option.

If you use the EXTCLOCK option, make sure that you disconnect from the external clock source when you test or calibrate the device with InstaCal, as the SYNC pin drives the output.

## Repetitive trigger events

Use [RETRIGMODE](#) with `cbAInScan()` to set up repetitive trigger events. Use the ConfigItem option BIADTRIGCOUNT with [cbSetConfig\(\)](#) to set the A/D trigger count, and the ConfigItem option BIDACTRIGCOUNT to set the D/A trigger count.

## Resolution

When configured for single-ended mode, the resolution of the data is 11 bits (data values between 0 and 2,047). However, the Universal Library maps this data to 12-bit values, so the range of data is no different from the differential configuration. Consequently, the data returned contains only even numbers between 0 and 4,094 when the NOCALIBRATEDATA option is used.

## Continuous scans

When running [cbAInScan\(\)](#)/ [AInScan\(\)](#) with the CONTINUOUS option, consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, set the total number of samples to be an integer multiple of the packet size and the number of channels in the scan.

## Channel-gain queue

When using `cbALoadQueue()`/`ALoadQueue()`, the channel gain queue is limited to 16 elements.

The queue accepts any combination of valid channels and gains in each element.

## USB-2404-60, USB-2404-10

The USB-2404-60 and USB-2404-10 support the following UL and UL for .NET features:

### Analog input

#### Functions

UL: [cbAIn32\(\)](#), [cbVIn32\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)\\*](#)

UL for .NET: [AIn32\(\)](#), [VIn32\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)\\*](#)

\*The queue can contain up to four elements of unique channel values. The gain elements are ignored.

#### Options

BACKGROUND, CONTINUOUS, BLOCKIO, NOCALIBRATEDATA, SCALEDATA

#### HighChan

0 to 3

#### Count

When using input scanning, the Count must be an integer multiple of the number of channels in the scan.

#### Rate

From 1.612 kS/s up to 50 kS/s per channel

#### Range

USB-2404-60: Ignored - not programmable; fixed at BIP60VOLTS ( $\pm 60$  volts)

USB-2404-10: Ignored - not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

### Event Notification

#### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

#### Event types

UL: ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_SCAN\_ERROR

UL for .NET: OnDataAvailable, OnEndOfAiScan, OnScanError

### Hardware considerations

#### Scan rate

When scanning at low rates, make sure you account for the granularity of the rate at low speeds by checking the returned rate value.

If you enter a scan rate between 1 kS/s and <1.612 kS/s, the rate is converted to 1.612 kS/s.

#### Retrieving data from the Windows buffer

Since these are high resolution devices, use [cbWinBufAlloc32\(\)/WinBufAlloc32\(\)](#) to specify the buffer size when scanning data. To retrieve data from the buffer, call [cbWinBufToArray32\(\)/WinBufToArray32\(\)](#).

## USB-2404-UI

The USB-2404-UI supports the following UL and UL for .NET features:

### Analog input

#### Functions

UL: [cbAIn32\(\)](#), [cbVIn32\(\)](#), [cbAInScan\(\)](#)

UL for .NET: [AIn32\(\)](#), [VIn32\(\)](#), [AInScan\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, BLOCKIO, HIGHRESRATE, SCALEDATA

#### HighChan

0 to 3

#### Count

When using input scanning, Count must be an integer multiple of the number of channels in the scan.

#### Rate

No channels configured in TC mode:

- High speed mode: 100 Hz
- Best 60 Hz rejection: 9.09 Hz
- Best 50 Hz rejection: 7.69 Hz
- High resolution: 2 Hz

One or more channels configured in TC mode:

- High speed mode: 50 Hz
- Best 60 Hz rejection: 8.33 Hz
- Best 50 Hz rejection: 7.14 Hz
- High resolution: 1.96 Hz

#### Range

##### Voltage mode:

BIP60VOLTS ( $\pm 60$  volts)

BIP15VOLTS ( $\pm 15$  volts)

BIP4VOLTS ( $\pm 4$  volts)

BIP1VOLTS ( $\pm 1$  volts)

BIPPT125VOLTS ( $\pm 0.125$  volts)

The Range argument is ignored when channels are configured for other input modes.

### Event Notification

#### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

#### Event types

UL: ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_SCAN\_ERROR

UL for .NET: OnDataAvailable, OnEndOfAiScan, OnScanError

### Miscellaneous

#### Functions

UL: [cbTEDSRead\(\)](#)

UL for .NET: [TEDSRead\(\)](#)

Reads data from a TEDS sensor into an array.

## Hardware considerations

### **VIn32()**

Only use `cbVIn32()/VIn32()` when the channel type is set to Voltage. Using this method with other channel types will return a `BadADChannel` error.

### **HIGHRESRATE**

Specify this scan option with `cbAInScan()/AInScan` to acquire data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel".

### **Retrieving data from the Windows buffer**

Since the USB-2404-UI is a high resolution device, use [cbWinBufAlloc32\(\)/WinBufAlloc32\(\)](#) to specify the buffer size when scanning data. To retrieve data from the buffer, call [cbWinBufToArray32\(\)/WinBufToArray32\(\)](#).

## USB-2408 Series

The USB-2408 Series includes the following devices:

- USB-2408
- USB-2408-2AO

The USB-2408 Series supports the following UL and UL for .NET features.

## Analog voltage input

### Functions

UL: [cbAIn32\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbATrig\(\)](#), [cbVIn\(\)](#), [cbVIn32\(\)](#)

UL for .NET: [AIn32\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [ATrig\(\)](#), [VIn\(\)](#), [VIn32\(\)](#)

### Options

BACKGROUND, BLOCKIO, CONTINUOUS, HIGHRESRATE, NOCALIBRATEDATA, SCALEDATA, SINGLEIO

### HighChan

0 to 15

While the entire range of channels listed above is always available, one channel is effectively lost for each channel configured for differential mode. Refer to [Analog input mode](#) in the *Hardware Considerations* section below for more information.

### Rate

0 to 1,102 S/s with the data rate set at maximum (3,750 Hz).

The maximum scan rate depends upon the data rate set for the channel(s) in the scan. Refer to [Maximum scan rate](#) in the *Hardware considerations* section below for more information.

When the Rate argument is set to 0, the scan will run at the maximum permissible rate according to the data rate set for each channel.

### Range

#### Voltage mode

BIP10VOLTS ( $\pm 10$ volts)	BIPPT625VOLTS ( $\pm 0.625$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIPPT312VOLTS ( $\pm 0.3125$ volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	BIPPT156VOLTS ( $\pm 0.15625$ volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	BIPPT078VOLTS ( $\pm 0.078125$ volts)

#### Thermocouple mode

BIPPT078VOLTS ( $\pm 0.078125$  volts)

Analog voltage input functions used on any analog input channel configured for thermocouple inputs will either return a voltage using the BIPPT078VOLTS ( $\pm 0.078125$  volts) scale or, when using [cbAIn32\(\)](#)/[AIn32\(\)](#), will return an error. [cbALoadQueue\(\)](#)/[ALoadQueue\(\)](#) will also return an error if a range other than BIPPT078VOLTS is applied to a thermocouple channel.

## Temperature input

### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)\*

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)\*

\* Only channels that are configured as thermocouples will be converted to temperature. Other channels included in [cbTInScan\(\)](#)/[TInScan\(\)](#) will be converted to voltage using the BIPPT078VOLTS ( $\pm 0.078125$  volts) range.

### Scale

CELSIUS, FAHRENHEIT, KELVIN, NOSCALE\*

\* Refer to [NOSCALE](#) in the *Hardware considerations* section below for more information on this option. This parameter has no effect on channels configured for voltage input.

### HighChan

0 to 7

## Analog voltage output (USB-2408-2AO only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

## Options

BACKGROUND, CONTINUOUS

## HighChan

0 to 1

## Rate

1 kHz, aggregate

## Range

BIP10VOLTS ( $\pm 10$  volts)

## Packet size

32 samples

## Data value

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Pacing

Hardware pacing, internal clock supported.

# Digital I/O

## Port I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

### PortNum

FIRSTPORTA

### DataValue

0 to 255

## Bit I/O

### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortType

FIRSTPORTA

### BitNum

0 to 7

# Counter input

## Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

## CounterNum

0 to 1

## Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter for this board to 0. No other values are valid.

The [Visual Basic signed integers](#) guidelines apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using

[cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

## Configuration

### Functions

UL: [cbGetConfig\(\)](#), [cbSetConfig\(\)](#)

### InfoType

BOARDINFO

### ConfigItem

BIFACTORYID

## String configuration

### Functions

UL: [cbGetConfigString\(\)](#), [cbSetConfigString\(\)](#)

### InfoType

BOARDINFO

### ConfigItem

BINODEID, BIFACTORYID\*

\* BIFACTORYID is read-only, and is not supported by [cbSetConfigString\(\)](#)/[SetConfigString\(\)](#).

### maxConfigLen

At least 64 for BINODEID

## Event notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event Types

UL: ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_SCAN\_ERROR

UL for .NET: OnDataAvailable, OnEndOfAiScan, OnScanError

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a Measurement Computing USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### Analog input mode

The USB-2408 and USB-2408-2AO have 16 analog inputs that are individually software configurable as single-ended or differential. Although all 16 channels are always available to the Universal Library, for each channel configured as differential you essentially lose one single-ended channel. Reading channels 8 through 15 will return a single-ended result for those channels, but if the associated channel is configured as differential, doing so would not be practical in most applications. For example, if channel 0 is configured as differential, reading channel 8 is valid, but of limited practical usefulness, since the connection would be one-half of a differential signal.

### Scanning both voltage and thermocouple inputs

Voltage and TC inputs can be mixed in the scan operation. Voltage inputs can be either single-ended or differential; TC inputs must be differential. The [AInScan\(\)](#) operation creates an array of raw data in 24-bit counts. When scanning both voltage and TC inputs, allocate the buffer with [ScaledWinBuffAlloc\(\)](#), and allocate the array as type Double. Call [AInScan\(\)](#)'s [ScaleData](#) option to convert the raw scan data to either voltage or temperature. Call [ScaledWinBufToArray\(\)](#) to copy the double-precision values from the buffer to the array.

### NOSCALE

Specify the NOSCALE option to retrieve raw data from the device. When NOSCALE is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.

### Maximum scan rate

The actual maximum scan rate depends upon the data rate set for the channel(s) in the scan. For  $n$  channels whose data rates are  $f_1, f_2 \dots f_n$ , the maximum scan rate  $f_{\max}$  can be calculated from the following equation:

$$1/f_{\max} = (640 \mu\text{sec} + 1/f_1) + (640 \mu\text{sec} + 1/f_2) + (640 \mu\text{sec} + 1/f_n)$$

Refer to the *Specifications* chapter in the hardware user's guide for the maximum throughput rate calculated for each selectable A/D data rate per number of input channels.

### Scanning multiple channels at low data rates

When scanning multiple channels at low data rates, a BADRATE error may be returned if the Rate parameter is set too high for the number of channels in the scan. For example, when scanning multiple channels with the data rate set to 2.5 Hz, the maximum number of input channels using an integer scan rate is two, since the Universal Library doesn't accept non-integer rates.

To resolve this limitation, set the Rate parameter to 0. The Library will calculate the fastest allowable rate using the maximum scan rate equation, and run the scan at the calculated rate. The value returned to the Rate parameter will be the calculated value rounded to the nearest integer. Note that the Rate parameter may return 0, even though the fastest allowable rate was used for the conversion.

Refer to the *Specifications* chapter in the hardware user's guide for the maximum scan rate calculated for each selectable A/D data rate per number of input channels.

### HIGHRESRATE

Specify the HIGHRESRATE scan option with `cbAInScan()/AInScan` to acquire data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel".

### Channel count

For output scans, the count must be set to an integer multiple of the number of channels or a [BADCOUNT](#) error is returned.

### Digital I/O channels

The state of each digital line can be read whether it is being used for output or for input. Each digital channel is an open-drain, and can either be driven low or allowed to float. If the digital line is intended to be used as an input, make sure that line is not driven low. A digital line is driven low by writing a logical "1" to the bit associated with the line. Implied in this is that the output function "inverts", so if all lines are driven low by writing a value of 255 to the port, the value read back will be 0.

### Device identifier

You can enter up to 64 characters for the value of the device's text identifier using the ConfigItem option BINODEID with `cbSetConfigString()`.



## USB-2416 Series

The USB-2416 Series includes the following devices:

- USB-2416
- USB-2416-4AO

The USB-2416 Series support the following UL and UL for .NET features.

## Analog voltage input

### Functions

UL: [cbAIn32\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbVIn\(\)](#), [cbVIn32\(\)](#)

UL for .NET: [AIn32\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [VIn\(\)](#), [VIn32\(\)](#)

### Options

BACKGROUND, BLOCKIO, CONTINUOUS, HIGHRESRATE, NOCALIBRATEDATA, SCALEDATA, SINGLEIO

### HighChan

0 to 31\*

0 to 63\* when the [AI-EXP32](#) expansion board is installed.

\* While the entire range of channels listed above is always available, one channel is effectively lost for each channel configured for differential mode. Refer to [Analog input mode](#) in the *Hardware Considerations* section below for more information.

### Rate

0 to 1,102 S/s with the data rate set at maximum (3,750 Hz).

The maximum scan rate depends upon the data rate set for the channel(s) in the scan. Refer to [Maximum scan rate](#) in the *Hardware considerations* section below for more information.

When the Rate argument is set to 0, the scan will run at the maximum permissible rate according to the data rate set for each channel.

### Range

#### Voltage mode

BIP20VOLTS ( $\pm 20$ volts)	BIPPT625VOLTS ( $\pm 0.625$ volts)
BIP10VOLTS ( $\pm 10$ volts)	BIPPT312VOLTS ( $\pm 0.3125$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIPPT156VOLTS ( $\pm 0.15625$ volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	BIPPT078VOLTS ( $\pm 0.078125$ volts)
BIP1PT25VOLTS ( $\pm 1.25$ volts)	

#### Thermocouple mode

BIPPT078VOLTS ( $\pm 0.078125$  volts)

Analog voltage input functions used on any analog input channel configured for thermocouple inputs will either return a voltage using the BIPPT078VOLTS ( $\pm 0.078125$  volts) scale or, in the case of [cbAIn32\(\)](#)/[AIn32\(\)](#), will return an error. [cbALoadQueue\(\)](#)/[ALoadQueue\(\)](#) will also return an error if a range other than BIPPT078VOLTS is applied to a thermocouple channel.

## Temperature input

### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)\*

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)\*

\* Only channels that are configured as thermocouples will be converted to temperature. Other channels included in [cbTInScan\(\)](#)/[TInScan\(\)](#) will be converted to voltage using the BIPPT078VOLTS ( $\pm 0.078125$  volts) range.

### Scale

CELSIUS, FAHRENHEIT, KELVIN, NOSCALE\*

\* Refer to [NOSCALE](#) in the *Hardware considerations* section below for more information on this option. This parameter has no effect on channels configured for voltage input.

### HighChan

0 to 15 (0 to 31 if the AI-EXP32 expansion board is installed.)

## Analog voltage output (USB-2416-4AO only)

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

## Options

BACKGROUND, CONTINUOUS

## HighChan

0 to 3

## Rate

1 kHz, aggregate

## Range

Ignored; fixed at BIP10VOLTS ( $\pm 10$  volts)

## Packet size

32 samples

## Data value

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Pacing

Hardware pacing, internal clock supported.

## Digital I/O

### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA (FIRSTPORTA, FIRSTPORTB and FIRSTPORTC when the AI-EXP32 expansion board is installed.)

DataValue

0 to 255

### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 7 (0-23 when the AI-EXP32 expansion board is installed.)

## Counter input

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

0 to 1

### Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter for this board to 0. No other values are valid.

The [Basic signed integers](#) guidelines apply when using `cbCIn()` or `CIn()` for values greater than 32,767 and when using `cbCIn32()` or `CIn32()` for values greater than 2,147,483,647.

## Configuration

### Functions

UL: [cbGetConfig\(\)](#), [cbSetConfig\(\)](#)

### InfoType

BOARDINFO

## String configuration

### Functions

UL: [cbGetConfigString\(\)](#), [cbSetConfigString\(\)](#)

### InfoType

BOARDINFO

### ConfigItem

BINODEID, BIFACTORYID\*

\* BIFACTORYID is read-only, and is not supported by `cbSetConfigString()/SetConfigString()`.

### maxConfigLen

At least 64 for BINODEID

## Event notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event Types

UL: ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_SCAN\_ERROR

UL for .NET: OnDataAvailable, OnEndOfAiScan, OnScanError

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a Measurement Computing USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### Analog input mode

The USB-2416 Series has 32 analog inputs that are individually software configurable as single-ended or differential. Although all 32 channels are always available to the Universal Library, for each channel configured as differential you essentially lose one single-ended channel. Reading channels 16 through 31 will return a single-ended result for those channels, but if the associated channel is configured as differential, doing so would not be practical in most applications. For example, if channel 0 is configured as differential, reading channel 16 is valid, but of limited practical usefulness, since the connection would be one-half of a differential signal. Additional channels can be added through connection to the AI-EXP32 expansion board.

TC inputs are supported by differential configuration only.

### Scanning both voltage and thermocouple inputs

Voltage and TC inputs can be mixed in the scan operation. Voltage inputs can be either single-ended or differential; TC inputs must be differential. The `AIInScan()` operation creates an array of raw data in 24-bit counts. When scanning both voltage and TC inputs, allocate the buffer with `ScaledWinBuffAlloc()`, and allocate the array as type `Double`. Call `AIInScan()`'s `ScaleData` option to convert the raw scan data to either voltage or temperature. Call `ScaledWinBufToArray()` to copy the double-precision values from the buffer to the array.

### NOSCALE

Specify the NOSCALE option to retrieve raw data from the device. When NOSCALE is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.

### Maximum scan rate

The actual maximum scan rate depends upon the data rate set for the channel(s) in the scan. Refer to the *Specifications* chapter in the hardware User's Guide for the maximum scan rate calculated for each selectable A/D data rate per number of input channels. For  $n$  channels whose data rates are  $f_1, f_2 \dots f_n$ , the maximum scan rate  $f_{\max}$  can be calculated from the following equation:

$$1/f_{\max} = (640 \mu\text{sec} + 1/f_1) + (640 \mu\text{sec} + 1/f_2) + (640 \mu\text{sec} + 1/f_n)$$

Refer to the *Specifications* chapter in the hardware User's Guide for the maximum scan rate calculated for each selectable A/D data rate per number of input channels.

### Scanning multiple channels at low data rates

When scanning multiple channels at low data rates, a BADRATE error may be returned if the Rate parameter is set too high for the number of channels in the scan. For example, when scanning multiple channels with the data rate set to 2.5 Hz, the maximum number of input channels using an integer scan rate is two, since the Universal Library doesn't accept non-integer rates.

To resolve this limitation, set the Rate parameter to 0. The Library will calculate the fastest allowable rate using the maximum scan rate equation, and run the scan at the calculated rate. The value returned to the Rate parameter will be the calculated value rounded to the nearest integer. Note that the Rate parameter may return 0, even though the fastest allowable rate was used for the conversion.

Refer to the *Specifications* chapter in the hardware User's Guide for the maximum scan rate calculated for each selectable A/D data rate per number of input channels.

### HIGHRESRATE

Specify the HIGHRESRATE scan option with `cbAInScan()/AInScan` to acquire data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel".

### Channel count

For output scans, the count must be set to an integer multiple of the number of channels or a [BADCOUNT](#) error is returned.

### Digital I/O channels

The state of each digital line can be read whether it is being used for output or for input. Each digital channel is an open-drain, and can either be driven low or allowed to float. If the digital line is intended to be used as an input, make sure that line is not driven low. A digital line is driven low by writing a logical "1" to the bit associated with the line. Implied in this is that the output function "inverts", so if all lines are driven low by writing a value of 255 to the port, the value read back will be 0.

### Device identifier

You can enter up to 64 characters for the value of the device's text identifier using the ConfigItem option BINODEID with `cbSetConfigString()`.

## USB-2500 Series

The USB-2500 Series includes the following hardware:

- USB-2523
- USB-2527
- USB-2533
- USB-2537

The USB-2500 Series supports the following UL and UL for .NET features.

### Analog Input

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbAPretrig\(\)](#)\*, [cbATrig\(\)](#), [cbFileAInScan\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [APretrig\(\)](#)\*, [ATrig\(\)](#), [FileAInScan\(\)](#)

\*Pretrigger capability is implemented in software. PretrigCount must be less than the TotalCount and cannot exceed 100,000 samples. If a trigger occurs while the number of collected samples is less than the PretrigCount, that trigger will be ignored. Requires a call to [cbSetTrigger](#)/ [SetTrigger](#) for the analog trigger type.

#### Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK, EXTTRIGGER, HIGHRESRATE

With EXTTRIGGER mode, the first channel in the scan is the analog trigger channel.

#### HighChan

USB-2537, USB-2533: 0 to 63 in single-ended mode, 0 to 31 in differential mode

USB-2527, USB-2523: 0 to 15 in single-ended mode, 0 to 7 in differential mode

#### Rate

Up to 1 MHz

#### Range

BIP10VOLTS ( $\pm 10$ volts)	BIPPT5VOLTS ( $\pm 0.5$ volts)
BIP5VOLTS ( $\pm 5$ volts)	BIPPT2VOLTS ( $\pm 0.2$ volts)
BIP2VOLTS ( $\pm 2$ volts)	BIPPT1VOLTS ( $\pm 0.1$ volts)
BIP1VOLTS ( $\pm 1$ volts)	

### Analog Output (USB-2537 and USB-2527 only)

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, NONSTREAMEDIO, SIMULTANEOUS

NONSTREAMEDIO can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524,288 samples.

#### HighChan

0 to 3

#### Rate

1 MHz

#### Range

Ignored - Not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

#### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

#### Pacing

Hardware pacing, external or internal clock supported.

## Digital I/O

► Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions.

## Configuration

### Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

## Port I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDInScan\(\)](#), [cbDOutScan\(\)](#)\*

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DInScan\(\)](#), [DOutScan\(\)](#)\*

\*FIRSTPORTA and FIRSTPORTB must be set for output to use this function. Refer to [DIO PortNum](#) in the *Hardware Considerations* section for more information.

### Options

ADCCLOCK, ADCCLOCKTRIG, BACKGROUND, CONTINUOUS, EXTCLOCK, EXTTRIGGER, NONSTREAMEDIO, HIGHRESRATE, WORDXFER

- The EXTTRIGGER option can only be used with the [cbDInScan\(\)](#) function. You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).
- The WORDXFER option can only be used with FIRSTPORTA.
- The NONSTREAMEDIO, ADCCLOCKTRIG, and ADCCLOCK options can only be used with the [cbDOutScan\(\)](#) function.
- The NONSTREAMEDIO option can only be used with the number of samples set equal to the size of the FIFO or less. The FIFO holds 524288 samples.

### Rate

12 MHz

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

### DataValue

0 to 255

0 to 65,535 using the WORDXFER option with FIRSTPORTA

## Bit I/O

### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortType

FIRSTPORTA

### BitNum

0 to 23

## Counter Input

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCConfigScan\(\)](#), [cbCInScan\(\)](#), [cbCClear\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CConfigScan\(\)](#), [CInScan\(\)](#), [CClear\(\)](#)

Note: Counters on these devices are zero-based (the first counter number is "0").

### Rate

6 MHz

### CounterNum

0 to 3

### Options

BACKGROUND, CONTINUOUS, EXTTRIGGER, HIGHRESRATE

You can use the [cbSetTrigger\(\)](#) function to program the trigger for rising edge, falling edge, or the level of the digital trigger input (TTL).

## LoadValue

0 to 65,535

The [Visual Basic signed integers](#) guidelines apply when using `cbCIn()` or `CIn()` for values greater than 32,767 and when using `cbCIn32()` or `CIn32()` for values greater than 2,147,483,647.

## Timer Output

UL: [cbTimerOutStart\(\)](#), [cbTimerOutStop\(\)](#)

UL for .NET: [TimerOutStart\(\)](#), [TimerOutStop\(\)](#)

## TimerNum

0 to 1

## Frequency

15.260 Hz to 1.0 MHz

## Triggering

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

## TrigType

TRIGABOVE, TRIGBELOW, TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE

Digital triggering (TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE) is not supported for pre-trigger acquisitions ([cbAPretrig\(\)](#) function).

Analog triggering (TRIGABOVE, TRIGBELOW) is not supported for the [cbDInScan\(\)](#) function and the [cbCInScan\(\)](#) function.

## Threshold

Analog hardware triggering, 12-bit resolution: 0 to 4,095 (supported for [cbAInScan\(\)](#) only)

Analog software triggering, 16-bit resolution: 0 to 65,535 (supported for [cbAPretrig\(\)](#) only)

## Temperature Input

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#), [cbGetTCValues\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#), [GetTCValues\(\)](#)

## Options

NOFILTER

## Scale

CELSIUS, FAHRENHEIT, KELVIN

## HighChan

0 to 3

## DAQ Input

UL: [cbDagInScan\(\)](#)

UL for .NET: [DagInScan\(\)](#)

## Options

BACKGROUND, BLOCKIO, CONTINUOUS, CONVERTDATA, DMAIO, EXTCLOCK, EXTTRIGGER, HIGHRESRATE

## ChanTypeArray

ANALOG, DIGITAL8, DIGITAL16, CTR16, CTR32LOW, CTR32HIGH, CJC, TC, SETPOINTSTATUS

When mixing ANALOG channel types with any other input types, the ANALOG channels should be the first in the list.

Note: For information on associating CJC channels with TC channels, refer to [Hardware considerations](#) below.

## ChanArray

ANALOG:

- USB-2537, USB-2533: 0 to 63 in single-ended mode, 0 to 31 in differential mode
- USB-2527, USB-2523: 0 to 15 in single-ended mode, 0 to 7 in differential mode

DIGITAL8: FIRSTPORTA, FIRSTPORTB, FIRSTPORTC

DIGITAL16: FIRSTPORTA

CTR16: 0 to 3 counters

CTR32LOW: 0 to 3 counters

CTR32HIGH: 0 to 3 counters

CJC: 0 to 2

TC: 0 to 3

SETPOINTSTATUS: 16-bit port that indicates the current state of the 16 possible setpoints.

ChanTypeArray flag value:

- SETPOINT\_ENABLE: Enables a setpoint. Refer to "Setpoints" in [Hardware Considerations](#) below for more information.

## Rate

Analog: Up to 1 MHz.

Digital: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

Counter: Up to 12 MHz if no analog channel is selected. Otherwise up to 1 MHz.

## GainArray

ANALOG only; ignore for other ChanTypeArray values.

BIP10VOLTS ( $\pm 10$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP1VOLTS ( $\pm 1$  volt)

BIPPT5VOLTS ( $\pm 0.5$  volt)

BIPPT2VOLTS ( $\pm 0.2$  volt)

BIPPT1VOLTS ( $\pm 0.1$  volt)

## PretrigCount

100,000 max

## DAQ Triggering

UL: [cbDagSetTrigger\(\)](#)

UL for .NET: [DagSetTrigger\(\)](#)

## TrigSource

TRIG\_IMMEDIATE, TRIG\_EXTTTL, TRIG\_ANALOGHW, TRIG\_ANALOGSW, TRIG\_DIGPATTERN, TRIG\_COUNTER, TRIG\_SCANCOUNT

## TrigSense

RISING\_EDGE, FALLING\_EDGE, ABOVE\_LEVEL, BELOW\_LEVEL, EQ\_LEVEL, NE\_LEVEL

## TrigEvent

START\_EVENT, STOP\_EVENT

## DAQ Setpoint

UL: [cbDagSetSetpoints\(\)](#)

UL for .NET: [DagSetSetpoints\(\)](#)

## SetpointFlagsArray

SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA, SF\_GREATERTHAN\_LIMITB, SF\_OUTSIDE\_LIMITS, SF\_HYSTERESIS, SF\_UPDATEON\_TRUEONLY, SF\_UPDATEON\_TRUEANDFALSE

## SetpointOutputArray

SO\_NONE, SO\_FIRSTPORTC, SO\_TMR0, SO\_TMR1

Also available for USB-2537 and USB-2527:

SO\_DAC0, SO\_DAC1, SO\_DAC2, SO\_DAC3

## LimitAArray

Any value valid for the associated input channel.

Ignored for SF\_GREATERTHAN\_LIMITB

## LimitBArray



Any value valid for the associated input channel and less than LimitA.  
Ignored for SF\_EQUAL\_LIMITA, SF\_LESSTHAN\_LIMITA

### Output#Array

For SetpointOutputArray = SO\_NONE: Ignored

For SetpointOutputArray = SO\_FIRSTPORTC: 0 to 65,535

For SetpointOutputArray = SO\_TMR#: 0 (to disable the timer) or 15.26 to 1,000,000 (to set the output frequency)

For SetpointOutputArray = SO\_DAC#: Voltage values between -10 and +10

### OutputMask#Array

For SetpointOutputArray = SO\_FIRSTPORTC: 0 to 65,535

For SetpointOutputArray = all other values: Ignored

### SetpointCount

0 (to disable setpoints) to 16

## DAQ Output (USB-2537 and USB-2527 only)

UL: [cbDagOutScan\(\)](#)

UL for .NET: [DagOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS, NONSTREAMEDIO, ADCCLOCKTRIG, ADCCLOCK

### ChanType

ANALOG, DIGITAL16

### ChanArray

ANALOG: 0 to 3

DIGITAL16: FIRSTPORTA (FIRSTPORTB must be configured as an output)

### Rate

ANALOG: Up to 1 MHz

DIGITAL16: Up to 12 MHz (system-dependent) if no analog channel is selected. Otherwise up to 1 MHz.

### Range

Ignored

## Hardware considerations

### Associating CJC channels with TC channels

The TC channels must immediately follow their associated CJC channels in the channel array. For accurate thermocouple readings, associate CJC0 with TC0, CJC1 with TC1 and TC2, and CJC2 with TC3.

The board must be configured for differential inputs when using thermocouples.

### Channel count

For input and output scans, the count must be set to an integer multiple of the number of channels or a [BADCOUNT](#) error is returned.

### Sampling and update rates

Sampling and update rates are system-dependent. Data overruns/underruns may occur with higher sampling rates when using BACKGROUND and CONTINUOUS modes. To avoid this, use a larger buffer/count size, or use NONSTREAMEDIO mode, if supported.

The minimum size buffer is 256 for cbAOutScan(), cbDInScan(), and cbDOutScan(). Values less than 256 result in a [BADBUFFERSIZE](#) error.

### HIGHRESRATE

Specify the HIGHRESRATE scan option to acquire data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1,000 seconds per channel".

### Settling time

For most applications, settling time should be left at the default value of 1  $\mu$ s. However, if you are scanning multiple channels and one or more channels are connected to a high impedance source, you may get better results by increasing the settling time. Keep in mind that increasing the settling time reduces the maximum acquisition rate. You can set the time between A/D conversions with the ADC Settling Time option in InstaCal. Select between 1  $\mu$ s, 5  $\mu$ s, 10  $\mu$ s, or 1 ms.

### Setpoints

You enable setpoints with the SETPOINT\_ENABLE flag. This flag must be OR'ed with the ChanTypeArray argument values. You set the setpoint criteria with cbDaqSetSetpoints()/DaqSetSetpoints(). The number of channels set with the SETPOINT\_ENABLE flag must match the number of setpoints set by the SetpointCount argument (cbDaqSetSetpoints()/DaqSetSetpoints()).

### Output non-streamed data to a DAC output channel

With NONSTREAMEDIO mode, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. The FIFO holds 524288 samples. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample updates are transferred or downloaded to the device, the device is responsible for outputting the data. You can't change the output buffer once the output begins.

### Trigger DAC output operations with the ADC clock

Specify the ADCCLOCK option to trigger a data output operation upon the start of the ADC clock.

### DIO PortNum

For cbDOutScan()/DOutScan() and cbDaqOutScan()/DaqOutScan(), FIRSTPORTA and FIRSTPORTB are treated as one 16-bit port. These functions can only be used with FIRSTPORTA. You must configure both FIRSTPORTA and FIRSTPORTB for output using the cbDConfigPort() function.

### Synchronous scanning with multiple boards

You can operate up to four USB-2500 Series boards synchronously by setting the direction of the A/D and D/A pacer pins (**XAPCR** or **XDPCR**) in InstaCal.

On the board used to pace each device, set the pacer pin that you want to use (XAPCR or XDPCR) for *Output*. On the board(s) that you want to synchronize with this board, set the pacer pin that you want to use (XAPCR or XDPCR) for *Input*.

Set the direction using the **XAPCR Pin Direction** and **XDPCR Pin Direction** settings on the InstaCal configuration dialog. If you have an older version of InstaCal, these settings might be labeled "ADC Clock Output" (set to *Enabled* to configure XAPCR for output) or "DAC Clock Output" (set to *Enabled* to configure XDPCR for output).

Wire the pacer pin configured for output to each of the pacer input pins that you want to synchronize.

### Quadrature encoder operations

To configure a counter channel as a multi-axis quadrature encoder, use the [cbCConfigScan\(\)/CConfigScan\(\)](#) Mode argument values to set a specified counter to encoder mode, set the encoder measurement mode to X1, X2, or X4, and then set the count to be latched either by the internal "start of scan" signal (default) or by the signal on the mapped channel.

You can optionally perform the following operations:

- Enable gating, so that the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.
- Enable "latch on Z" to latch counter outputs using the Encoder Z mapped signal.
- Enable "clear on Z" so that the counter is cleared on the rising edge of the mapped (Z) channel. By default, "clear on Z" is disabled, and the counter is not cleared.

### Asynchronous reads

The CConfigScan() method's Bit32 counter mode option only affects counter resolution for asynchronous calls ([CIn\(\)](#) and [CIn32\(\)](#)), and only when the counter is configured for StopAtMax.

This mode is recommended for use only with CIn32(). Using the Bit32 option with CIn() is not very useful, since the value returned by CIn() is only 16 bits. The effect is that the value returned by CIn() rolls over 65,535 times before stopping.

# USB-7202

The USB-7202 supports the following UL and UL for .NET features.

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#)\*, [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#)\*, [FileAInScan\(\)](#), [ATrig\(\)](#)

\*The channel-gain queue is limited to eight elements. The channels specified in the queue must be contiguous and in increasing order, except when wrapping around from channel 7 to channel 0. The gains may be any valid value.

### Options

BACKGROUND, BLOCKIO\*, SINGLEIO\*, BURSTIO\*\*, CONTINUOUS, EXTTRIGGER, CONVERTDATA, NOCALIBRATEDATA, EXTCLOCK, HIGHRESRATE

\* The packet size is based on the Options setting: When set to BLOCKIO, the packet size is 31 samples. When set to SINGLEIO, the packet size equals the number of channels being sampled.

\*\* BURSTIO can only be used with the number of samples (Count) set equal to the size of the FIFO or less. The USB-7202 FIFO holds 32,768 samples. BURSTIO cannot be used with the CONTINUOUS option.

### Mode

Single-ended

### HighChan

0 to 7 in single-ended mode

### Count

In BURSTIO mode, Count *must* be an integer multiple of the number of channels in the scan:

- For one-, two-, four-, and eight-channel scans, the maximum Count is 32,768 samples.
- For three- and six-channel scans, the maximum Count is 32,766 samples.
- For five-channel scans, the maximum Count is 32,765 samples.
- For seven-channel scans, the maximum Count is 32,767 samples.

### Rate

200 kilohertz (kHz) maximum for BURSTIO mode (50 kHz for any one channel).

The maximum rate is 100 kHz for all other modes (50 kHz for any one channel).

When using [cbAInScan\(\)](#) or [AInScan\(\)](#), the minimum sample rate is 1 Hz. In BURSTIO mode, the minimum sample rate is 20 Hz/channel.

### Range

BIP10VOLTS ( $\pm 10$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP1VOLTS ( $\pm 1$  volts)

### Pacing

Hardware pacing, internal clock supported. External clock supported via the SYNC pin.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

Digital triggering: TRIGPOSEDGE, TRIGNEGEDGE

External digital (TTL) hardware triggering supported. Set the hardware trigger source with the Trig\_In input.

## Digital I/O

### Configuration

#### Functions

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

## Port I/O

Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#)

PortNum

AUXPORT (eight bits, bit-configurable)

DataValue

0 to 255 for AUXPORT

## Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

## Counter I/O

Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

The counter on this device is zero-based.

CounterNum

1

Count

$2^{32}-1$  when reading the counter.

LoadValue

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* topic apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

RegNum

LOADREG0

## Event Notification

Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

Event types

ON\_SCAN\_ERROR, ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a Measurement Computing USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware Considerations

### Acquisition rate

Since the maximum data acquisition rate depends on the system connected to the device, it is possible to "lose" data points when scanning at higher rates. If the requested speed cannot be sustained, an [OVERRUN](#) error will occur.

### Continuous scans

When running [cbAInScan\(\)](#)/ [AInScan\(\)](#) with the CONTINUOUS option, consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, make the total number of samples an integer multiple of the packet size and the number of channels.

### EXTCLOCK

You can set the SYNC pin as a pacer input or a pacer output from InstaCal. By default, this pin is set for pacer input. If set for output when using the [cbAInScan\(\)](#)/[AInScan\(\)](#) option, EXTCLOCK results in a [BADOPTION](#) error.

### BURSTIO

BURSTIO mode allows higher sampling rates for sample counts up to the size of the FIFO. The USB-7202 device's FIFO holds 32,768 samples. Data is collected into the device's local FIFO. Data transfers to the PC don't occur until the scan completes. For BACKGROUND scans, the Count and Index returned by [cbGetStatus\(\)](#) and [GetStatus\(\)](#) remain 0, and Status = RUNNING until the scan finishes. The Count and Index are not updated until the scan is completed. When the scan is complete and the data is retrieved, [cbGetStatus\(\)](#) and [GetStatus\(\)](#) are updated to the current Count and Index, and Status = IDLE.

BURSTIO is required for aggregate Rate settings above 100 kHz, but Count is limited to sample counts up to the size of the FIFO (32,768 samples). Count settings must be an integer multiple of the number of channels in the scan.

## USB-7204

The USB-7204 supports the following UL and UL for .NET features.

## Analog input

### Functions

UL: [cbAIn\(\)](#), [cbAInScan\(\)](#), [cbALoadQueue\(\)](#), [cbFileAInScan\(\)](#), [cbATrig\(\)](#)

UL for .NET: [AIn\(\)](#), [AInScan\(\)](#), [ALoadQueue\(\)](#), [FileAInScan\(\)](#), [ATrig\(\)](#)

### Options

BACKGROUND, BLOCKIO\*, CONTINUOUS, EXTCLOCK, EXTTRIGGER, RETRIGMODE\*\*, NOCALIBRATEDATA, SINGLEIO, HIGHRESRATE

\*The packet size is based on the Options setting:

BLOCKIO: 31

SINGLEIO: 1

\*\*RETRIGMODE can only be used with [cbAInScan\(\)](#)/[AInScan\(\)](#).

### HighChan

0 to 7 in single-ended mode

0 to 3 in differential mode

### Count

In CONTINUOUS mode, Count must be an integer multiple of the packet size.

### Rate

50 kHz maximum for BLOCKIO mode.

The throughput depends on the system being used. Most systems can achieve 40 kHz aggregate.

When using [cbAInScan\(\)](#)/[AInScan\(\)](#), the minimum sample rate is 1 Hz.

### Range

Single-ended:

BIP10VOLTS ( $\pm 10$  volts)

Differential:

BIP20VOLTS ( $\pm 20$  volts)

BIP2PT5VOLTS ( $\pm 2.5$  volts)

BIP10VOLTS ( $\pm 10$  volts)

BIP2VOLTS ( $\pm 2$  volts)

BIP5VOLTS ( $\pm 5$  volts)

BIP1PT25VOLTS ( $\pm 1.25$  volts)

BIP4VOLTS ( $\pm 4$  volts)

BIP1VOLTS ( $\pm 1$  volts)

### Pacing

Hardware pacing, internal clock supported.

External clock supported via the SYNC pin.

## Triggering

### Functions

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

### TrigType

TRIGPOSEDGE

TRIGNEGEDGE

External digital (TTL) hardware triggering is supported. Use the TRIG\_IN input (pin # 18 on the screw terminal) for the external trigger signal.

## Analog Output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS

The number of samples (Count) in a CONTINUOUS scan must be an integer multiple of the packet size (32).

### HighChan

0 to 1

### Count

Count must be an integer multiple of the number of channels in the scan.

In a CONTINUOUS scan, Count must be an integer multiple of the packet size (32).

### Rate

Up to 10 kHz maximum for a single channel

Up to 5 kHz maximum for two channels

### Range

Ignored - not programmable; fixed at UNI4VOLTS (0 to 4 V, nominal. Actual range is 0 to 4.096 V.)

### DataValue

0 to 4,095

## Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### Configuration Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

### Port I/O Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB

DataValue

0 to 255 for FIRSTPORTA or FIRSTPORTB

### Bit I/O Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 15 for on FIRSTPORTA

# Counter I/O

## Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#)/[CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#)/[CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

The counter on this device is zero-based.

## CounterNum

1

## Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

[cbCLoad\(\)](#)/[CLoad\(\)](#) and [cbCLoad32\(\)](#)/[CLoad32\(\)](#) are only used to reset the counter for this device to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* section apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

## RegNum

LOADREG0

## Event notification

### Functions

UL: [cbEnableEvent\(\)](#),[cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#),[DisableEvent\(\)](#)

### Event types

ON\_SCAN\_ERROR (analog input and analog output), ON\_DATA\_AVAILABLE, ON\_END\_OF\_AI\_SCAN, ON\_END\_OF\_AO\_SCAN

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink. When you have several USB devices connected to the computer, use this function to identify a particular device by making its LED blink.

## Hardware considerations

### Acquisition rate

Since the maximum data acquisition rate depends on the system connected to the device, it is possible to "lose" data points when scanning at higher rates. The Universal Library cannot always detect this data loss.

Most systems can sustain rates of 40 kS/s aggregate in BLOCKIO mode, and 1 kS/s aggregate in SINGLEIO mode.

### EXTCLOCK

By default, the SYNC pin is configured for pacer output and provides the internal pacer A/D clock signal. To configure the pin for pacer input, use the EXTCLOCK option.

If you use the EXTCLOCK option, make sure that you disconnect from the external clock source when you test or calibrate the device with InstaCal, as the SYNC pin drives the output.

### Repetitive trigger events

Use [RETRIGMODE](#) with [cbAInScan\(\)](#) to set up repetitive trigger events. Use the ConfigItem option BIADTRIGCOUNT with [cbSetConfig\(\)](#) to set the A/D trigger count, and the ConfigItem option BIDACTRIGCOUNT to set the D/A trigger count.

### Resolution

When configured for single-ended mode, the resolution of the data is 11 bits (data values between 0 and 2,047). However, the Universal Library maps this data to 12-bit values, so the range of data is no different from the differential configuration. Consequently, the data returned contains only even numbers between 0 and 4,094 when the NOCALIBRATEDATA option is used.



### Continuous scans

When running [cbAInScan\(\)](#)/ [AInScan\(\)](#) with the CONTINUOUS option, consider the packet size and the number of channels being scanned. In order to keep the data aligned properly in the array, set the total number of samples to be an integer multiple of the packet size and the number of channels in the scan.

### Concurrent operations

The USB-7204 supports these concurrent operations:

UL function/method	Can be run with:
<a href="#">cbAOutScan()</a> / <a href="#">AOutScan()</a> (BACKGROUND mode)	<ul style="list-style-type: none"><li>▪ <a href="#">cbDOut()</a> / <a href="#">DOut()</a></li><li>▪ <a href="#">cbCLoad()</a> / <a href="#">CLoad()</a></li><li>▪ <a href="#">cbCLoad32()</a> / <a href="#">CLoad32()</a></li></ul>
<a href="#">cbAInScan()</a> / <a href="#">AInScan()</a> (BACKGROUND mode)	<ul style="list-style-type: none"><li>▪ <a href="#">cbAOut()</a> / <a href="#">AOut()</a></li><li>▪ <a href="#">cbDIn()</a> / <a href="#">DIn()</a></li><li>▪ <a href="#">cbDBitIn()</a> / <a href="#">DBitIn()</a></li><li>▪ <a href="#">cbDOut()</a> / <a href="#">DOut()</a></li><li>▪ <a href="#">cbDBitOut()</a> / <a href="#">DBitOut()</a></li><li>▪ <a href="#">cbDConfigPort()</a> / <a href="#">DConfigPort()</a></li><li>▪ <a href="#">cbCIn()</a> / <a href="#">CIn()</a></li><li>▪ <a href="#">cbCIn32()</a> / <a href="#">CIn32()</a></li><li>▪ <a href="#">cbCLoad()</a> / <a href="#">CLoad()</a></li><li>▪ <a href="#">cbCLoad32()</a> / <a href="#">CLoad32()</a></li></ul>

### Channel-gain queue

When using [cbALoadQueue\(\)](#)/[ALoadQueue\(\)](#), the channel gain queue is limited to 16 elements.

The queue accepts any combination of valid channels and gains in each element.

### Analog output

When you include both analog output channels in [cbAOutScan\(\)](#)/[AOutScan\(\)](#), the two channels are updated simultaneously.

## Analog Output Hardware

All devices with analog outputs support the [cbAOut\(\)/AOut\(\)](#), [cbVOut\(\)/VOut\(\)](#), and [cbAOutScan\(\)/AOutScan\(\)](#) functions.

The [cbAOutScan\(\)/AOutScan\(\)](#) functions are designed primarily for devices which support hardware paced analog output, but it is also useful when simultaneous update of all channels is desired. If the hardware is configured for simultaneous update, this function loads each DAC channel with the appropriate value before issuing the update command.

## CIO- and PCIM-DDA06 Series

The CIO- and PCIM-DDA06 Series includes the following hardware:

- CIO-DDA06
- CIO-DDA06/16
- CIO-DDA06/Jr
- CIO-DDA06/Jr/16
- PCIM-DDA06/16

The CIO- and PCIM-DDA06 Series supports the following UL and UL for .NET features.

## Analog output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS (CIO-DDA06 Series only)

### HighChan

0 to 5

### Count

HighChan - LowChan + 1 max

### Rate

Ignored

### Range

Ignored - not programmable

CIO-DDA06/Jr and CIO-DDA06/Jr/16:

Fixed at BIP5VOLTS ( $\pm 5$  volts)

CIO-DDA06/16 and PCIM-DDA06/16: fixed at one of four switch-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)

CIO-DDA06/12: fixed at one of eight switch-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2VOLTS (0 to 2.5 volts)
BIP1PT67VOLTS ( $\pm 1.67$ volt)	UNI1VOLTS (0 to 1.67 volt)

### DataValue

0 to 4,095

For /16 hardware, the following argument values are also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

**DataValue**

0 to 255 for FIRSTPORTA or FIRSTPORTB

0 to 15 for FIRSTPORTC

**BitNum**

0 to 23 for FIRSTPORTA

**Hardware considerations****Pacing analog output**

Software only

**Initializing the "zero power-up" state**

When using the CIO-DDA06 "zero power-up state" hardware option, use [cbAOutScan\(\)](#)/ [AOutScan\(\)](#) to set the desired output value and enable the DAC outputs.

## CIO-DAC Series (excluding HS) and PC104-DAC06

The CIO-DAC Series (excluding HS) includes the following hardware:

- CIO-DAC02, CIO-DAC02/16
- CIO-DAC08, CIO-DAC08/16, CIO-DAC08-I
- CIO-DAC16, CIO-DAC16/16, CIO-DAC16-I

This topic also includes the PC104-DAC06.

The CIO-DAC Series (excluding HS) and PC104-DAC06 support the following UL and UL for .NET features.

## Analog output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS

### HighChan

DAC02: 0 to 1

DAC06: 0 to 5

DAC08: 0 to 7

DAC16: 0 to 15

### Rate

Ignored

### Count

HighChan - LowChan + 1 max

### Range

Ignored - not programmable. The range for all devices in this series is fixed at one of four jumper/switch-selectable ranges:

BIP10VOLTS ( $\pm 10$  volts)

UNI10VOLTS (0 to 10 volts)

BIP5VOLTS ( $\pm 5$  volts)

UNI5VOLTS (0 to 5 volts)

In addition to these four ranges, the CIO-DAC16, CIO-DAC08, and CIO-DAC02/16 allow switch-selectable ranges:

BIP2PT5VOLTS ( $\pm 2.5$  volts)

UNI2PT5VOLTS (0 to 2.5 volts)

The CIO-DAC02, CIO-DAC08-I, and CIO-DAC16-I allow:

MA4TO20 (4 to 20 mA)

### DataValue

0 to 4,095

For the /16 series, the following argument values are also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Hardware considerations

### Pacing analog output

Software only

## CIO-DAC04/12-HS

The CIO-DAC04/12-HS supports the following UL and UL for .NET features.

### Analog output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS

#### HighChan

0 to 3

#### Rate

500 kilohertz (kHz)

#### Range

Ignored - not programmable; fixed at one of six switch-selectable ranges:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)

#### DataValue

0 to 4,095

### Digital I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

#### PortNum

AUXPORT\*

#### DataValue

0 to 255

#### BitNum

0 to 7

\* AUXPORT is not configurable for these boards.

### Hardware considerations

#### Pacing analog output

- Hardware pacing, external or internal clock supported.
- The external clock is hardwired to the DAC pacer. If an internal clock is to be used, do not connect a signal to the External Pacer input.

## cSBX-DDA04

The cSBX-DDA04 supports the following UL and UL for .NET features.

## Analog output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, SIMULTANEOUS

### Rate

300,000

### Pacing

Hardware pacing, external or internal clock supported.

## Digital I/O

### Functions

UL for .NET: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDOutScan\(\)](#), [cbDInScan\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DOutScan\(\)](#), [DInScan\(\)](#)

### PortNum

AUXPORT\*

\* AUXPORT is not configurable for this board.

### DataValue

0 to 255 using [cbDIn\(\)](#) or [cbDInScan\(\)](#)

0 to 16383

### BitNum

0 to 7 using [cbDBitIn\(\)](#)

0 to 13 using [cbDBitOut\(\)](#)

### Rate

500 kHz (refer to "Hardware considerations" below)

### Pacing

Hardware pacing supported

## Hardware considerations

### Interleaving analog and digital output data

The cSBX-DDA04 board allows interleaving of analog and digital output data. To support interleaving, a control bit indicates the data type. The control bit is the MSB of each 16-bit word of analog or digital data. The MSB = 0 for analog data and the MSB = 1 for digital data.

The data is passed to the board and then directed to the correct output type by hardware on the board which detects and acts on the MSB control bit.

- To use this interleaving capability with the Universal Library, set HighChan and LowChan to NOTUSED, and indicate the data type and channel in the most significant four bits of the data values in the buffer.
- To use this interleaving capability with the Universal Library for .NET, set HighChan and LowChan to NotUsed, and indicate the data type and channel in the most significant four bits of the data values in the buffer.

## PCI-DAC6702, PCI-DAC6703

The PCI-DAC6702 and PCI-DAC6703 support the following UL and UL for .NET features.

### Analog output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### HighChan

PCI-DAC6702: 7

PCI-DAC6703: 15

#### Count

HighChan - LowChan + 1 max

#### Rate

Ignored

#### Range

Ignored - Not programmable; fixed at BIP10VOLTS ( $\pm 10.1$  V)

#### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Digital I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#), [cbDConfigBit\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#), [DConfigBit\(\)](#)

#### PortNum

AUXPORT is bitwise configurable for these boards, and must be configured using [cbDConfigBit\(\)](#) or [cbDConfigPort\(\)](#) before use as output.

#### DataValue

0 to 255

#### BitNum

0 to 7

### Configuration

#### Functions

UL: [cbGetConfig\(\)](#), [cbSetConfig\(\)](#)

UL for .NET: [GetDACStartup\(\)](#), [GetDACUpdateMode\(\)](#), [SetDACStartup\(\)](#), [SetDACUpdateMode\(\)](#)

#### ConfigItem

BIDACSTARTUP, BIDACUPDATEMODE, BIDACUPDATECMD

### Hardware considerations

#### Pacing analog output

Software only



## PCI-DDA02, DDA04, and DDA08 Series

The PCI-DDA02, DDA04, and DDA08 Series includes the following hardware:

- PCI-DDA02/12, PCI-DDA02/16
- PCI-DDA04/12, PCI-DDA04/16
- PCI-DDA08/12, PCI-DDA08/16

The PCI-DDA02, DDA04, and DDA08 Series support the following UL and UL for .NET features.

## Analog output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS

### HighChan

DDA02:0 to 1

DDA04:0 to 3

DDA08:0 to 7

### Count

HighChan - LowChan + 1 max

### Rate

Ignored

### Range

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)
BIP2PT5VOLTS ( $\pm 2.5$ volts)	UNI2PT5VOLTS (0 to 2.5 volts)

### DataValue

0 to 4,095

For /16 hardware, the following argument values are also valid:

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Pacing

Software only

## Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

### DataValue

0 to 15 for PORTCL and PORTCH

0 to 255 for PORTA or PORTB

### BitNum

0 to 47 using FIRSTPORTA

## Hardware considerations

## Pacing analog output

Software only

## PCM-DAC Series and PC-CARD-DAC08

The PCM-DAC Series includes the following hardware:

- PCM-DAC02
- PCM-DAC08

This topic also includes the PC-CARD-DAC08.

The PCM-DAC Series and PC-CARD-DAC08 support the following UL and UL for .NET features.

### Analog output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

PCM-DAC02: Ignored

PCM-DAC08 and PC-CARD-DAC08: SIMULTANEOUS

#### HighChan

DAC02: 0 to 1

DAC08: 0 to 7

#### Rate

Ignored

#### Count

HighChan - LowChan + 1 max

#### Range

PCM-DAC08 and PC-CARD-DAC08: Ignored - Not programmable; fixed at BIP5VOLTS ( $\pm 5$  volts)

PCM-DAC02:

BIP10VOLTS ( $\pm 10$ volts)	UNI10VOLTS (0 to 10 volts)
BIP5VOLTS ( $\pm 5$ volts)	UNI5VOLTS (0 to 5 volts)

#### DataValue

0 to 4,095

## Digital I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB

#### DataValue

0 to 15 using FIRSTPORTA or FIRSTPORTB

#### BitNum

0 to 7 using FIRSTPORTA

## Hardware considerations

### Pacing analog output

Software only

### Digital configuration

Supports two configurable 4-bit ports — FIRSTPORTA and FIRSTPORTB. Each can be independently configured as either inputs or outputs via [cbDConfigPort\(\)](#) or [DConfigPort\(\)](#).

## USB-3100 Series

The USB-3100 Series includes the following hardware:

- USB-3101, USB-3102, USB-3103, USB-3104, USB-3105, USB-3106
- USB-3110, USB-3112, USB-3114

The USB-3100 Series supports the following UL and UL for .NET features.

## Analog output

### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

### Options

SIMULTANEOUS (cbAOutScan() / AOutScan() only)

### HighChan

USB-3101, USB-3102, and USB-3110: 0 to 3

USB-3103, USB-3104, and USB-3112: 0 to 7

USB-3105, USB-3106, and USB-3114: 0 to 15

### Count

HighChan - LowChan + 1 max

### Rate

Ignored

### Range

Ignored (except in the case of cbVOut), since it's not programmable; configurable for BIP10VOLTS ( $\pm 10$  volts), or UNI10VOLTS (0 to 10 volts), or MA0TO20 (0 to 20 mA) via InstaCal.

USB-3102, USB-3104, USB-3106: Also configurable for MA0TO20 (0 to 20mA) via InstaCal.

### DataValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

## Digital I/O

### Configuration

Functions

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AUXPORT

DataValue

0 to 255 for AUXPORT

### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

## Counter I/O

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

### Count

$2^{32}-1$  when reading the counter.

### LoadValue

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter for this board to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* section apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

### RegNum

LOADREG1

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the USB LED on a Measurement Computing USB device to blink.

When you have several USB devices connected to the computer, use these functions to identify a particular device by making its USB LED blink.

## Hardware considerations

### Scan options

The SIMULTANEOUS scan option can only be used with [cbAOutScan\(\)](#) / [AOutScan\(\)](#).

### Simultaneous mode

Set the direction of the SYNCLD pin (pin 49) with the **Simultaneous Mode** option in InstaCal to be either Master (output) or Slave (input).

- Specify the SIMULTANEOUS scan option and set the Simultaneous Mode option to "Master" to output the internal D/A LOAD signal on the SYNCLD pin.
- Specify the SIMULTANEOUS scan option and set the SIMULTANEOUS Mode option to "Slave" to configure the SYNCLD pin to receive the D/A LOAD signal from an external source. Output channels are updated simultaneously when the SYNCLD receives the signal.

In slave mode, analog outputs may either be updated immediately or when a positive edge is seen on the SYNCLD pin (this is under software control.) The SYNCLD pin must be at a low logic level for DAC outputs to update immediately. If an external source is pulling the pin high, no update will occur.

When you do not specify SIMULTANEOUS, the analog outputs are updated in sequential order, and the SYNCLD pin is ignored.

**External current limiting may be required for high drive devices (USB-3110, USB-3112, USB-3114)**

The voltage outputs on the USB-3110, USB-3112, and USB-3114 incorporate high-drive current output capability. The high drive current outputs allow each of the voltage outputs to sink/source up to 40 mA (maximum) of load current.

The voltage outputs should not be kept in a short-circuit condition for longer than the specified 100 ms. For those applications that may potentially exceed the 40 mA maximum current limit or the 100 ms short-circuit condition, external current limiting must be used to prevent potential damage to the USB-3110.

**Simultaneous update of voltage and current outputs (USB-3102, USB-3104, USB-3106)**

Each voltage output channel on the USB-3102, USB-3104, USB-3106 has an associated current output channel. The voltage and current outputs are grouped as channel pairs. Each D/A converter output controls a voltage and current channel pair simultaneously. When you write to a voltage output, its associated current output is also updated.

Each voltage/current channel pair can be updated individually or simultaneously. Leave each pair of unused voltage and current outputs disconnected.

## USB-3101FS

The USB-3101FS supports the following UL and UL for .NET features:

### Analog Output

#### Functions

UL: [cbAOut\(\)](#), [cbVOut\(\)](#), [cbAOutScan\(\)](#)

UL for .NET: [AOut\(\)](#), [VOut\(\)](#), [AOutScan\(\)](#)

#### Options

BACKGROUND, CONTINUOUS, NOCALIBRATEDATA, SCALEDATA, SIMULTANEOUS

#### Count

When using output scanning, Count must be an integer multiple of the number of channels in the scan.

#### HighChan

0 to 3

#### Rate

100 kS/s per channel, maximum

#### Range

Ignored, (except in the case of cbVOut) since it's not programmable; fixed at BIP10VOLTS ( $\pm 10$  volts)

### Event Notification

#### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

#### Event types

UL: ON\_END\_OF\_AO\_SCAN, ON\_SCAN\_ERROR

UL for .NET: OnEndOfAoScan, OnScanError

## **COM422 Series Hardware**

No library functions are supported for these devices, but you can use InstaCal to configure the serial protocol in conjunction with the Set422.exe utility program. All other serial communications are handled by Windows standard serial communications handlers.



## COM485 Series Hardware

COM485 Series boards support the Universal Library function [cbRS485\(\)](#) and the Universal Library for .NET function [RS485\(\)](#) for controlling the transmit and receive enable register. All other serial communications are handled by Windows standard serial communications handlers.

## Counter Hardware

### Visual Basic signed integers

When reading or writing ports that are 16 or more bits wide, be aware of the following issue using signed integers, which is required when using Visual Basic:

On some devices, such as the CIO-CTR10 count register or AUXPORT digital ports, the ports are 16-bits wide or more. When accessing the data at these ports, the digital values are arranged as a single 16-bit word or a 32-bit double word.

When using signed integers, values above 0111 1111 1111 1111 (32,767 decimal) can be confusing. The next increment, 1000 0000 0000 0000 has a decimal value of -32,768. When using signed integers, this is the value that is returned from a 16-bit counter at half of maximum count. The value for full count (just before the counter turns over) is -1. Keep this in mind if you are using Visual Basic, since Visual Basic does not supply unsigned integers (values from 0 to 65,535) or unsigned longs (values from 0 to 4,294,967,295). Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.

The Universal Library provides functions for initialization and configuration of counter chips, and can configure a counter for any of the counter operations.

However, counter configuration does not include counter-use, such as event counting and pulse width. Counter-use is accomplished by programs which use the counter functions. The Universal Library provides the [cbCFreqIn\(\)](#) function for counter use, while the Universal Library for .NET provides the [CFreqIn\(\)](#) method. Other functions and methods may be added for counter use to later revisions.

### Counter chip data sheets

To use a counter for any but the simplest counting function, you must 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.

To fully understand and maximize the performance of counter/timer hardware and the related function calls, review the following related data sheet(s):

Chip name	Data sheet location
82C54	<a href="#">82C54.pdf</a> located in the <i>Documents</i> subdirectory where the UL is installed (C:/Program files/Measurement Computing/DAQ by default).
AM9513	<a href="#">9513A.pdf</a> located in the <i>Documents</i> subdirectory where the UL is installed
Z8536	The Z8536 document is included with the product designed with this chip.
LS7266	<a href="#">ls7266r1.pdf</a> located in the <i>Documents</i> subdirectory where the UL is installed.



### 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.

## CTR Series

The CTR Series includes the following hardware:

- PCI-CTR05
- PCI-CTR10
- PCI-CTR20HD
- CIO-CTR05
- CIO-CTR10
- CIO-CTR20HD
- CIO-CTR10HD
- PC104-CTR10HD

The CTR Series supports the following UL and UL for .NET features.

## Counter I/O

### Functions

UL: [cbC9513Config\(\)](#), [cbC9513Init\(\)](#), [cbCStoreOnInt\(\)](#), [cbCFreqIn\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C9513Config\(\)](#), [C9513Init\(\)](#), [CStoreOnInt\(\)](#), [CFreqIn\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

### CounterNum

1 to 5 (All boards in this series)

CTR10 and CTR10HD also support counters 6 through 10

CTR20HD also support counters 11 through 20

### RegNum

LOADREG1 – 5, HOLDREG1 – 5, ALARM1CHIP1, ALARM2CHIP1

CTR10 also supports LOADREG6 – 10, HOLDREG6 – 10, ALARM1CHIP2, ALARM2CHIP2

CTR20HD also supports LOADREG6 – 20, HOLDREG6 – 20, ALARM1CHIP2 – ALARM1CHIP4, ALARM2CHIP2 – ALARM2CHIP4

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### ChipNum

1 (All boards in this series)

CTR10 and CTR10HD also support chip 2

CTR20HD also support chips 3 and 4

### FOUT Source

CTRINPUT1 – 5, GATE1 – 5, FREQ1 – 5

These values refer to the sources on a particular 9513 chip, so are limited to the sources on that particular chip. For example, to set the source to the input for counter 6, use CTRINPUT1 (the first counter on the second 9513 chip).

### CountSource

TCPREVCTR, CTRINPUT1 – 5, GATE1 – 5, FREQ1 – 5

These values refer to the sources on a particular 9513 chip, so are limited to the sources on that particular chip. For example, to set the source to the input for counter 6, use CTRINPUT1 (the first counter on the second 9513 chip). Likewise for the TCPREVCTR value: when applied to the first counter on a chip (counter 6, for example) the "previous counter" is counter 5 on that chip (for this example, counter 10).

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

### PortNum

AUXPORT\*

### DataValue

CTR05: 0 to 255

CTR10: 0 to 65,535 (refer to [Visual Basic signed integers](#) in the *Introduction: Counter Boards topic* for more information.)

#### BitNum

CTR05: 0 to 7

CTR10: 0 to 15

\* AUXPORT is not configurable for these boards.

## Event Notification

PCI-CTR05, PCI-CTR10 and PCI-CTR20HD only.

#### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

#### EventType

ON\_EXTERNAL\_INTERRUPT/OnExternalInterrupt

## Hardware considerations

#### Clock Input Frequency (PCI boards only)

The clock source for each of the four counters is configurable with InstaCal:

PCI-CTR05: 1 MHz, 5 MHz

PCI-CTR10: 1 MHz, 3.33 MHz, 5 MHz

PCI-CTR20HD: 1 MHz, 1.67 MHz, 3.33 MHz, 5 MHz, or External

#### Event notification

ON\_EXTERNAL\_INTERRUPT cannot be used in conjunction with [cbCStoreOnInt\(\)](#) or [CStoreOnInt\(\)](#).

CTR Series boards that support event notification only support external rising edge interrupts.

## PCI-INT32, CIO-INT32

The PCI-INT32 and CIO-INT32 support the following UL and UL for .NET features.

### Counter I/O

#### Functions

UL: [cbC8536Config\(\)](#), [cbC8536Init\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8536Config\(\)](#), [C8536Init\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

#### CounterNum

1 to 6

#### ChipNum

1 or 2

#### RegNum

LOADREG1 through LOADREG6

#### LoadValue

Values up to 65,535 ( $2^{16} - 1$ ) may be used. (Refer to [Visual Basic signed integers](#) in the "Introduction: Counter Boards" topic for more information.)

### Digital I/O

#### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL

SECONDPORTA, SECONDPORTB and SECONDPORTCL

#### DataValue

0 to 255 using PORTA or PORTB

0 to 15 using PORTCL

#### BitNum

0 to 39 using FIRSTPORTA

### Hardware considerations

#### Argument value vs. configuration

These boards have two 8536 chips, which have both counter and digital I/O and interrupt vectoring capabilities. The numbers stated for digital I/O apply when both chips are configured for the maximum number of digital devices. The numbers stated for counter I/O apply when both chips are configured for the maximum number of counter devices.

## PPIO-CTR06

### Counter I/O

#### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

#### CounterNum

1 to 6

### Digital I/O

#### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

#### PortNum

AUXPORT\*

#### Datavalue

0 to 15, or 0 to 255, depending on jumper setting

#### BitNum

0 to 3, or 0 to 7, depending on jumper setting

\* AUXPORT is not configurable for this board.

## QUAD02 Series and QUAD04 Series

The QUAD02 Series includes the following hardware:

- PCM-QUAD02
- CIO-QUAD02

The QUAD04 Series includes the following hardware:

- PCI-QUAD04
- CIO-QUAD04

The QUAD02 Series and QUAD04 Series support the following UL and UL for .NET features.

## Counter I/O

### Functions

UL: [cbC7266Config\(\)](#), [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCLoad\(\)](#), [cbCLoad32\(\)](#), [cbCStatus\(\)](#)

UL for .NET: [C7266Config\(\)](#), [CIn\(\)](#), [CIn32\(\)](#), [CLoad\(\)](#), [CLoad32\(\)](#), [CStatus\(\)](#)

### CounterNum

PCM-QUAD02, CIO-QUAD02: 1 to 2

CIO-QUAD04, PCI-QUAD04: 1 to 4

### RegNum

UL: COUNT1, COUNT2, PRESET1, PRESET2, PRESCALER1, PRESCALER2

UL for .NET: QuadCount1, QuadCount2, QuadPreset1, QuadPreset2, QuadPreScaler1, QuadPreScaler2

CIO-QUAD04, PCI-QUAD04 also support:

UL: COUNT3, COUNT4, PRESET3, PRESET4, PRESCALER3, PRESCALER4

UL for .NET: QuadCount3, QuadCount4, QuadPreset3, QuadPreset4, QuadPreScaler3, QuadPreScaler4

### LoadValue

When using [cbCLoad32\(\)](#) or [CLoad32\(\)](#) to load the COUNT# or PRESET# registers, values up to 16.78 million ( $2^{24} - 1$ ) may be loaded. Values using [cbCLoad\(\)](#) and [CLoad\(\)](#) are limited to 65,535 ( $2^{16} - 1$ ). (Refer to the [Basic signed integers](#) discussion in the "Introduction: Counter Boards" topic). When loading the PRESCALER# register, values may be from 0 to 255.

Digital Filter Clock frequency = 10 megahertz (MHz) / LoadValue + 1.

## Hardware Considerations

### Loading and reading 24-bit values

The QUAD series boards feature a 24-bit counter. You can use the [cbCIn\(\)/CIn\(\)](#) and [cbCLoad\(\)/CLoad\(\)](#) functions for counts that are less than 16 bits (65,535), or use the [cbCIn32\(\)/CIn32\(\)](#) and [cbCLoad32\(\)/CLoad32\(\)](#) functions for any number supported by the LS7266 counter (24 bits = 16777216).

### Cascading counters (PCI-QUAD04 only)

The PCI-QUAD04 can be set up for cascading counters. By setting the appropriate registers, the following configurations are possible:

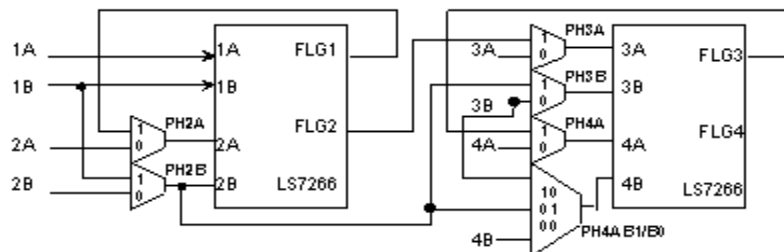
- Four 24-bit counters, or
- Two 48-bit counters, or
- One 24-bit and one 72-bit counters, or
- One 96-bit counter

The OUTPUT pins of a counter are directed to the next counter by setting FLG1 to CARRY/BORROW, and FLG2 to UP/DOWN. Bits 3 and 4 of the IOR Register control are set to 1,0 to accomplish this.

You can set these bits by using the functions [cbC7266Config\(BoardNum, CounterNum, Quadrature, CountingMode, DataEncoding, IndexMode, InvertIndex, FlagPins, and GateEnable\)](#). When using the Universal Library for .NET, use the [C7266Config\(\)](#) method.

The constant CARRYBORROW\_UPDOWN (value of 3) is used for the FlagPins parameter.

The IOR register cannot be read. However, you can read the values of the BADR2+9 register. The value for Base 2 can be determined by looking at the resources used by the board. The 8-bit region is BADR2. The BADR+9 register contains values for PhxA and PhxB, for x = 1 to 4 to identify counters. The diagram below indicates the routing of the FLG pins depending on the value of PhxA and PhxB. The actual values of the BADR2+9 register are shown below:



Register BADR2 + 9 D0-D6							
	PH2A	PH2B	PH3A	PH3B	PH4A	PH4B1/PH4B0	Value
<b>Case1:</b> (4) 24-bit counters (1/2/3/4)	0	0	0	0	0	0,0	0.0
<b>Case2:</b> (2) 48-bit counters (1-2/3-4)	1	1	0	0	1	1,0	53
<b>Case3:</b> (1) 24-bit counter and (1) 72-bit counter (1/2-3-4)	0	0	1	1	1	0,1	3C
<b>Case4:</b> (1) 96-bit counter (1-2-3-4)	1	1	1	1	1	0,1	3F
Defaults to 0x00 (no inter-counter connections).							

Examples	
Case 1:	<b>(4) 24-bit counters (1/2/3/4)</b> cbC7266Config(0,1,0,0,2,0,0,1,0) cbC7266Config(0,2,0,0,2,0,0,1,0) cbC7266Config(0,3,0,0,2,0,0,1,0) cbC7266Config(0,4,0,0,2,0,0,1,0)
Case 2:	<b>(2) 48-bit counters (1-2/3-4)</b> cbC7266Config(0,1,0,0,2,0,0,3,0) cbC7266Config(0,2,0,0,2,0,0,1,0) cbC7266Config(0,3,0,0,2,0,0,3,0) cbC7266Config(0,4,0,0,2,0,0,1,0)
Case 3:	<b>(1) 24-bit counter and (1) 72-bit counter (1/2-3-4)</b> cbC7266Config(0,1,0,0,2,0,0,1,0) cbC7266Config(0,2,0,0,2,0,0,3,0) cbC7266Config(0,3,0,0,2,0,0,3,0) cbC7266Config(0,4,0,0,2,0,0,1,0)
Case 4:	<b>(1) 96-bit counter (1-2-3-4)</b> cbC7266Config(0,1,0,0,2,0,0,3,0) cbC7266Config(0,2,0,0,2,0,0,3,0) cbC7266Config(0,3,0,0,2,0,0,3,0) cbC7266Config(0,4,0,0,2,0,0,1,0)
The actual value of the BADR+9 register is not set until the cbCLoad()/CLoad() command is called.	

## Counter4 setting

Setting Counter4 to CARRYBORROW-UPDOWN is NOT VALID.



# USB-QUAD08

The USB-QUAD08 supports the following UL and UL for .NET features.

## Counter I/O

### Functions

UL: [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCIn64\(\)](#), [cbCLoad\(\)](#), [cbCLoad32\(\)](#), [cbCLoad64\(\)](#), [cbCConfigScan\(\)](#), [cbCInScan\(\)](#), [cbCClear\(\)](#)

UL for .NET: [CIn\(\)](#), [CIn32\(\)](#), [CIn64\(\)](#), [CLoad\(\)](#), [CLoad32\(\)](#), [CLoad64\(\)](#), [CConfigScan\(\)](#), [CInScan\(\)](#), [CClear\(\)](#)

**Note:** Counters on this device are zero-based (the first counter number is "0").

### CounterNum

0 to 7

### RegNum

MAXLIMITREG0 to MAXLIMITREG7

### Options

BACKGROUND, CONTINUOUS, CTR32BIT, CTR48BIT, EXTCLOCK, EXTTRIGGER, HIGHRESRATE

### LoadValue

When using [cbCLoad64\(\)](#) to load the MAXLIMIT register, values up to  $2^{48} - 1$  may be loaded. Values using [cbCLoad32\(\)](#) and [CLoad32\(\)](#) are limited to 4,294,967,295 ( $2^{32} - 1$ ). Values using [cbCLoad\(\)](#) and [CLoad\(\)](#) are limited to 65,535 ( $2^{16} - 1$ ). Refer to the [Visual Basic signed integers](#) discussion in the "Introduction: Counter Boards" topic for more information.

## Digital I/O

### Configuration

#### Functions

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

#### PortNum

AUXPORT\*

### Port I/O

#### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

#### PortNum

AUXPORT\*

#### DataValue

0 to 255

### Bit I/O

#### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

#### BitNum

0 to 7

## Timers

UL: [cbPulseOutStart\(\)](#), [cbPulseOutStop\(\)](#)

UL for .NET: [PulseOutStart\(\)](#), [PulseOutStop\(\)](#)

### TimerNum

0 and 1

### Frequency

0.01123 Hz to 5 MHz

## DutyCycle

0 to 1, non-inclusive

## PulseCount, Initial Delay, IdleState

Ignored

## Triggering

UL: [cbSetTrigger\(\)](#)

UL for .NET: [SetTrigger\(\)](#)

## TrigType

TRIGHIGH, TRIGLOW, TRIGPOSEDGE, TRIGNEGEDGE

## Hardware Considerations

### Loading and reading 16, 32, and 48-bit values

The USB-QUAD08 counters can be configured as 16-, 32-, or 48-bit.

For most situations, the 32-bit functions ([cbCLoad32\(\)/CLoad32\(\)](#) and [cbCIn32\(\)/CIn32\(\)](#)) are preferred for reading and writing the counter. While each of these functions are valid regardless of the configuration, keep in mind that the upper bits will be truncated when the data value is larger than the Count argument can represent. Refer to [32-bit values using a signed integer data type](#) for information on 32-bit values using unsigned integers.

The following functions can handle count values that are less than  $2^{16}$  (65,536) without truncating:

- [cbCIn\(\)/CIn\(\)](#)
- [cbCLoad\(\)/CLoad\(\)](#)

The following functions can handle count values that are less than  $2^{32}$  (4,294,967,296) without truncating:

- [cbCIn32\(\)/CIn32\(\)](#)
- [cbCLoad32\(\)/CLoad32\(\)](#)

The following functions can handle count values that are less than  $2^{48}$  (281,474,976,710,656) without truncating:

- [cbCIn64\(\)/CIn64\(\)](#)
- [cbCLoad64\(\)/CLoad64\(\)](#)

### Mapped channel

The [cbCConfigScan\(\)](#) MappedChannel argument and the [CConfigScan\(\)](#) mapCounter parameter are ignored for the USB-QUAD08. Use the device's Index input to gate, latch, decrement, or clear/reload a counter.

### Scanning

Synchronous reads of data can be accomplished using the [cbCInScan\(\)/CInScan\(\)](#) functions. However, keep in mind that the count value is set to 0 at the initiation of a scan. For this reason, the use of the Terminal Count outputs in conjunction with synchronous reads should be avoided for the USB-QUAD08. Instead, use the asynchronous functions [cbCIn32\(\)/CIn32\(\)](#) or the variations to read the counter without disruption of the Terminal Count pulse train.

### Edge detection

The [cbCConfigScan\(\)](#) EdgeDetection argument and the [CConfigScan\(\)](#) edgeDetection parameter is used to detect a rising or falling edge on the counter specified by the CounterNum argument.

For the USB-QUAD08, the standard arguments (CTR\_RISING\_EDGE/CTR\_FALLING\_EDGE) apply to the Phase A input. The polarity for the Phase B and Index inputs can also be set by using the bit fields as described below:

- 000 (CTR\_RISING\_EDGE): Phase A, Phase B, Index input
- 001 (CTR\_FALLING\_EDGE): Phase A input
- 010 (CTR\_FALLING\_EDGE): Phase B input
- 100 (CTR\_FALLING\_EDGE): Index input

These values can be combined using a bitwise Or operation.

### HIGHRESRATE

Specify the [cbCInScan\(\)/CInScan\(\)](#) HIGHRESRATE scan option to acquire data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel".

### Counter resolution

The counter resolution is set by default to 48-bits. Use the [CInScan\(\)](#) method's Ctr16Bit, Ctr32Bit, and Ctr48Bit scan options to

change the counter resolution.

### **Asynchronous reads**

The CConfigScan() method's Bit48 and EncoderModeBit48 counter mode options only affect counter resolution for asynchronous calls ([CIn\(\)](#), [CIn32\(\)](#), and [CIn64\(\)](#)).

The Bit48 and EncoderModeBit48 modes are recommended for use only with CIn64(). Using these mode options with CIn() and CIn32() are not very useful, since the value returned by CIn() is only 16 bits, and the value returned by CIn32() is only 32 bits. The effect is that the value returned by CIn() rolls over at 65,535, and the value returned by CIn32() rolls over at 4,294,967,295.

## USB-4300 Series

The USB-4300 Series includes the following hardware:

- USB-4301
- USB-4302
- USB-4303
- USB-4304

The USB-4300 Series supports the following UL and UL for .NET features.

## Counter I/O

### Functions

UL: [cbC9513Config\(\)](#), [cbC9513Init\(\)](#), [cbCStoreOnInt\(\)](#), [cbCFreqIn\(\)](#), [cbCIn\(\)](#), [cbCIn32\(\)](#), [cbCLoad\(\)](#), [cbCLoad32\(\)](#)

UL for .NET: [C9513Config\(\)](#), [C9513Init\(\)](#), [CStoreOnInt\(\)](#), [CFreqIn\(\)](#), [CIn\(\)](#), [CIn32\(\)](#), [CLoad\(\)](#), [CLoad32\(\)](#)

### CounterNum

USB-4301 and USB-4302: 1 through 5

USB-4303 and USB-4304: 1 through 5, and 6 through 10

### RegNum

USB-4301 and USB-4302:

LOADREG1 – 5, HOLDREG1 – 5, ALARM1CHIP1, ALARM2CHIP1

USB-4303 and USB-4304:

LOADREG1 – 10, HOLDREG1 – 10, ALARM1CHIP1, ALARM1CHIP2, ALARM2CHIP1, ALARM2CHIP2

### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### ChipNum

USB-4301: 1

USB-4302: 1

USB-4303: 1, 2

USB-4304: 1, 2

### FOUT Source

CTRINPUT1 – 5, GATE1 – 5, FREQ1 – 5

These values refer to the sources on a particular 9513 chip, so are limited to the sources on that particular chip. For example, to set the source to the input for counter 6, use CTRINPUT1 (the first counter on the second 9513 chip).

### CountSource

- TCPREVCTR
- CTRINPUT1 – 5
- GATE1 – 5
- FREQ1 – 5

These values refer to the sources on a particular 9513 chip, so are limited to the sources on that particular chip. For example, to set the source to the input for counter 6, use CTRINPUT1 (the first counter on the second 9513 chip). Likewise for the TCPREVCTR value: when applied to the first counter on a chip (for counter 6, the "previous counter" is counter 5 on that chip (for this example, counter 10).

## Digital I/O

### Port I/O

#### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

#### PortNum

AUXPORT\*

DataValue

0 to 255

\* AUXPORT is not configurable for these boards

## Bit I/O

### Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

BitNum

0 to 7

\* AUXPORT is not configurable for these boards.

## Event Notification

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### EventType

ON\_EXTERNAL\_INTERRUPT

### EventParameter

LATCH\_DI, LATCH\_DO

LATCH\_DI can only be used with [cbDIn\(\)](#) and [cbDBitIn\(\)](#). LATCH\_DO can only be used with [cbDOut\(\)](#) and [cbDBitOut\(\)](#).

## Hardware considerations

### Clock input frequency

The clock speed is configurable with InstaCal for 1 MHz, 1.67 MHz, 3.33 MHz, or 5 MHz.

### Event notification

ON\_EXTERNAL\_INTERRUPT cannot be used with [cbCStoreOnInt\(\)](#) or [CStoreOnInt\(\)](#).

### Interrupt Input pin

You can configure the Interrupt Input pin (**INT**) with InstaCal to trigger off rising or falling edge inputs. You can program this pin to perform the following tasks:

- Send an event notification to the computer. The transfer rate is system-dependent.
- Latch digital input data.
- Latch digital output data.
- Save the current value of a counter. You can configure this option for each counter individually.

### Digital bit latching

Digital input bit latching is supported by [cbDIn\(\)](#) and [cbDBitIn\(\)](#). Digital output bit latching is supported by [cbDOut\(\)](#) and [cbDBitOut\(\)](#).

- Use the EventParam option LATCH\_DI with [cbDIn\(\)](#) and [cbDBitIn\(\)](#) to return the data that was latched in at the most recent interrupt edge. The current value of the digital inputs (0 or 1) is read and stored. The stored value is updated when an active edge occurs on the Interrupt Input pin (**INT**).

There is a latency period between when an active interrupt edge occurs on the **INT** pin and when the action triggered by that interrupt occurs. This latency can be as long as 100  $\mu$ s, but typically varies from about 9  $\mu$ s to about 40  $\mu$ s between interrupts.

- Use the EventParam option LATCH\_DO with [cbDOut\(\)](#) and [cbDBitOut\(\)](#) to latch out the data most recently written to the device. The digital outputs are not set to the value written until an active edge occurs on the Interrupt Input pin (**INT**).

## Digital Input Hardware

To maximize the performance of the digital input function calls, refer to the [82C55 data sheet](#). This document is also available in the *Documents* subdirectory where the UL is installed (C:\Program files\Measurement Computing\DAQ by default). You can also refer to the 8536 data sheet, although this document is not installed with the Universal Library.



## CIO-DI Series and PC104-DI48

The CIO-DI Series includes the following hardware:

- CIO-DI48
- CIO-DI96
- CIO-DI192

This topic also includes the PC104-DI48.

The CIO-DI Series and PC104-DI48 support the following UL and UL for .NET features.

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDBitIn\(\)](#)

UL for .NET: [DIn\(\)](#), [DBitIn\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

For DI48, DI96 and DI192, the following argument values are also valid:

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

For DI96 and DI192, the following argument values are also valid:

THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH

FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

For DI192, the following argument values are also valid:

FIFTHPORTA through EIGHTHPORTCH

### DataValue

0 to 255 for PORTA or PORTB,

0 to 15 for PORTCL or PORTCH

### BitNum

0 to 23 for FIRSTPORTA

For DI48, DI96 and DI192, the following argument value is also valid:

24 to 47 using FIRSTPORTA

For DI96 and DI192, the following argument values are also valid:

48 to 95 using FIRSTPORTA

For DI192, the following argument value is also valid:

96 to 191

## CIO-DISO48

The CIO-DISO48 supports the following UL and UL for .NET features.

### Digital I/O

#### Functions

UL: [cbDIn\(\)](#), [cbDBitIn\(\)](#)

UL for .NET: [DIn\(\)](#), [DBitIn\(\)](#)

#### PortNum

FIRSTPORTA, SECONDPORTA, THIRDPORTA, FOURTHPORTA, FIFTHPORTA, SIXTHPORTA

#### DataValue

0 to 255

#### BitNum

0 to 47 using FIRSTPORTA



# Digital Input/Output Hardware

## Basic signed integers

When reading or writing ports that are 16-bits wide, be aware of the following issue using signed integers, as you are forced to do when using Basic:

- With some devices, the AUXPORT digital ports are set up as one 16-bit port. When using [cbDOut\(\)](#) or [DOut\(\)](#), the digital values are written as a single 16-bit word. Using signed integers, writing values above 0111 1111 1111 1111 (32,767 decimal) can be confusing. The next increment, 1000 0000 0000 0000 has a decimal value of -32,768. Using signed integers, this is the value that you would use for turning on the MSB only. The value for all bits on is -1. Keep this in mind if you are using Basic, since Basic does not supply unsigned integers (values from 0 to 65,536).

To maximize the performance of the digital I/O function calls, refer to the [82C55 data sheet](#). This document is also available in the *Documents* subdirectory where the UL is installed (C:\Program files\Measurement Computing\DAQ by default). You can also refer to the 8536 data sheet, although this document is not installed with the Universal Library.



## Digital ports and corresponding bit numbers (82C55-based hardware and emulations)

The following table lists the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

cbDConfigPort() port reference	cbDIn(), cbDOut() port reference	Values	cbDBitIn(), cbDBitOut() port reference	Bit Number
FIRSTPORTA	FIRSTPORTA	0-255	FIRSTPORTA	0 to 7
FIRSTPORTB	FIRSTPORTB	0-255	FIRSTPORTA	8 to 15
FIRSTPORTCL	FIRSTPORTCL	0-15	FIRSTPORTA	16 to 19
FIRSTPORTCH	FIRSTPORTCH	0-15	FIRSTPORTA	20 to 23
SECONDPORTA	SECONDPORTA	0-255	FIRSTPORTA	24 to 31
SECONDPORTB	SECONDPORTB	0-255	FIRSTPORTA	32 to 39
SECONDPORTCL	SECONDPORTCL	0-15	FIRSTPORTA	40 to 43
SECONDPORTCH	SECONDPORTCH	0-15	FIRSTPORTA	44 to 47
THIRDPORTA	THIRDPORTA	0-255	FIRSTPORTA	48 to 55
THIRDPORTB	THIRDPORTB	0-255	FIRSTPORTA	56 to 63
THIRDPORTCL	THIRDPORTCL	0-15	FIRSTPORTA	64 to 67
THIRDPORTCH	THIRDPORTCH	0-15	FIRSTPORTA	68 to 71
FOURTHPORTA	FOURTHPORTA	0-255	FIRSTPORTA	72 to 79
FOURTHPORTB	FOURTHPORTB	0-255	FIRSTPORTA	80 to 87
FOURTHPORTCL	FOURTHPORTCL	0-15	FIRSTPORTA	88 to 91
FOURTHPORTCH	FOURTHPORTCH	0-15	FIRSTPORTA	92 to 95
FIFTHPORTA	FIFTHPORTA	0-255	FIRSTPORTA	96 to 103
FIFTHPORTB	FIFTHPORTB	0-255	FIRSTPORTA	104 to 111
FIFTHPORTCL	FIFTHPORTCL	0-15	FIRSTPORTA	112 to 115
FIFTHPORTCH	FIFTHPORTCH	0-15	FIRSTPORTA	116 to 119
SIXTHPORTA	SIXTHPORTA	0-255	FIRSTPORTA	120 to 127

SIXTHPORTB	SIXTHPORTB	0-255	FIRSTPORTA	128 to 135
SIXTHPORTCL	SIXTHPORTCL	0-15	FIRSTPORTA	136 to 139
SIXTHPORTCH	SIXTHPORTCH	0-15	FIRSTPORTA	140 to 143
SEVENTHPORTA	SEVENTHPORTA	0-255	FIRSTPORTA	144 to 151
SEVENTHPORTB	SEVENTHPORTB	0-255	FIRSTPORTA	152 to 159
SEVENTHPORTCL	SEVENTHPORTCL	0-15	FIRSTPORTA	160 to 163
SEVENTHPORTCH	SEVENTHPORTCH	0-15	FIRSTPORTA	164 to 167
EIGHTHPORTA	EIGHTHPORTA	0-255	FIRSTPORTA	168 to 175
EIGHTHPORTB	EIGHTHPORTB	0-255	FIRSTPORTA	176 to 183
EIGHTHPORTCL	EIGHTHPORTCL	0-15	FIRSTPORTA	184 to 187
EIGHTHPORTCH	EIGHTHPORTCH	0-15	FIRSTPORTA	188 to 191

## Notes

- For devices that support AUXPORT, in general only bit numbers 0 to 7 apply. Refer to board-specific information for details on the number of bits your hardware supports.
- For devices that support [synchronous I/O scanning](#), such as the USB-2500 Series, PORTC is configured as one 8-bit port (PORTCH and PORTCL are not supported). Refer to board-specific information for details on which ports are supported by your hardware.

## AC5 Series

The AC5 Series includes the following hardware:

- CIO-DUAL-AC5
- PCI-DUAL-AC5
- PCI-QUAD-AC5
- PC104-AC5

The AC5 Series supports the following UL and UL for .NET features.

## Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

DataValue

0 to 255 using FIRSTPORTA or FIRSTPORTB

0 to 15 using FIRSTPORTCL or FIRSTPORTCH

BitNum

0 to 23 using FIRSTPORTA

### PortNum

DUAL-AC5 and QUAD-AC5 boards also support:

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

DataValue

0 to 255 using SECONDPORTA or SECONDPORTB

0 to 15 using SECONDPORTCL or SECONDPORTCH

BitNum

0 to 47 using FIRSTPORTA

### PortNum

QUAD-AC5 boards also support:

THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH, FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

DataValue

0 to 255 using THIRDPORTA or THIRDPORTB

0 to 15 using THIRDPORTCL or THIRDPORTCH

BitNum

0 to 95 using FIRSTPORTA

## CIO-PDMA16, CIO-PDMA32

The CIO-PDMA16, CIO-PDMA32 support the following UL and UL for .NET features.

### Digital I/O

#### Functions

UL: [cbDInScan\(\)](#), [cbDOutScan\(\)](#), [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DInScan\(\)](#), [DOutScan\(\)](#), [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

#### PortNum

AUXPORT, FIRSTPORTA, FIRSTPORTB

#### DataValue

0 to 7 using AUXPORT (only [cbDOut\(\)](#)/[DOut\(\)](#) are supported)

0 to 255 using FIRSTPORTA and FIRSTPORTB

0 to 65,535 using WORDXFER FIRSTPORTA

#### BitNum

0 to 2 using AUXPORT (only [cbDBitOut\(\)](#)/[DBitOut\(\)](#) are supported)

0 to 15 using FIRSTPORTA

#### Rate

CIO-PDMA16: 125 kWords

CIO-PDMA32: 750 kWords

#### Options

BACKGROUND, CONTINUOUS, EXTCLOCK, WORDXFER

### Hardware considerations

#### Digital I/O pacing

Hardware pacing, external or internal clock supported.

## DIO Series (Excluding USB)

The DIO Series (Excluding USB) includes the following hardware:

- CIO-DIO24, CIO-DIO24H, CIO-DIO48, CIO-DIO48H, CIO-DIO96, CIO-DIO192
- PCI-DIO24, PCI-DIO24H, PCI-DIO48H, PCI-DIO96, PCI-DIO96H, PCI-DIO24/LP, PCI-DIO24/S
- CPCI-DIO24H, CPCI-DIO48H, CPCI-DIO96H
- PPIO-DIO24
- PC-CARD-DIO48
- PC104-DIO48

**Notes:** Additional topics are available that document DIO hardware not listed above. If your DIO hardware is not listed above, refer to the following list:

- USB-DIO24/37, USB-DIO24H/37, USB-1024LS, and USB-1024HLS: refer to the topic "[USB-1024 Series and USB-DIO24 Series](#)".
- USB-DIO96H and USB-DIO96H/50: refer to the topic "[USB-DIO96H \(formerly USB-1096HFS\)](#)".
- CIO-DIO24/CTR3, PCI-DIO24H/CTR3, PCM-D24/CTR3, and PC-CARD-D24/CTR3: refer to the topic "[DIO24/CTR3 Series and D24/CTR3 Series](#)".
- PCI-DIO48/CTR15: refer to the topic "[PCI-DIO48/CTR15](#)".
- PCIe-DIO24 and PCIe-DIO96H: refer to the topic "[PCIe-DIO24 and PCIe-DIO96H](#)".

The DIO Series (Excluding USB) supports the following UL and UL for .NET features.

## Digital I/O

Hardware in this series either have an 82C55 chip or are based on 82C55, mode 0 emulation. Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

DIO48, DIO48H, DIO96, and DIO192 also support:

- SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

DIO96 and DIO192 also support:

- THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH
- FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

DIO192 also supports:

- FIFTHPORTA through EIGHTHPORTCH

### DataValue

0 to 255 using PORTA or PORTB

0 to 15 using PORTCL or PORTCH

### BitNum

0 to 23 using FIRSTPORTA

For DIO48, DIO48H, DIO96, and DIO192, the following values are also valid:

- 24 to 47 using FIRSTPORTA

For DIO96, and DIO192, the following argument values are also valid:

- 48 to 95 using FIRSTPORTA

For DIO192, the following argument values are also valid:

- 96 to 191

## Event Notification

CIO- and PCI- DIO24 and DIO24H, PCI-DIO24/LP and PCI-DIO24/S only.

### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

### Event Types

ON\_EXTERNAL\_INTERRUPT/OnExternalInterrupt

## Hardware Considerations

### Event notification

DIO Series boards that support event notification only support external rising edge interrupts.

## Notes

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions and methods.

## DIO24/CTR3 Series and D24/CTR3 Series

The DIO24/CTR3 Series and D24/CTR3 Series includes the following hardware:

- PCI-DIO24H/CTR3
- CIO-DIO24/CTR3
- PC-CARD-D24/CTR3
- PCM-D24/CTR3

The DIO24/CTR3 Series and D24/CTR3 Series supports the following UL and UL for .NET features.

### Digital I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

#### DataValue

0 to 255 using FIRSTPORTA or FIRSTPORTB

0 to 15 using FIRSTPORTCL or FIRSTPORTCH

#### BitNum

0 to 23 using FIRSTPORTA

### Counter I/O

#### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

#### CounterNum

1 to 3

#### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

#### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Event notification

CIO-DIO24/CTR3 and PC-CARD-D24/CTR3 only.

#### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

#### Event types

ON\_EXTERNAL\_INTERRUPT/OnExternalInterrupt

### Hardware considerations

#### Counter configuration

Counter source functions are programmable using InstaCal.

## PCI-DIO48/CTR15

The PCI-DIO48/CTR15 board supports the following UL and UL for .NET features.

### Digital I/O

#### Functions

UL: [cbDOut\(\)](#), [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DOut\(\)](#), [DIn\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#), [DConfigPort\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTC, FIRSTPORTCH

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

#### DataValue

0 to 255 using PORTA or PORTB

0 to 15 using PORTCL or PORTCH

#### BitNum

0 to 47 using FIRSTPORTA

### Counter I/O

#### Functions

UL: [cbC8254Config\(\)](#), [cbCIn\(\)](#), [cbCLoad\(\)](#)

UL for .NET: [C8254Config\(\)](#), [CIn\(\)](#), [CLoad\(\)](#)

#### CounterNum

1 to 15

#### Config

HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE, HARDWARESTROBE

#### LoadValue

0 to 65,535 (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

### Event notification

#### Functions

UL: [cbEnableEvent\(\)](#), [cbDisableEvent\(\)](#)

UL for .NET: [EnableEvent\(\)](#), [DisableEvent\(\)](#)

#### Event types

ON\_EXTERNAL\_INTERRUPT/OnExternalInterrupt



## PCIe-DIO24 and PCIe-DIO96H

The PCIe-DIO24 and PCIe-DIO96H support the following UL and UL for .NET features.

### Digital I/O

- Based on 82C55, mode 0 emulation. Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Configuration

Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

The PCIe-DIO96H also supports:

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH

FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

#### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

The PCIe-DIO96H also supports:

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH

FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

DataValue

PCIe-DIO24:

0 to 15 for FIRSTPORTCL or FIRSTPORTCH

0 to 255 for FIRSTPORTA or FIRSTPORTB

PCIe-DIO96H:

0 to 15 for FIRSTPORTCL, SECONDPORTCL, THIRDPORTCL, FOURTHPORTCL, FIRSTPORTCH, SECONDPORTCH, THIRDPORTCH, FOURTHPORTCH

0 to 255 for FIRSTPORTA, SECONDPORTA, THIRDPORTA, FOURTHPORTA, FIRSTPORTB, SECONDPORTB, THIRDPORTB, FOURTHPORTB

#### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

PCIe-DIO24: 0 to 23 using FIRSTPORTA

PCIe-DIO96H: 0 to 95 using FIRSTPORTA

## Hardware considerations

### Pull-up/down resistor configuration

Each digital port has an associated resistor. You set the up/down configuration of each port's resistor with InstaCal. Configuration options are stored in non-volatile memory in EEPROM, and are loaded on power up.

### Notes

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions and methods.

## PDISO8 Series and PDISO16 Series

The PDISO8 Series includes the following hardware:

- CIO-PDISO8
- PCI-PDISO8
- PC104-PDISO8
- USB-PDISO8
- USB-PDISO8/40

The PDISO16 Series includes the following hardware:

- E-PDISO16
- PCI-PDISO16
- CIO-PDISO16

The PDISO8 and PDISO16 Series supports the following UL and UL for .NET features.

### Digital I/O

#### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AUXPORT

DataValue

PDISO8: 0 to 255 for AUXPORT

PDISO16: 0 to 65,535 for AUXPORT (Refer to [16-bit values using a signed integer data type](#) for information on 16-bit values using unsigned integers.)

#### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

PDISO8: 0 to 7 on AUXPORT

PDISO16: 0 to 15 on AUXPORT

### Miscellaneous

USB-PDISO8, USB-PDISO8/40, and E-PDISO16 only.

#### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the USB LED on the USB module to blink, and the LINK LED on the Ethernet module to blink.

When you have several modules connected to the computer, use this function to identify a particular module by making its LED blink.

### Hardware Considerations

#### Establishing and requesting control of an E-PDISO16

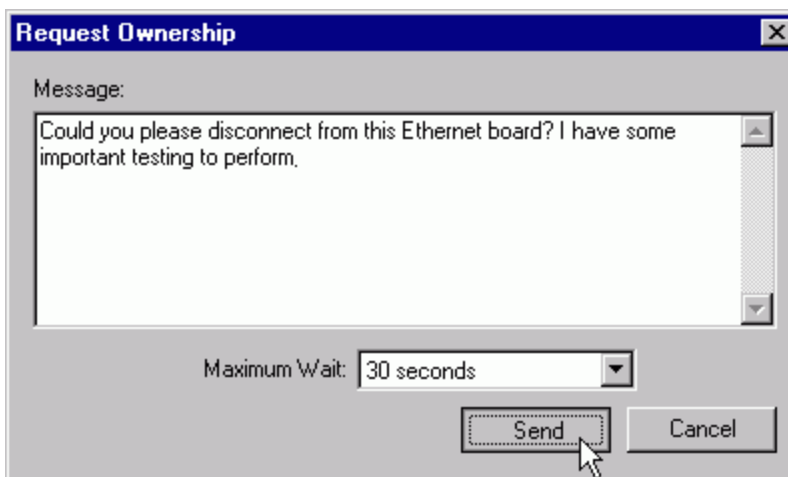
Through InstaCal, you can configure the system to automatically attempt to establish control over the E-PDISO16 when an application starts up. To do this, check the "Try to acquire ownership on application startup" option on InstaCal's **Ethernet Settings** tab. Note that only one computer should have this option selected; otherwise, two or more computers might compete for control over the E-PDISO16. To manually request control over the E-PDISO16, press the **Request Ownership** button on the Ethernet Settings tab.

Only one computer can establish control over an E-PDISO16 at a time. Additional computers that contact the device can only query the state of the device and its ports. The name of the computer with control over the E-PDISO16 appears in the Device Owner property on the Ethernet Settings tab.

### Sending a request for control of an E-PDISO16

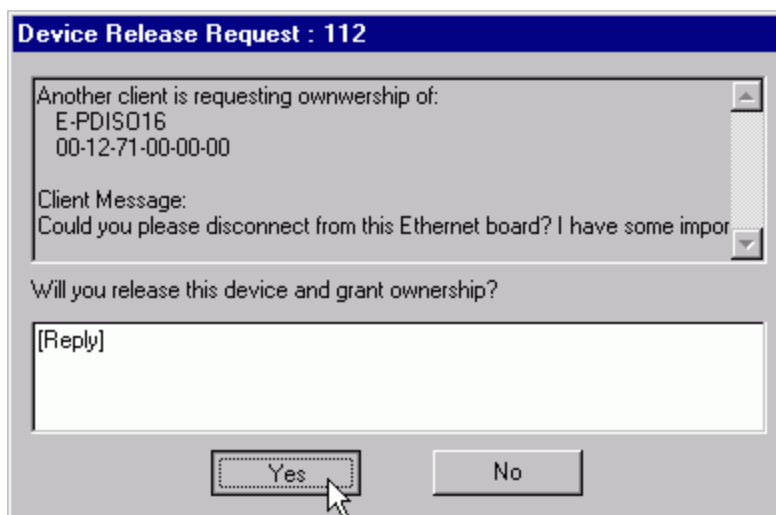
If another computer already has control over E-PDISO16 when you connect to it, you can send a message to the controlling computer. Do the following.

1. From InstaCal's main window, double-click on the E-PDISO16.
2. From the **Ethernet Settings** tab, click on the **Request Ownership** button.
3. On the **Request Ownership** dialog, enter your message (up to 256 characters). Press **Ctrl-Enter** to go to a new line.
4. You can set how long the message is displayed on the computer that controls the E-PDISO16 from the **Maximum Wait** drop-down list box.
5. Click on the **Send** button to send the message.



### Receiving a request for control of an E-PDISO16

If your computer controls an E-PDISO16 and you receive a message from another person requesting control of the device, the message shows on your screen for the time set in the **Maximum Wait** drop-down list.



- **Yes:** Click on **Yes** to give up ownership/control over the network device.

The computer automatically disconnects from the network connection, and control over the device transfers to the computer that sent the message. The **Device Owner** property in InstaCal updates with the name of the computer that gained control of the device.

- **No:** Click on **No** when you do not agree to give up ownership or control over the network device.

When you click on a button, the message box and selected response displays on the computer that sent the message.

### Receiving a message

When a computer sends a message to the computer controlling the device, the message displays on the monitor of the controlling computer for the time specified by the **Time-out** value.

The message box has two buttons used to respond to the message. When you receive a message, enter a response in the message box and click on one of the following buttons.

- **Yes:** Click on **Yes** to give up ownership/control over the network device.

The computer automatically disconnects from the network connection, and control over the device transfers to the computer that sent the message. The **Device Owner** property in InstaCal updates with the name of the computer that gained control of the device.

- **No**: Click on **No** when you do not agree to give up ownership or control over the network device.

When you click on a button, the message box and selected response displays on the computer that sent the message.

## Switch & Sense 8/8

The Switch & Sense 8/8 supports the following UL and UL for .NET features.

### Digital I/O

#### Port I/O functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AUXPORT

DataValue

0 to 255 for AUXPORT

#### Bit I/O functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

### Miscellaneous

#### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink.

When you have several modules connected to the computer, use these functions to identify a particular module by making its LED blink.

## USB-1024 Series and USB-DIO24 Series

The USB-1024 Series includes the following hardware:

- USB-1024LS
- USB-1024HLS

The USB-DIO24 Series includes the following hardware:

- USB-DIO24/37
- USB-DIO24H/37

The USB-1024 and USB-DIO24 Series supports the following UL and UL for .NET features.

### Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Configuration functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

#### Port I/O functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

DataValue

0 to 15 for FIRSTPORTCL or FIRSTPORTCH

0 to 255 for FIRSTPORTA or FIRSTPORTB

#### Bit I/O functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 23 on FIRSTPORTA

### Counter I/O

#### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

#### CounterNum

1

#### Count

0 to  $2^{32}-1$  when reading the counter.

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter for this board to 0. No other values are valid.

The [Basic signed integers](#) guidelines apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#)

[\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

## RegNum

LOADREG1

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink.

When you have several USB devices connected to the computer, use these functions to identify a particular device by making its LED blink.

## Notes

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions and methods.



## USB-DIO96H (formerly USB-1096HFS) and USB-DIO96H/50

The USB-DIO96H (formerly USB-1096HFS) and USB-DIO96H/50 support the following UL and UL for .NET features.

### Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Configuration

Functions

UL: [cbDConfigPort\(\)](#)

UL for .NET: [DConfigPort\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH

FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

#### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH

FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

DataValue

0 to 15 for FIRSTPORTCL, SECONDPORTCL, THIRDPORTCL, FOURTHPORTCL, FIRSTPORTCH, SECONDPORTCH, THIRDPORTCH, FOURTHPORTCH

0 to 255 for FIRSTPORTA, SECONDPORTA, THIRDPORTA, FOURTHPORTA, FIRSTPORTB, SECONDPORTB, THIRDPORTB, FOURTHPORTB

#### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

0 to 95 on FIRSTPORTA

### Counter I/O

#### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although cbCIn() and CIn() are valid for use with this counter, cbCIn32() or CIn32() may be more appropriate, since the values returned may be greater than the data types used by cbCIn() and CIn() can handle.

\*\*cbCLoad(), cbCLoad32(), CLoad() and CLoad32() only accept Count=0. These functions are used to reset the counter.

#### CounterNum

1

## Count

0 to  $2^{32}-1$  when reading the counter.

The [Basic signed integers](#) guidelines apply when using `cbCIn()` or `CIn()` for values greater than 32,767 and when using `cbCIn32()` or `CIn32()` for values greater than 2,147,483,647.

0 when loading the counter.

`cbCLoad()` and `cbCLoad32()` / `CLoad()` and `CLoad32()` are only used to reset the counter for this board to 0. No other values are valid.

## RegNum

LOADREG1

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the USB LED to blink.

When you have several boards connected to the computer, use this function/method to identify a specific board by making its LED blink.

## Notes

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library digital I/O functions and methods.

## USB-SSR Series

The USB-SSR Series includes the following hardware:

- USB-SSR08
- USB-SSR24

The USB-SSR Series supports the following UL and UL for .NET features.

### Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

USB-SSR08: FIRSTPORTCL, FIRSTPORTCH

USB-SSR24: FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

DataValue

USB-SSR08: 0 to 15 for FIRSTPORTCL or FIRSTPORTCH

USB-SSR24: 0 to 255 for FIRSTPORTA or FIRSTPORTB, 0 to 15 for FIRSTPORTCL or FIRSTPORTCH

#### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

USB-SSR08: 16 to 23 on FIRSTPORTA

USB-SSR24: 0 to 23 on FIRSTPORTA

## Miscellaneous

#### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the USB LED on a USB device to blink.

When you have several modules connected to the computer, use these functions to identify a particular module by making its LED blink.

## Hardware considerations

#### Do not change the state of switches while a program is running

The USB-SSR Series devices have three onboard switches (labeled S1, S2, and S3) that are used to control the I/O direction, logic polarity, and pull-up/down state of output relays.

Do not change the state of any switches on a USB-SSR Series device while a program is running. The Universal Library stores the current state of each switch, and changing a switch setting while a program is running can cause unpredictable results.

#### Controlling relays on the USB-SSR08

USB-SSR Series hardware is bidirectional, so you first need to set the I/O direction of each relay module group using the onboard switch S1.

The USB-SSR08 has two 4-bit ports (FIRSTPORTCL or FIRSTPORTCH). Each port controls four relays.

- To read four relays at a time, call [DIn\(\)](#), and specify either FIRSTPORTCL or FIRSTPORTCH.

To read just one of the input modules, call [DBitIn\(\)](#), specify FIRSTPORTA and a bit number of 16 to 23.

- To control four relays at a time, call `DOut()`, specify either `FIRSTPORTCL` or `FIRSTPORTCH`, and send a data value between 0 and 15. To control all eight relays, make two consecutive `DOut()` calls.

To control just one of the eight relays, call `DBitOut()`, specify `FIRSTPORTA`, and send a data value of either 0 or 1 and a bit number of 16 to 23. Bits 16 through 23 map to relays 1 through 8 on the USB-SSR08.

The relays are controlled in this way to allow code migration without changes when switching from the older SSR-RACK08 board to the USB-SSR08.

## Digital Output Hardware

To maximize the performance of the digital output function calls, refer to the [82C55 data sheet](#). This document is also available in the *Documents* subdirectory where the UL is installed (C:\Program files\Measurement Computing\DAQ by default). You can also refer to the 8536 data sheet, although this document is not installed with the Universal Library.



## CIO-DO Series and PC104-DO48

The CIO-DO Series includes the following hardware:

- CIO-DO48H
- CIO-DO96H
- CIO-DO192H
- CIO-DO24DD
- CIO-DO48DD

This topic also includes:

- PC104-DO48H

CIO-DO Series and PC104-DO48 support the following UL and UL for .NET features.

## Digital I/O

### Functions

UL: [cbDOut\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DOut\(\)](#), [DBitOut\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

For DO48H, DO48DD, DO96H and DO192H, the following argument values are also valid:

SECONDPORTA, SECONDPORTB, SECONDPORTCL, SECONDPORTCH

For DO96H and DO192H, the following argument values are also valid:

THIRDPORTA, THIRDPORTB, THIRDPORTCL, THIRDPORTCH

FOURTHPORTA, FOURTHPORTB, FOURTHPORTCL, FOURTHPORTCH

For DO192H, the following argument values are also valid:

FIFTHPORTA through EIGHTHPORTCH

### DataValue

0 to 255 using PORTA or PORTB

0 to 15 using PORTCL or PORTCH

### BitNum

0 to 23 using FIRSTPORTA

For DO48H, DO48DD, DO96H and DO192H the following argument values are also valid:

24 to 47 using FIRSTPORTA

For DO96H and DO192H, the following argument values are also valid:

48 to 95 using FIRSTPORTA

For DO192H, the following argument values are also valid:

96 to 191

## CIO-RELAY Series

The CIO-RELAY Series includes the following hardware:

- CIO-RELAY08
- CIO-RELAY16
- CIO-RELAY16/M
- CIO-RELAY24
- CIO-RELAY32

The CIO-RELAY Series supports the following UL and UL for .NET features.

## Digital I/O

### Functions

UL: [cbDOut\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DOut\(\)](#), [DBitOut\(\)](#)

### PortNum

FIRSTPORTA

For CIO-RELAY16 & 16/M, the following argument value is also valid:

FIRSTPORTB

For CIO-RELAY24, the following argument value is also valid:

SECONDPORTA

For CIO-RELAY32, the following argument value is also valid:

SECONDPORTB

### DataValue

0 to 255

### BitNum

0 to 7 using FIRSTPORTA

For CIO-RELAY16 and CIO-RELAY16/M, the following argument values are also valid:

0 to 15 using FIRSTPORTA

For CIO-RELAY24, the following argument values are also valid:

0 to 23 using FIRSTPORTA

For CIO-RELAY32, the following argument values are also valid:

0 to 31 using FIRSTPORTA

## USB-ERB Series

The USB-ERB Series includes the following hardware:

- USB-ERB08
- USB-ERB24

The USB-ERB Series supports the following UL and UL for .NET features.

### Digital I/O

- Click here to display a table of the port numbers and corresponding bit numbers that are set by the digital I/O functions for hardware designed with the 82C55 chip or 82C55 emulation.

#### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

USB-ERB08: FIRSTPORTCL, FIRSTPORTCH

USB-ERB24: FIRSTPORTA, FIRSTPORTB, FIRSTPORTCL, FIRSTPORTCH

DataValue

USB-ERB08: 0 to 15 for FIRSTPORTCL or FIRSTPORTCH

USB-ERB24: 0 to 255 for FIRSTPORTA or FIRSTPORTB, 0 to 15 for FIRSTPORTCL or FIRSTPORTCH

#### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

FIRSTPORTA

BitNum

USB-ERB08: 16 to 23 on FIRSTPORTA

USB-ERB24: 0 to 23 on FIRSTPORTA

## Miscellaneous

#### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink.

When you have several modules connected to the computer, use these functions to identify a particular module by making its LED blink.

## Hardware considerations

#### Invert/non-invert switch (S1)

Do not change the state of the invert/non-invert switch (labeled S1) on a USB-ERB Series device while a program is running. The Universal Library stores the current state of this switch, and changing the switch setting while a program is running can cause unpredictable results.



### Controlling relays on the USB-ERB08

USB-ERB Series hardware are output-only boards (no inputs), so setting the port direction is not required. The USB-ERB08 has two 4-bit ports (FIRSTPORTCL or FIRSTPORTCH). Each port controls four relays.

- To control four relays at a time, call DOut(), specify either FIRSTPORTCL or FIRSTPORTCH, and send a data value between 0 and 15. To control all 8 relays, make two consecutive DOut() calls.
- To control just one of the eight relays, call DBitOut(), specify FIRSTPORTA, and send a data value of either 0 or 1 and a bit number of 16 to 23. Bits 16 through 23 map to relays 1 through 8 on the USB-ERB08.

The relays are controlled in this way to allow code migration without changes when switching from the older CIO-ERB08 board to the USB-ERB08.

## Expansion Hardware

Auto-detected expansion boards are automatically added to the InstaCal configuration when InstaCal is launched. The device properties are automatically adjusted to reflect the expansion properties. Auto-detected expansion boards are not shown as a separate device in the InstaCal device tree.

Manually configured expansion boards, such as the CIO-EXP series, are added to the InstaCal configuration by selecting the compatible board on the main InstaCal form, and selecting the **Add Exp Board...** option from the **Install** menu. Manually configured expansion boards are shown in the InstaCal device tree as a branch attached to the device to which it was added.

## AI-EXP32

The AI-EXP32 expansion board is used in combination with compatible parent boards, such as a [USB-2416 Series](#) board.

The AI-EXP32 supports all of the analog input and temperature input capabilities of the parent board, but expands the channel count as follows:

### Analog input

#### HighChan

32 to 63 in single-ended mode, 16 to 31 in differential mode.

### Temperature input

#### HighChan

8 to 31

### Hardware considerations

The parent board must be configured for differential inputs when using thermocouples.

# AI-EXP48

The AI-EXP48 expansion board is used in combination with compatible parent boards, such as a [USB-1616HS Series](#) board.

The AI-EXP48 supports all of the analog input and temperature input capabilities of the parent board, but expands the channel count as follows:

## Analog input

### HighChan

16 to 63 in single-ended mode, 8 to 31 in differential mode.

## Temperature input

### HighChan

8 to 31

## DAQ input

### ChanArray

ANALOG: 0 to 63 in single-ended mode, 0 to 31 in differential mode

CJC: 6 to 11

TC: 8 to 31

## Hardware considerations

### Associating CJC channels with TC channels

The TC channels must immediately follow their associated CJC channels in the channel array. For accurate thermocouple readings, associate CJC channels with the TC channels as listed in the following table:

CJC channels	TC channels
CJC6	TC8 through TC11
CJC7	TC12 through TC15
CJC8	TC16 through TC19
CJC9	TC20 through TC23
CJC10	TC24 through TC27
CJC11	TC28 through TC31

The parent board must be configured for differential inputs when using thermocouples.

TC inputs are supported by differential mode configuration only.

## CIO-EXP Series

The CIO-EXP Series includes the following hardware:

- CIO-EXP16
- CIO-EXP32
- CIO-EXP-BRIDGE
- CIO-EXP-GP
- CIO-EXP-RTD

The CIO-EXP Series supports the following UL and UL for .NET features.

## Temperature input

### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)

### Scale

CELSIUS, FAHRENHEIT, KELVIN

### HighChan

From 16 up to 255 for 16-channel boards, and from 64 up to 303 for 64-channel boards. The value depends on the number of boards connected and the application.

## Hardware considerations

CIO-EXP boards are used only in combination with an A/D board. Channel numbers for accessing the expansion boards begin at 16 for 8-channel and 16-channel boards, and 64 for 64-channel boards. To calculate the channel number (Chan) for access to CIO-EXP channels, use the following formula:

$$\text{Chan} = (\text{ADChan} * 16) + (16 + \text{MuxChan})$$

MuxChan is a number ranging from 0 to 15 that specifies the channel number on a particular bank of the expansion board. An EXP32 has two banks, so the channel numbers for one EXP32 connected to an A/D board would range from 16 to 47.

If all A/D channels are not used for CIO-EXP output, direct input to the A/D board is still available at these channels (using channel numbers below 16).

When CIO-EXP boards are used for temperature input, set the gain of the A/D board to a specific range:

- When using A/D boards with programmable gain, the range is set by the Universal Library.
- When using boards with switch-selectable gains, set the gain to a range that is dependent on the temperature sensor in use.

Generally, thermocouple measurements require the A/D board to be set to 5 V bipolar, if available, or 10 V bipolar if not. RTD sensors require a setting of 10 V unipolar, if available. These checks are made when you configure the system for temperature measurement using InstaCal.

## MEGA-FIFO

The MEGA-FIFO supports the following UL and UL for .NET features.

### Memory I/O

Only used in combination with a board which has **DT-Connect**.

### Functions

UL: [cbMemSetDTMode\(\)](#), [cbMemReset\(\)](#), [cbMemRead\(\)](#), [cbMemWrite\(\)](#), [cbMemReadPretrig\(\)](#)

UL for .NET: [MemSetDTMode\(\)](#), [MemReset\(\)](#), [MemRead\(\)](#), [MemWrite\(\)](#), [MemReadPretrig\(\)](#)

Some of these functions are integrated into the [cbAInScan\(\)](#) function and [AInScan\(\)](#) method. For example, if you use **MEGA-FIFO** with an A/D board and select the EXTMEMORY option, you would not have to call the [cbMemSetDTMode\(\)](#) and [cbMemWrite\(\)](#) functions or the [MemSetDTMode\(\)](#) and [MemWrite\(\)](#) methods.

### EXTMEMORY option

Continuous mode cannot be used with the EXTMEMORY/ExtMemory option.

## MetraBus Hardware

To use any MetraBus I/O board, a MetraBus interface board, such as the ISA-MDB64, PCI-MDB64 or a CPCI-MDB64, is required for the Universal Library functions to operate correctly. The interface board and a MetraBus cable provide the interface between the PC bus (ISA-, PC104-, PCI-, or CPCI) and the MetraBus I/O Boards.

The MetraBus system includes at least one controller board that communicates with real-world interface boards via a data bus (ribbon cable). The implication is that there will always be two or more boards in the system.

## MDB64 Series

The MDB64 Series includes the following hardware:

- ISA-MDB64
- PCI-MDB64
- PC104-MDB64

This series makes up the controller portion of the MetraBus system. The Universal Library contains no functions to communicate specifically with this board. The functions in the library are directed to the devices on the bus instead.

For example, if this board was installed in InstaCal as board 0, and an MII-32 was installed as board 1, the communication would be directed to board 1. To read digital bits from this configuration, the function would be [cbDBitIn\(\)/DBitIn\(\)](#), and the value of the BoardNum argument would be 1.



## MEM Series Relay

All MetraBus boards require a cable and an interface board (such as an ISA-, PC104-, or PCI- MDB64) to interface to the host computer system.

The MEM Series Relay boards include the following hardware:

- MEM-32
- MEM-8

The MEM Series Relay boards supports the following UL and UL for .NET features.

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDOut\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DBitIn\(\)](#), [DOut\(\)](#), [DBitOut\(\)](#)

### PortNum

FIRSTPORTA

For MEM-32, the following argument values are also valid:

- FIRSTPORTB
- SECONDPORTA, SECONDPORTB

### DataValue

0 to 255 for PORTA or PORTB

### BitNum

0 to 7 for FIRSTPORTA

For MEM-32, the following argument values are also valid:

- 0 to 31 for FIRSTPORTA

## Hardware Considerations

### Reading back the output state of a MEM Series relay

Although the MEM Series Relay is a digital output-only board, the state of the outputs can be read back using the UL functions [cbDIn\(\)](#) and [cbDBitIn\(\)](#), or the UL for .NET methods [DIn\(\)](#) and [DBitIn\(\)](#).

## MIO and MII Digital I/O

All MetraBus boards require a cable and an interface board (such as an ISA-, PC104-, or PCI- MDB64) to interface to the host computer system.

The MIO and MII Digital I/O boards include the following hardware:

- MII-32
- MIO-32

The MIO and MII Digital I/O boards support the following UL and UL for .NET features.

### Digital In (MII-32 only)

#### Functions

UL: [cbDIn\(\)](#), [cbDBitIn\(\)](#)

UL for .NET: [DIn\(\)](#), [DBitIn\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB, SECONDPORTA, SECONDPORTB

#### DataValue

0 to 255 for PORTA or PORTB

#### BitNum

0 to 31 for FIRSTPORTA

### Digital Out (MIO-32 only)

#### Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#), [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#), [DBitIn\(\)](#), [DBitOut\(\)](#)

#### PortNum

FIRSTPORTA, FIRSTPORTB

SECONDPORTA, SECONDPORTB

#### DataValue

0 to 255 for PORTA or PORTB

#### BitNum

0 to 31 for FIRSTPORTA

## Hardware considerations

### Functions/methods for reading back the MIO-32 output state

Although the **MIO-32** is a digital output-only board, the state of the outputs can be read back using the UL functions [cbDIn\(\)](#) and [cbDBitIn\(\)](#), or the UL for .NET methods [DBitIn\(\)](#) and [DIn\(\)](#).

## MSSR-24

All MetraBus boards require a cable and an interface board (such as an ISA-, PC104-, or PCI- MDB64) to interface to the host computer system.

The MSSR-24 supports the following UL and UL for .NET features.

## Digital I/O

### Functions

UL: [cbDIn\(\)](#), [cbDBitIn\(\)](#), [cbDOut\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DBitIn\(\)](#), [DOut\(\)](#), [DBitOut\(\)](#)

### PortNum

FIRSTPORTA, FIRSTPORTB

SECONDPORTA

### DataValue

0 to 255

### BitNum

0 to 24 using FIRSTPORTA

## Temperature Input Hardware

This section provides details on using temperature input devices in conjunction with the Universal Library and Universal Library for .NET.

For information on the CIO-EXP series, refer to [CIO-EXP Series](#) in the *Expansion Hardware* section.

## CIO-DAS-TEMP

The CIO-DAS-TEMP supports the following UL and UL for .NET features.

### Temperature input

#### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)

#### Options

NOFILTER

#### Scale

CELSIUS, FAHRENHEIT, KELVIN

#### HighChan

31

### Hardware considerations

#### Pacing Input

The rate of measurement is fixed at approximately 25 samples per second.

#### Selecting thermocouples

J, K, E, T, R, S or B type thermocouples may be selected using InstaCal.

## CIO-DAS-TC, PCI-DAS-TC

The CIO-DAS-TC and PCI-DAS-TC support the following UL and UL for .NET features.

### Temperature input

#### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)

#### Options

FILTER, NOFILTER

#### Scale

CELSIUS, FAHRENHEIT, KELVIN, VOLTS

#### HighChan

15

### Hardware considerations

#### Pacing input

The rate of measurement is fixed at approximately 25 samples per second.

#### Selecting thermocouples

J, K, E, T, R, S or B type thermocouples may be selected using InstaCal.

#### Open thermocouples

When using [cbTInScan\(\)](#) or [TInScan\(\)](#) with the DAS-TC, an open thermocouple error ([OPENCONNECTION](#)) on any of the channels will cause all data to be returned as -9999.0. This is a hardware limitation. If your application requires isolating channels with defective thermocouples attached and returning valid data for the remainder of the channels, use the [cbTIn\(\)](#) function or [TIn\(\)](#) method instead.

To read the voltage input of the thermocouple, select VOLTS for the Scale parameter in [cbTIn\(\)](#) and [cbTInScan\(\)](#) or [TIn\(\)](#) and [TInScan\(\)](#).

## USB-5200 Series

The USB-5200 Series includes the following devices:

- USB-5201
- USB-5203

The USB-5200 Series supports the following UL and UL for .NET features.

## Temperature input

### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)

### Options

N/A

### Scale

CELSIUS, FAHRENHEIT, KELVIN, NOSCALE\*

\*Refer to [NOSCALE](#) in the *Hardware considerations* section below for more information on this option.

### HighChan

0 to 7

## Digital I/O

### Configuration

Functions

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

### Port I/O

Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AUXPORT

DataValue

0 to 255 for AUXPORT

### Bit I/O

Functions

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

## Data Logging

UL: [cbLogConvertFile\(\)](#), [cbLogGetAIChannelCount\(\)](#), [cbLogGetAIInfo\(\)](#), [cbLogGetCJCInfo\(\)](#), [cbLogGetDIOInfo\(\)](#), [cbLogGetFileInfo\(\)](#), [cbLogGetFileName\(\)](#), [cbLogGetPreferences\(\)](#), [cbLogGetSampleInfo\(\)](#), [cbLogReadAIChannels\(\)](#), [cbLogReadCJCChannels\(\)](#), [cbLogReadDIOChannels\(\)](#), [cbLogReadTimeTags\(\)](#), [cbLogSetPreferences\(\)](#)

UL for .NET: [ConvertFile\(\)](#), [GetAIChannelCount\(\)](#), [GetAIInfo\(\)](#), [GetCJCInfo\(\)](#), [GetDIOInfo\(\)](#), [GetFileInfo\(\)](#), [GetFileName\(\)](#), [GetPreferences\(\)](#), [GetSampleInfo\(\)](#), [ReadAIChannels\(\)](#), [ReadJCChannels\(\)](#), [ReadDIOChannels\(\)](#), [ReadTimeTags\(\)](#), [SetPreferences\(\)](#)

The `cbLogGetCJCInfo()` function and the `GetCJCInfo()` method return the number of CJC temperature channels logged in the binary file ("0" or "2".)

The `cbLogGetDIOInfo()` function and the `GetDIOInfo()` method return the number of digital I/O channels logged in the binary file ("0" to "8".)

## Delimiter

Comma, Semicolon, Space, Tab

## LoggingUnits

Temperature, Raw

## Units

Celsius, Fahrenheit, Kelvin

## TimeFormat

TwelveHour, TwentyFourHour

## TimeZone

Local, GMT

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink.

When you have several modules connected to the computer, use these functions to identify a particular device by making its LED blink.

## Hardware considerations

### Logging and storing measurement data

Temperature measurements can be stored on a CompactFlash memory card (64 MB CF card included with hardware). Each sample is stored on the card as a binary file. You set up the following logging options through InstaCal:

- the temperature input channel(s) to log
- the channel format – raw data or temperature
- the start mode to begin a logging session
- the interval in seconds between samples
- the alarm conditions used to trigger the DIO bits

InstaCal provides further options for copying, converting, and deleting the binary files. You can access log data stored on the memory card with a CompactFlash reader, or by transferring the files from InstaCal to a computer for processing and conversion using the USB bus.

**Note:** A card reader is not required to access log data on a device installed with firmware 3.0 and later. A device with this firmware version appears in Windows Explorer as a removable drive from which you can directly access the log data.

### External power required for data logging

Due to processing limitations, data logging to the memory card is not allowed when the device is connected to an active USB bus on the computer. When operating as a data logger, disconnect the USB cable from the computer, and connect the external power supply shipped with the device.

**Note:** When using a self-powered hub, make sure it is attached to the PC USB port *before* connecting it to a USB-5200 Series device. If a powered hub is connected to the device first, it may be detected by the device as a power supply and go into logging mode.

### Configuring the DIO channels to generate alarms

USB-5200 Series devices provide eight independent temperature alarms. Each alarm controls an associated digital I/O channel as an alarm output. The input to each alarm is one of the temperature input channels. Use InstaCal to set up the temperature conditions used to activate an alarm, and the output state of the channel when activated (active high or low).

Digital channels that are configured as alarms will power up in an output state. When an alarm is activated, the associated DIO channel is driven to the output state defined by the alarm configuration.

The alarms function both in data logging mode and while attached to the USB port on a computer. The alarm configurations are stored in non-volatile memory on the device and are loaded on power up.



## Pacing temperature readings

The internal update rate for temperature measurement is a fixed value for these devices. If the UL reads the device faster than the internal update rate, temperature readings "repeat." For example, if using `cbTIn()/TIn()` in a loop to measure a rapidly changing temperature, readings do not change for several iterations of the loop, then "jump" when the update occurs internally.

## Using single sensors with `cbTInScan()`

When using single sensors for RTD or thermistor sensors, ignore the data for channels that do not have sensors attached. It is best to use `cbTIn()/TIn()` for these configurations, since you can select the channels to read. If you use `cbTInScan()/TInScan()`, however, data for all channels over the entire range of channels are returned. Since some channels are not populated in this configuration, you should filter out the data for channels without sensors.

## NOSCALE

Specify the NOSCALE option to retrieve raw data in volts or resistance from the device. When NOSCALE is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.

## Saving configuration settings

InstaCal allows you to save device configuration settings to a file, or to load a configuration from a previously saved file.

- Each USB-5203 channel can be configured to measure temperature data collected by one of five categories of temperature sensors: thermistors, thermocouples (one of eight types), RTDs, semiconductors, and *Disabled*.
- Each USB-5201 channel can be configured to measure temperature data collected by one of eight thermocouple types.

## Recommended warm-up time

Allow the device to warm-up for 30 minutes before taking measurements. This warm-up time minimizes thermal drift and achieves the specified rated accuracy of measurements.

For RTD and thermistor measurements, this warm-up time is also required to stabilize the internal current reference.

## Calibration

Any time you change the sensor category for the USB-5203, calibration is automatically performed by InstaCal. If the device is not warmed up when this occurs, calibrate the device again after the specified warm-up time.

## Error codes

- The UL returns -9999 when a value is out of range or an open connection is detected.
- The UL returns -9000 when the device is not ready. This usually occurs right after the device is powered up and calibration factors are being loaded.

## USB-TEMP Series, USB-TC Series

The USB-TEMP Series includes the following hardware:

- USB-TEMP
- USB-TEMP-AI

The USB-TC Series includes the following hardware:

- USB-TC
- USB-TC-AI

The USB-TEMP Series and the USB-TC Series support the following UL and UL for .NET features.

## Temperature input

### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)

### Options

N/A

### Scale

CELSIUS, FAHRENHEIT, KELVIN, NOSCALE\*

\*Refer to [NOSCALE](#) in the *Hardware considerations* section below for more information on this option.

### HighChan

USB-TEMP and USB-TC: 0 to 7

USB-TEMP-AI and USB-TC-AI: 0 to 3

## Voltage input (USB-TEMP-AI and USB-TC-AI)

### Functions

UL: [cbVIn\(\)](#)

UL for .NET: [VIn\(\)](#)

### Options

N/A

### HighChan

0 to 3

### Range

This board uses the Range set in InstaCal, so the Range argument to this function is ignored.

## Digital I/O

### Configuration

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

### Port I/O

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AUXPORT

DataValue

0 to 255 for AUXPORT

## Bit I/O

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

## Counter I/O (USB-TEMP-AI and USB-TC-AI)

### Functions

UL: [cbCIn\(\)](#)\*, [cbCIn32\(\)](#), [cbCLoad\(\)](#)\*\*, [cbCLoad32\(\)](#)\*\*

UL for .NET: [CIn\(\)](#)\*, [CIn32\(\)](#), [CLoad\(\)](#)\*\*, [CLoad32\(\)](#)\*\*

\*Although [cbCIn\(\)](#) and [CIn\(\)](#) are valid for use with this counter, [cbCIn32\(\)](#) or [CIn32\(\)](#) may be more appropriate, since the values returned may be greater than the data types used by [cbCIn\(\)](#) and [CIn\(\)](#) can handle.

\*\*[cbCLoad\(\)](#), [cbCLoad32\(\)](#), [CLoad\(\)](#) and [CLoad32\(\)](#) only accept Count=0. These functions are used to reset the counter.

### CounterNum

1

### Count

$2^{32}-1$  when reading the counter.

0 when loading the counter.

[cbCLoad\(\)](#) and [cbCLoad32\(\)](#) / [CLoad\(\)](#) and [CLoad32\(\)](#) are only used to reset the counter for this board to 0. No other values are valid.

The [Basic signed integers](#) guidelines in the *Introduction: Digital Input Output Boards* section apply when using [cbCIn\(\)](#) or [CIn\(\)](#) for values greater than 32,767 and when using [cbCIn32\(\)](#) or [CIn32\(\)](#) for values greater than 2,147,483,647.

### RegNum

1

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink.

When you have several devices connected to the computer, use this function/method to identify a particular device by making its LED blink.

## Hardware considerations

### Pacing temperature readings

The internal update rate for measurements is a fixed value for these devices. If the UL reads the device faster than the internal update rate, readings "repeat." For example, if using [cbTIn\(\)/TIn\(\)](#) in a loop to measure a rapidly changing temperature, readings do not change for several iterations of the loop, then "jump" when the update occurs internally

### Using single sensors with [cbTInScan\(\)](#)

When using single sensors for RTD or thermistor sensors, you should ignore the data for channels that do not have sensors attached. It is best to use [cbTIn\(\)/TIn\(\)](#) for these configurations, since you can select which channels to read. If you use [cbTInScan\(\)/TInScan\(\)](#), however, data for all channels over the entire range of channels are returned. Since some channels are not populated in this configuration, you should filter out the data for channels without sensors.

### NOSCALE

Specify the NOSCALE option to retrieve raw data in volts or resistance from the device. When NOSCALE is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.

### Saving configuration settings

InstaCal allows you to save configuration settings to a file or load a configuration from a previously saved file.

- Each USB-TEMP and USB-TEMP-AI channel can be configured to measure temperature data collected by one of five categories of temperature sensors: thermistors, thermocouples, RTDs, semiconductors, and *Disabled*.

- Each USB-TC and USB-TC-AI channel can be configured to measure temperature data collected by one of eight types of thermocouples.
- Each USB-TEMP-AI and USB-TC-AI voltage input channel can be configured for single-ended or differential mode and for one of four ranges -  $\pm 10$  V,  $\pm 5$  V,  $\pm 2.5$  V, or  $\pm 1.25$  V.

### **Recommended warm-up time**

Allow the device to warm-up for 30 minutes before taking measurements. This warm-up time minimizes thermal drift and achieves the specified rated accuracy of measurements.

For RTD or thermistor measurements, this warm-up time is also required to stabilize the internal current reference.

### **Calibration**

Any time the sensor category is changed in the configuration, a calibration is automatically performed by InstaCal. If the device has not been warmed up when this occurs, you should re-calibrate after the specified warm-up time.

### **Error codes**

- The UL returns -9999 when a value is out of range or an open connection is detected.
- The UL returns -9000 when the device is not ready. This usually occurs right after the device is powered up and calibration factors are being loaded.

## WEB-TEMP, WEB-TC

The WEB-TEMP and WEB-TC support the following UL and UL for .NET features.

### Temperature input

#### Functions

- UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)
- UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)

#### Scale

CELSIUS, FAHRENHEIT, KELVIN, NOSCALE\*

\*Refer to [NOSCALE](#) in the *Hardware considerations* section below for more information on this option.

#### HighChan

0 to 7

### Digital I/O

#### Configuration

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

#### Port I/O

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AUXPORT

DataValue

0 to 255 on AUXPORT

#### Bit I/O

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

### Configuration

#### Functions

UL: [cbGetConfig\(\)](#), [cbSetConfig\(\)](#), [cbGetConfigString\(\)](#), [cbSetConfigString\(\)](#)

#### ConfigItem

BINODEID, BINETIOTIMEOUT, BIHIDELOGINDLG

#### Device Number

0

#### maxConfigLen

up to 48

## Miscellaneous

### Functions

UL: [cbDeviceLogin\(\)](#), [cbDeviceLogout\(\)](#), [cbFlashLED\(\)](#)

UL for .NET: [DeviceLogin\(\)](#), [DeviceLogout\(\)](#), [FlashLED\(\)](#)

Call [cbFlashLED\(\)](#)/[FlashLED\(\)](#) to flash the POWER/COMM LED on a WEB device. This is useful if you have multiple devices connected and you want to identify a particular device.

## Hardware considerations

### Web based

If the user name and password have changed from the default, log in with the new user name and password to change configuration settings. Only one user can be logged in at a time. The log in session times out after five minutes of inactivity. Log in is not required to view the current configuration in InstaCal.

Hardware options are configurable on the web browser or with InstaCal. If hardware options are changed on the web browser while InstaCal is open, restart or refresh InstaCal to update its configuration pages with the settings stored on the device. Network parameters and some configuration settings for resistance measurement are configurable with InstaCal only.

Configuration options are stored in non-volatile memory in EEPROM, and are loaded on power up.

### Network parameters

The following network parameters are configurable with InstaCal. Configurable network options are enabled when you start InstaCal if the default user name and password are assigned. If a custom user name and password are assigned, the configurable network options are enabled after you log in.

- **Identifier:** Text that identifies the WEB device. This value is optional, and is not set by default. You can enter up to 48 alpha-numeric characters. To set this value in code, use the UL ConfigItem option BINODEID with [cbSetConfigString\(\)](#).
- **DHCP:** Enables automatic configuration of the IP address for a WEB device by a DHCP Server. When a DHCP-enabled server is available, an IP address is automatically assigned to the device when it is detected on the network. This value is set to *Enabled* by default. Disable this option when the server is not DHCP-enabled, or when you want to enter a static IP address.
- **IP:** The IP address that is currently stored on the device is displayed in the **Current Settings** frame on the InstaCal **Board Configuration** dialog. By default, this address is set automatically when a DHCP server is available. If you are setting a static IP address manually, enter it in the IP text box on the Default Settings frame. Every device connected to the network must have a unique IP address. This value is set to 192.168.0.101 by default.
- **Subnet:** The Subnet Mask that is currently stored on the device is displayed in the **Current Settings** frame on the InstaCal **Board Configuration** dialog. The Subnet Mask is the part of the IP address that denotes the local Subnet. By default, the Subnet Mask is set automatically when a DHCP server is available. If you are setting a static IP address manually, enter the Subnet Mask in the **Subnet** text box on the **Default Settings** frame.  
  
This value is set to 255.255.255.0 by default. The first three groups of numbers indicate the network number to which the device is connected, and the last group indicates the node number within the network that identifies the device.
- **Gateway:** The Gateway IP address that is currently stored on the device is displayed in the **Current Settings** frame on the InstaCal **Board Configuration** dialog. By default, the Gateway IP address is set automatically when a DHCP server is available. If you are setting a static IP address manually, enter the Gateway in the **Gateway** text box on the **Default Settings** frame. This value is set to 192.168.0.1 by default. The Gateway parameter is used for communication between devices on different networks.
- **Server:** Enables or disables the device web page server. This value is set to *Enabled* by default. When enabled, you can view the device web page with a web browser. When disabled, you can only access the device with InstaCal or the Universal Library. Disable when you want to restrict access to the device web page. Changes to this setting take affect the next time you power up the device.
- **Change Login** button: Opens a dialog to change the user name and password used to log in to a device session. Once changed, log in is required to change configurable options on the device. The user name and password are not stored on the host computer, and must be entered each time you start the application. Refer to *Logging in to a device session* below for more information.
- **Login** button: This button is enabled when login is required.

The InstaCal configuration page also lists the unique 64-bit physical (MAC) address assigned to the device. You cannot change this address.

## Logging in to a device session

You must be logged in to a device session in order to change the configuration settings of a device or change the state of the digital outputs. A user name and password are required to log in if they are not set to the default values. For security, it is recommended that you change the login values from the defaults. The log in session times out after five minutes of inactivity.

The default user name is set to *webtemp* for the WEB-TEMP, and *webtc* for the WEB-TC. The default password is *mccdaq* for both devices. You can change these values in InstaCal with the **Change Login** button after you are logged in to a device session. Each value can be up to eight alphanumeric characters.

Using InstaCal, when the user name and password have been changed from the default values, the configuration page opens with configurable items disabled and the **Login** button enabled. Click the **Login** button and then enter the values. The **INVALIDLOGIN** error is returned if the login information is not valid. The **SESSIONINUSE** error is returned if you attempt to log in when a session is currently open by another user. Only one user can be logged in to a session at a time.

Similarly, applications written with the Universal Library will perform a background log in when required if the login parameters are set to the default values. If custom values have been set, you have the option to allow the default login dialog to pop up when required or to disable the default dialog and handle login in your code.

To disable the default login dialog when using the Universal Library, you can select the "Show Login dialog prompt" option in InstaCal, or for a more permanent result, disable the default dialog using [cbSetConfig\(\)](#) with the *BIHIDELOGINDLG* ConfigItem argument within your application code.

## Factory default reset

To restore the network parameters (including the user name and password) to the factory default settings, press and hold the device **reset** button for three seconds. You do not have to be logged in to restore the default network settings.

## Manually adding devices to InstaCal

If a device is not yet connected to the local network, or if it is connected remotely to a different LAN, InstaCal will be unable to detect it. If autodetection fails, you can manually add the device to InstaCal using the **Web** tab on the **Board Selection List** dialog, and specify the IP address and port to use in the broadcast.

The default IP address and port add a placeholder to the configuration of a WEB device detected on the network. The default IP address broadcasts to all devices detected on the local subnet. The default port lists the default port number that is used to interface with the UL.

Any instance of the device type responding to the broadcast will attach to the placeholder. You can specify the device to attach to the placeholder by clicking the MAC check box and entering the device type and instance ID. Enter **C0** to locate a WEB-TC, or **C2** to locate a WEB-TEMP. Enter any value from **0x00000** to **0x2FFFE** (except 0x1FFFF) for the instance ID. The first three octets of a MAC address indicate the vendor ID and cannot be changed.

## Configuring the DIO channels to generate alarms

The WEB-TEMP and WEB-TC provide eight independent temperature alarms. Each alarm controls an associated digital I/O channel as an alarm output. The input to each alarm is one of the temperature input channels. You set up the temperature conditions to activate an alarm, and the output state of the digital channel (active high or low) when activated. You can view the alarm status on the web browser.

Digital channels that are configured as alarms will power up in an output state. When an alarm is activated, the associated DIO channel is driven to the output state defined by the alarm configuration. The alarm configurations are stored in non-volatile memory on the device and are loaded on power up. Alarm settings can be configured using the device web browser or InstaCal.

## Pacing temperature readings

The internal update rate for temperature measurement is a fixed value for these devices. If the UL reads the device faster than the internal update rate, temperature readings "repeat." For example, if using [cbTIn\(\)](#) in a loop to measure a rapidly changing temperature, readings do not change for several iterations of the loop, then "jump" when the update occurs internally.

## Using single sensors with cbTInScan() (WEB-TEMP only)

When using single sensors for RTD or thermistor sensors, you should ignore the data for channels that do not have sensors attached. It is best to use [cbTIn\(\)](#) for these configurations, since you can select which channels to read. If you use [cbTInScan\(\)](#), however, data for all channels over the entire range of channels are returned. Since some channels are not populated in this configuration, filter out the data for channels without sensors.

## NOSCALE

Specify the NOSCALE option to retrieve raw data in volts or resistance from the device. When NOSCALE is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.

## Channel names

You can specify a custom name for each of the device channels with InstaCal. Enter up to 10 alpha-numeric characters in the **Name** text box on each channel configuration page.

## Saving configuration settings

InstaCal allows you to save hardware configuration settings to a file, or load a configuration from a previously saved file.

Each WEB-TEMP channel can be configured to measure temperature data collected by one of five categories of temperature sensors: thermistors, thermocouples, RTDs, semiconductors, and *Disabled*. Each WEB-TC channel can be configured to measure temperature data collected by one of eight types of thermocouples.

## Recommended warm-up time

Allow the device to warm-up for 30 minutes before taking measurements. This warm-up time minimizes thermal drift and achieves the specified rated accuracy of measurements. For RTD or thermistor measurements, this warm-up time is also required to stabilize the internal current reference.

### **Calibration**

You can manually calibrate a WEB device using InstaCal or the web interface.

Any time the sensor category is changed in the configuration for the WEB-TEMP, a calibration is automatically performed. If the device is not warmed up when this occurs, re-calibrate after the specified warm-up time.

### **Timeout errors**

In some cases, there can be delays in obtaining the data from the WEB device, causing a [NOREMOTEACK](#) error to be generated. This can be caused by other users making configuration changes on the device, or by slow or busy network connections.

You can use the ConfigItem option *BINETIOTIMEOUT* with [cbSetConfig\(\)](#) to set the time (in mS) to wait for a device to acknowledge a command or query made via the network connection.



## WLS Series

The WLS Series includes the following hardware:

- WLS-IFC
- WLS-TEMP
- WLS-TC

These devices support the following UL and UL for .NET features.

## Temperature input (WLS-TEMP and WLS-TC)

### Functions

UL: [cbTIn\(\)](#), [cbTInScan\(\)](#)

UL for .NET: [TIn\(\)](#), [TInScan\(\)](#)

### Scale

CELSIUS, FAHRENHEIT, KELVIN, NOSCALE\*

\*Refer to [NOSCALE](#) in the *Hardware considerations* section below for more information on this option.

### HighChan

0 to 7

## Digital I/O (WLS-TEMP and WLS-TC)

### Configuration

UL: [cbDConfigBit\(\)](#), [cbDConfigPort\(\)](#)

UL for .NET: [DConfigBit\(\)](#), [DConfigPort\(\)](#)

PortNum

AUXPORT

PortType

AUXPORT

### Port I/O Functions

UL: [cbDIn\(\)](#), [cbDOut\(\)](#)

UL for .NET: [DIn\(\)](#), [DOut\(\)](#)

PortNum

AUXPORT

DataValue

0 to 255 on AUXPORT

### Bit I/O

UL: [cbDBitIn\(\)](#), [cbDBitOut\(\)](#)

UL for .NET: [DBitIn\(\)](#), [DBitOut\(\)](#)

PortType

AUXPORT

BitNum

0 to 7 on AUXPORT

## Configuration

### Functions

UL: [cbGetConfig\(\)](#), [cbSetConfig\(\)](#), [cbGetConfigString\(\)](#), [cbSetConfigString\(\)](#)

UL for .NET: [GetDeviceNotes\(\)](#), [SetDeviceNotes\(\)](#), [GetDeviceId\(\)](#), [SetDeviceId\(\)](#), [GetPANID\(\)](#), [SetPANID\(\)](#), [GetRFChannel\(\)](#), [SetRFChannel\(\)](#), [GetRSS\(\)](#)

### ConfigItem

BIRFCHANNEL, BIPANID, BINODEID, BIDEVNOTES

The following argument value is also valid for the WLS-TEMP and WLS-TC when they are operating as remote devices:

## Miscellaneous

### Functions

UL: [cbFlashLED\(\)](#)

UL for .NET: [FlashLED\(\)](#)

Causes the LED on a USB device to blink.

When you have several devices connected to the computer, use this function/method to identify a particular device by making its LED blink.

## Hardware considerations

You can operate the WLS-TEMP and WLS-TC as remote devices that communicate with the computer through a USB-to-wireless interface device, such as the WLS-IFC. The interface device can communicate with multiple remote WLS Series devices over a wireless link.

### Network parameters (wireless operation)

Use InstaCal to configure the network parameters required for wireless communication. Configuration options are stored in non-volatile memory in EEPROM, and are loaded on power up.

Network parameters can only be modified when the device is connected locally to the computer through the USB port. After configuring the network settings for a remote device, unplug from the computer and move the device to its remote location.

The following network parameters are programmable with InstaCal:

- **Identifier:** Text that identifies the WLS Series device. This value is optional, and is not set by default. You can enter up to 20 alpha-numeric characters.  
You can set the text identifier value using the ConfigItem option BINODEID with [cbSetConfigString\(\)](#) or [SetDeviceId\(\)](#) while the device is connected locally to the computer through the USB port, or when the device is operating remotely.
- **PAN:** The personal area network ID assigned to the device. This value is set to 1000 hex by default (4096 decimal). Most users do not need to change this value. However, you may want to change the PAN value in the following situations:
  - You have multiple WLS Series devices and do not want to allow communication between all of them. Set the PAN ID to the same value on each device that you want to communicate.
  - If other WLS Series devices are operating in the vicinity, you can avoid accidental changes to your device settings by changing the default PAN value.  
To change the PAN ID, enter a 16-bit hexadecimal value between 0 and FFFE. (Hexadecimal values consist of numbers between 0 and 9 and letters between A and F. In this case, up to four characters could be entered.)  
You can set the PAN value using the ConfigItem option BIPANID with [cbSetConfig\(\)](#) when the device is connected locally to the computer through the USB port.
- **RF channel:** The IEEE 802.15.4 RF (radio frequency) channel number used to transmit/receive data over the wireless link. Select a channel number between 12 and 23.

The table below lists each channel available along with its corresponding transmission frequency.

RF Channel	Transmission Frequency (GHz)	RF Channel	Transmission Frequency (GHz)
12	2.410	18	2.440
13	2.415	19	2.445
14	2.420	20	2.450
15	2.425	21	2.455
16	2.430	22	2.460
17	2.435	23	2.465

You can set the RF channel using the ConfigItem option BIRFCHANNEL with [cbSetConfig\(\)](#) while the device is connected locally to the computer through the USB port.

- **AES Key:** The value used to encrypt a message (optional). Click the AES Key button and enter up to 16 alpha-numeric characters to enable encryption. This value is write-only; it cannot be read back.

Only devices with matching parameter settings for PAN ID, RF channel, and AES encryption (if set) can communicate with each other.

- **Device Notes:** Use the Device Notes tab to enter up to 239 ASCII characters of additional text – for example, what the device is measuring, and which device it is communicating with. You should also enter the AES key on this tab, since that value cannot be read back by the device. You can set the text to store in the device memory using the ConfigItem option

The InstaCal configuration page also lists the unique 64-bit address assigned to the device. You cannot change this address.

### Received Signal Strength (wireless operations)

When a WLS Series device is operating remotely, The InstaCal configuration page includes a bar graph. The bar graph indicates the strength of the signal received by the remote device from the wireless interface module, and the fade margin of signals received by a device (refer to the following table.)

Active bars	Fade margin	RSS (dBm)
0 - Weak signal	< 10 dBm	-82 dBm > rss
1 - Moderate signal	≥ 10 dBm	-72 dBm > rss ≥ -82 dBm
2 - Strong signal	≥ 20 dBm	-62 dBm > rss ≥ -72 dBm
3 - Very strong signal	≥ 30 dBm	rss > -62 dBm

The number of bars corresponds to the number of LEDs that are lit on the remote device. The bar graph display updates every two seconds on the InstaCal form.

If the signal is not strong enough for communication between the interface device and the remote device, no bars or LEDs show, and a [NOREMOTEACK](#) error is returned. If this occurs, try moving or re-orienting the device to increase the strength of the signal.

You can retrieve the value in dBm of the signal strength received by a remote device using the ConfigItem option BIRSS with [cbGetConfig\(\)](#).

### External power required for wireless operations

An external power supply is required to power remote devices. For wireless operations, connect the device USB cable to the AC-to-USB power adapter that shipped with the device.

### Always connect an external hub to its power supply

If you are using a hybrid hub – one that can operate in either self-powered or bus-powered mode – always connect it to its external power supply.

If you use a hub of this type without connecting to external power, communication errors may occur that could result in corrupt configuration information on the wireless device. You can restore the factory default configuration settings with InstaCal.

### Factory default reset

To restore factory default configuration settings, click on the **Reset Defaults** button on the InstaCal configuration page. The device must be locally connected to the computer USB port to restore default settings.

### Configuring the DIO channels to generate alarms (WLS-TEMP and WLS-TC)

The WLS-TEMP and WLS-TC both provide eight independent temperature alarms. Each alarm controls an associated digital I/O channel as an alarm output. The input to each alarm is one of the temperature input channels. Use InstaCal to set up the temperature conditions to activate an alarm, and the output state of the channel (active high or low) when activated.

Digital channels that are configured as alarms will power up in an output state. When an alarm is activated, the associated DIO channel is driven to the output state defined by the alarm configuration. The alarms function both in wireless mode and while attached to the USB port on a computer. The alarm configurations are stored in non-volatile memory on the device and are loaded on power up.

Alarm settings can be configured when the device is connected locally to the computer through the USB port, or when the device is operated remotely through a wireless interface.

### Pacing temperature readings

The internal update rate for temperature measurement is a fixed-value for these devices. If the UL reads the device faster than the internal update rate, temperature readings "repeat." For example, if using [cbTIn\(\)/TIn\(\)](#) in a loop to measure a rapidly changing temperature, readings do not change for several iterations of the loop, then "jump" when the update occurs internally.

### Using single sensors with cbTInScan()

When using single sensors for RTD or thermistor sensors, you should ignore the data for channels that do not have sensors attached. It is best to use [cbTIn\(\)/TIn\(\)](#) for these configurations, since you can select which channels to read. If you use [cbTInScan\(\)/TInScan\(\)](#), however, data for all channels over the entire range of channels are returned. Since some channels are not populated in this configuration, you should filter out the data for channels without sensors.

### NOSCALE

Specify the NOSCALE option to retrieve raw data in volts or resistance from the device. When NOSCALE is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.

### Saving configuration settings (WLS-TEMP and WLS-TC)

InstaCal allows you to save WLS-TEMP and WLS-TC configuration settings to a file or load a configuration from a previously saved file.

Each WLS-TEMP channel can be configured to measure temperature data collected by one of five categories of temperature

sensors: thermistors, thermocouples, RTDs, semiconductors, and *Disabled*.

Each WLS-TC channel can be configured to measure temperature data collected by one of eight types of thermocouples.

### **Recommended warm-up time**

Allow the WLS-TEMP and WLS-TC to warm-up for 30 minutes before taking measurements. This warm-up time minimizes thermal drift and achieves the specified rated accuracy of measurements.

For RTD or thermistor measurements, this warm-up time is also required to stabilize the internal current reference.

### **Calibration**

Any time the sensor category is changed in the configuration for the WLS-TEMP, a calibration is automatically performed by InstaCal. If the device has not been warmed up when this occurs, you should re-calibrate after the specified warm-up time.

### **Error codes**

The UL returns -9999 when a value is out of range or an open connection is detected.

The UL returns -9000 when the device is not ready. This usually occurs right after the device is powered up and calibration factors are being loaded.

With wireless operations, the UL returns NOREMOTEACK when the signal is not strong enough for communication between the interface device and the remote device.

## Measurement Computing Device IDs

The following table lists the device ID associated with each Measurement Computing hardware type. This information is returned by the BoardName and BoardNum arguments. Devices are listed alphabetically in the table.

Board name	Device ID (in decimal)
CIO-COM422	20481
CIO-COM485	20482
CIO-CTR05	2049
CIO-CTR10	2050
CIO-CTR10-HD	2051
CIO-CTR20-HD	2052
CIO-DAC02	1537
CIO-DAC02/16	1796
CIO-DAC04/12-HS	2564
CIO-DAC04/16-HS	19
CIO-DAC08	1538
CIO-DAC08/16	1797
CIO-DAC08I	1541
CIO-DAC16	1539
CIO-DAC16/16	1798
CIO-DAC16I	1540
CIO-DAS08	3073
CIO-DAS08/AOH	3077
CIO-DAS08/AOL	3076
CIO-DAS08/AOM	3079
CIO-DAS08/Jr	3080
CIO-DAS08Jr/16	3082
CIO-DAS08PGH	3075
CIO-DAS08PGL	3074
CIO-DAS08/PGM	3078
CIO-DAS1401/12	3588
CIO-DAS1402/12	3589
CIO-DAS1402/16	3590
CIO-DAS16	257
CIO-DAS16/330	260
CIO-DAS16/330i	261
CIO-DAS16/F	258
CIO-DAS16/Jr	259
CIO-DAS16/Jr16	265
CIO-DAS16/M1	262
CIO-DAS16/M1/16	9
CIO-DAS1601/12	3585
CIO-DAS1602/12	3586
CIO-DAS1602/16	3587
CIO-DAS48PGA	3329
CIO-DAS6402/12	8
CIO-DAS6402/16	10
CIO-DAS800	24577
CIO-DAS801	24578
CIO-DAS802	24579
CIO-DAS802/16	24580
CIO-DAS-TC	46
CIO-DAS-TEMP	4353
CIO-DDA06/12	1793
CIO-DDA06/16	1794

CIO-DDA06/Jr	1795
CIO-DDA06Jr/16	1799
CIO-DI192	1037
CIO-DI48	1033
CIO-DI96	1035
CIO-DIO192	1029
CIO-DIO24	1025
CIO-DIO24/CTR3	1030
CIO-DIO24H	1026
CIO-DIO48	1027
CIO-DIO48H	1031
CIO-DIO96	1028
CIO-DISO48	8193
CIO-DO192H	1038
CIO-DO24DD	1039
CIO-DO48DD	1040
CIO-DO48H	1034
CIO-DO96H	1036
CIO-DUAL422	20483
CIO-DUAL-AC5	1032
CIO-EXP16	769
CIO-EXP32	770
CIO-EXP-BRIDGE	773
CIO-EXP-GP	771
CIO-EXP-RTD	772
CIO-INT32	12289
CIO-PDISO16	2306
CIO-PDISO8	2305
CIO-PDMA16	1281
CIO-PDMA32	18
CIO-QUAD02	47
CIO-QUAD04	48
CIO-RELAY08	4098
CIO-RELAY16	4097
CIO-RELAY16/M	4099
CIO-RELAY16M	17
CIO-RELAY24	42
CIO-RELAY32	43
CIO-SSH16	513
CPCI-DIO24H	85
CPCI-DIO48H	91
CPCI-DIO96H	90
CPCI-GPIB	14
DEMO-BOARD	45
E-PDISO16	137
ISA-MDB64	70
MAI-16	80
MEGA-FIFO	3841
MEM-8	73
MEM-32	74
MII-32	71
miniLAB 1008	117
MIO-32	72
MSSR-24	78
PC104-AC5	88
PC104-CTR10-HD	2053

PC104-DAC06	1543
PC104-DAS08	3081
PC104-DAS16Jr/12	263
PC104-DAS16Jr/16	264
PC104-DI48	1042
PC104-DIO48	1041
PC104-DO48H	1043
PC104-MDB64	79
PC104-PDISO8	2307
PC-CARD-D24/CTR3	61
PC-CARD-DAC08	92
PC-CARD-DAS16/12	58
PC-CARD-DAS16/12-AO	59
PC-CARD-DAS16/16	56
PC-CARD-DAS16/16-AO	57
PC-CARD-DAS16/330	60
PC-CARD-DIO48	62
PCI-2511	165
PCI-2513	166
PCI-2515	167
PCI-2517	168
PCI-COM232	63
PCI-COM232/2	64
PCI-COM232/4	65
PCI-COM422	66
PCI-COM422/2	67
PCI-COM485	68
PCI-COM485/2	69
PCI-CTR05	24
PCI-CTR10	110
PCI-CTR20HD	116
PCI-DAC04/12HS	38
PCI-DAC04/16HS	39
PCI-DAC6702	112
PCI-DAC6703	113
PCI-DAS08	41
PCI-DAS1000	76
PCI-DAS1001	26
PCI-DAS1002	27
PCI-DAS1200	15
PCI-DAS1200Jr	25
PCI-DAS16/M1	31
PCI-DAS1602/12	16
PCI-DAS1602/16	1
PCI-DAS1602JR/16	28
PCI-DAS3202/16	87
PCI-DAS4020/12	82
PCI-DAS6013	120
PCI-DAS6014	121
PCI-DAS6023	93
PCI-DAS6025	94
PCI-DAS6030	95
PCI-DAS6031	96
PCI-DAS6032	97
PCI-DAS6033	98

PCI-DAS6034	99
PCI-DAS6035	100
PCI-DAS6036	111
PCI-DAS6040	101
PCI-DAS6052	102
PCI-DAS6070	103
PCI-DAS6071	104
PCI-DAS64	50
PCI-DAS64/M1/16	53
PCI-DAS64/M2/16	54
PCI-DAS64/M3/16	55
PCI-DAS6402/12	30
PCI-DAS6402/16	29
PCI-DAS-TC	52
PCI-DDA02/12	32
PCI-DDA02/16	35
PCI-DDA04/12	33
PCI-DDA04/16	36
PCI-DDA08/12	34
PCI-DDA08/16	37
PCI-DIO24	40
PCI-DIO24/LP	119
PCI-DIO24/S	126
PCI-DIO24H	20
PCI-DIO24H/CTR3	21
PCI-DIO48H	11
PCI-DIO48H/CTR15	22
PCI-DIO96	84
PCI-DIO96H	23
PCI-DUAL-AC5	51
PCI-INT32	44
PCI-MDB64	75
PCI-PDISO16	13
PCI-PDISO8	12
PCI-QUAD04	77
PCI-QUAD-AC5	89
PCIE-DAS1602/16	277
PCIE-DIO24	219
PCIE-DIO96H	218
PCIM-DAS1602/16	86
PCIM-DAS16JR/16	123
PCIM-DDA06/16	83
PCM-COM422	16388
PCM-COM485	16389
PCM-D24/CTR3	16386
PCM-DAC02	16387
PCM-DAC08	16401
PCM-DAS08	16385
PCM-DAS16D/12	16390
PCM-DAS16D/12AO	16395
PCM-DAS16D/16	16392
PCM-DAS16S/12	16391
PCM-DAS16S/16	16393
PCM-DAS16S/330	16394
PCM-QUAD02	49
USB-1024LS, PMD-1024LS	118



PPIO-AI08	2818
PPIO-CTR06	2819
PPIO-DIO24H	2817
Switch & Sense 8/8	132
USB-201	275
USB-204	276
USB-1024HLS, PMD-1024HLS	127
USB-1096HFS	131
USB-1208FS, PMD-1208FS	130
USB-1208FS-Plus	232
USB-1208LS, PMD-1208LS	122
USB-1208HS	196
USB-1208HS-2AO	197
USB-1208HS-4AO	198
USB-1408FS	161
USB-1408FS-Plus	233
USB-1602HS	213
USB-1602HS-2AO	214
USB-1604HS	215
USB-1604HS-2AO	216
USB-1608FS, PMD-1608FS	125
USB-1608FS-Plus	234
USB-1608G	272
USB-1608GX	273
USB-1608GX-2AO	274
USB-1608HS	189
USB-1608HS-2AO	153
USB-1616FS	129
USB-1616HS	203
USB-1616HS-2	204
USB-1616HS-4	205
USB-1616HS-BNC	217
USB-2404-10	222
USB-2404-60	223
USB-2404-UI	225
USB-2408	253
USB-2408-2AO	254
USB-2416	208
USB-2416-4AO	209
USB-2523	177
USB-2527	178
USB-2533	179
USB-2537	180
USB-3101	154
USB-3101FS	224
USB-3102	155
USB-3103	156
USB-3104	157
USB-3105	158
USB-3106	159
USB-3110	162
USB-3112	163

USB-3114	164
USB-4301	174
USB-4302	184
USB-4303	185
USB-4304	186
USB-5201 (<Rev. 3 fw)	152
USB-5201 (Rev. 3 fw and later)	175
USB-5203 (<Rev. 3 fw)	151
USB-5203 (Rev. 3 fw and later)	176
USB-7202	242
USB-7204	240
USB-DIO24/37	147
USB-DIO24H/37	148
USB-DIO96H	146
USB-DIO96H/50	149
USB-ERB08	139
USB-ERB24	138
USB-PDISO8	140
USB-PDISO8/50	150
USB-QUAD08	202
USB-SSR08	134
USB-SSR24	133
USB-TC	144
USB-TC-AI	187
USB-TEMP	141
USB-TEMP-AI	188
WEB-TC	192
WEB-TEMP	194
WLS-IFC	181
WLS-TC	182
WLS-TEMP	183

## Introduction

The topics listed below provide a brief explanation of each UL function and UL for .NET method, and provides you with a general idea of what you can do with the Universal Library and the Universal Library for .NET.

Universal Library functions and Universal Library for .NET methods:

- [Analog I/O Functions](#) and [Analog I/O Methods](#)
- [Configuration Functions](#) and [Configuration Methods](#)
- [Counter Functions](#) and [Counter Methods](#)
- [DataLogger Functions](#) and [DataLogger Methods](#)
- [Digital I/O Functions](#) and [Digital I/O Methods](#)
- [Error Handling Functions](#) and [Error Handling Methods and Properties](#)
- [Memory Board Functions](#) and [Memory Board Methods](#)
- [Revision Control Functions](#) and [Revision Control Methods](#)
- [Streamer File Functions](#) and [Streamer File Methods](#)
- [Synchronous I/O Functions](#) and [Synchronous I/O Methods](#)
- [Temperature Input Functions](#) and [Temperature Input Methods](#)
- [Windows Memory Management Functions](#) and [Windows Memory Management Methods](#)
- [Miscellaneous Functions](#) and [Miscellaneous Methods](#)

## Important

We highly recommend that you review and run one of the many example programs included with the Universal Library installation. These programs often provide the ideal description of the various functions, as well as providing a starting point from which you can use to write your own programs.

## Multi-threading

Only one application program that calls the Measurement Computing driver can be running at a time.

For example, when you are running a program created with the Universal Library, you cannot change any hardware configuration settings with the InstaCal program until you first stop the UL program. This is because InstaCal stores hardware configuration settings in a file (cb.cfg) which is read by the Universal Library when you run an application. To change device settings, stop the UL application and run InstaCal.

## Analog I/O Functions

The analog I/O functions perform analog input and output and convert analog data.

Most analog I/O functions include options that may not be compatible with your hardware. To determine which of these functions are compatible, refer to the board-specific information contained in the *Universal Library User's Guide*.

- [cbAIn\(\)](#) - Takes a single reading from an analog input channel (A/D), and returns a 16-bit integer value.
- [cbAIn32\(\)](#) - Takes a single reading from an analog input channel (A/D), and returns a 32-bit unsigned integer value.
- [cbAInScan\(\)](#) - Repeatedly scans a range of analog input (A/D) channels. You can specify the channel range, the number of samples, the sampling rate, and the A/D range. The data that is collected is stored in memory for later transfer to an array.
- [cbALoadQueue\(\)](#) - Loads a series of chan/gain pairs into A/D board's queue. These chan/gains are used with all subsequent analog input functions.
- [cbAOut\(\)](#) - Outputs a single value to an analog output (D/A).
- [cbAOutScan\(\)](#) - Repeatedly scans a range of analog output (D/A) channels. You can specify the channel range, the number of samples, the update rate, and the D/A range. The data values from consecutive elements of a memory buffer are sent to each D/A channel in the scan.
- [cbAPretrig\(\)](#) - 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 before the trigger occurred, and fills the remainder of the buffer with post-trigger samples. You can specify the channel range, the sampling rate, the A/D range, and the number of pre-trigger samples. All of the data that is collected is stored in memory for later transfer to an array.
- [cbATrig\(\)](#) - Reads the analog input and waits until it goes above or below a specified threshold. When the trigger condition is met, the current sample is returned.
- [cbAConvertData\(\)](#) - Converts analog data from data plus channel tags to separate data and channel tags when using 12-bit devices that support channel tags.
- [cbACalibrateData\(\)](#) - Calibrates analog data after an acquisition using the NOCALIBRATEDATA option is complete.
- [cbAConvertPretrigData\(\)](#) - Used to properly arrange data and separate channel tags (if necessary) after a pre-trigger acquisition is complete, when data was acquired without using the CONVERTDATA option.
- [cbVIn\(\)](#) - Reads an A/D input channel, and returns a single precision float voltage value.
- [cbVIn32\(\)](#) - Reads an A/D input channel, and returns a double precision float voltage value.
- [cbVOut\(\)](#) - Sets the value of a D/A output.

## Configuration Functions

The configuration information for all boards is stored in the configuration file CB.CFG. This information is loaded from CB.CFG by all programs that use the library. The Library includes the following functions to retrieve or change configuration options:

- [cbGetConfig\(\)](#) - Returns the current value for a specified configuration option.
- [cbGetConfigString\(\)](#) - Retrieves configuration or device information as a null-terminated string.
- [cbSetConfig\(\)](#) - Sets the current value for a specified configuration option.
- [cbSetConfigString\(\)](#) - Sets the configuration or device information as a null-terminated string.
- [cbGetSignal\(\)](#) - Retrieves the configured auxiliary or DAQ Sync connection and polarity for the specified timing and control signal. This function is intended for advanced users.
- [cbSelectSignal\(\)](#) - Configures timing and control signals to use specific auxiliary or DAQ Sync connections as a source or destination. This function is intended for advanced users.
- [cbSetTrigger\(\)](#) - Sets up trigger parameters used with the EXTTRIGGER option for [cbAInScan\(\)](#).

## Counter Functions

Counter functions load, read, and configure counters. There are several types of counters used in Measurement Computing devices. Some of the counter commands only apply to one type of counter.

- [cbC7266Config\(\)](#) - Sets the basic operating mode of an LS7266 counter.
- [cbC8254Config\(\)](#) - Sets the basic operating mode of the 8254 counter.
- [cbC8536Config\(\)](#) - Sets the operating mode of the 8536 counter.
- [cbC8536Init\(\)](#) - Initializes and selects all of the chip-level features for a 8536 counter board. The options set by this command are associated with each counter chip, not the individual counters within it.
- [cbC9513Config\(\)](#) - Sets the operating mode of the 9513 counter. This function sets all of the programmable options that are associated with a 9513 counter. It is similar in purpose to [cbC8254Config\(\)](#) except that it is used with a 9513 counter.
- [cbC9513Init\(\)](#) - Initializes and selects all of the chip level features for a 9513 counter board. The options set by this command are associated with each counter chip, not the individual counters within it.
- [cbCClear\(\)](#) - Clears a scan counter value (sets it to zero).
- [cbCConfigScan\(\)](#) - Configures a scan counter channel. This function only works with counter boards that have counter scan capability.
- [cbCFreqIn\(\)](#) - Measures the frequency of a signal by counting it for a specified period of time (GateInterval), and then converting the count to count/sec (Hz). This function only works with 9513 counters.
- [cbCIn\(\)](#) - Reads a counter's current value as a 16-bit integer.
- [cbCIn32\(\)](#) - Reads a counter's current value as a 32-bit integer.
- [cbCIn64\(\)](#) - Reads a counter's current value as a 64-bit integer.
- [cbCInScan\(\)](#) - Scans a range of scan counter channels, and stores the samples in memory for later transfer to an array.
- [cbCLoad\(\)](#) - Loads a counter with a 16-bit integer initial value.
- [cbCLoad32\(\)](#) - Loads a counter with a 32-bit integer initial value.
- [cbCLoad64\(\)](#) - Loads a counter with a 64-bit integer initial value.
- [cbCStatus\(\)](#) - Read the counter status of a counter. Returns various bits that indicate the current state of a counter.
- [cbCStoreOnInt\(\)](#) - Installs an interrupt handler that stores the current count whenever an interrupt occurs. This function only works with 9513 counters.
- [cbCPulseOutStart\(\)](#) - Starts a timer to generate digital pulses at a specified frequency and duty cycle.
- [cbCPulseOutStop\(\)](#) - Stops a timer output.
- [cbCTimerOutStart\(\)](#) - Starts a timer square wave output.
- [cbCTimerOutStop\(\)](#) - Stops a timer square wave output.

## Data Logger Functions

The data logger functions read and convert binary files logged by MCC hardware equipped with a data logger capability.

- [cbLogConvertFile\(\)](#) - Converts a binary log file to a comma-separated values (.CSV) text file or another text file format that you specify.
- [cbLogGetAIChannelCount\(\)](#) - Retrieves the total number of analog input channels logged in a binary file.
- [cbLogGetAIInfo\(\)](#) - Retrieves the channel number and unit value of each analog input channel logged in a binary file.
- [cbLogGetCJCInfo\(\)](#) - Retrieves the number of CJC temperature channels logged in a binary file.
- [cbLogGetDIOInfo\(\)](#) - Retrieves the number of digital I/O channels logged in a binary file.
- [cbLogGetFileInfo\(\)](#) - Retrieves the version level and byte size of a binary file.
- [cbLogGetFileName\(\)](#) - Retrieves the name of the n<sup>th</sup> file in the directory containing binary log files.
- [cbLogGetPreferences\(\)](#) - Retrieves API preference settings for time stamped data, analog data, and CJC temperature data. Returns the default values unless changed using [cbLogSetPreferences\(\)](#).
- [cbLogGetSampleInfo\(\)](#) - Retrieves the sample interval, sample count, and the date and time of the first data point contained in a binary file.
- [cbLogReadAIChannels\(\)](#) - Retrieves analog input data from a binary file, and stores the values in an array.
- [cbLogReadCJCChannels\(\)](#) - Retrieves CJC temperature data from a binary file, and stores the values in an array.
- [cbLogReadDIOChannels\(\)](#) - Retrieves digital I/O channel data from a binary file, and stores the values in an array.
- [cbLogReadTimeTags\(\)](#) - Retrieves date and time values logged in a binary file. This function stores date values in the DateTags array, and time values in the TimeTags array.
- [cbLogSetPreferences\(\)](#) - Sets preferences for returned time stamped data, analog temperature data, and CJC temperature data.

## Digital I/O Functions

The digital function perform digital input and output operations on various types of digital I/O ports.

- [cbDBitIn\(\)](#) - Reads a single bit from a digital input port.
- [cbDBitOut\(\)](#) - Sets a single bit on a digital output port.
- [cbDConfigBit\(\)](#) - Configures a specific digital bit as input or output.
- [cbDConfigPort\(\)](#) - Selects whether a digital port is an input or an output.
- [cbDIn\(\)](#) - Reads a specified digital input port.
- [cbDInScan\(\)](#) - Reads a specified number of bytes or words from a digital input port at a specified rate.
- [cbDOut\(\)](#) - Writes a byte to a digital output port.
- [cbDOutScan\(\)](#) - Writes a series of bytes or words to a digital output port at a specified rate.



## Error Handling Functions

The Universal Library includes two functions for handling errors.

- [cbErrHandling\(\)](#) - Sets the method of reporting and handling errors for all function calls.
- [cbGetErrMsg\(\)](#) - Returns the error message associated with a specific error code.

## Memory Board Functions

The memory board functions read and write data to and from a memory board, and also set modes that 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 to a memory board. To do this, 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 it.

- [cbMemSetDTMode\(\)](#) - Sets DT-Connect mode on a memory board. 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 to one of these settings.
- [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\(\)](#) - Reads a specified number of points from a memory board starting at a specified address.
- [cbMemWrite\(\)](#) - 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.

## Revision Control Functions

As new revisions of the library are released, bugs from previous revisions are fixed and occasionally new functions are added. It is our goal to preserve the existing programs you have written and therefore to never change the order or number of arguments in a function. However, sometimes this is not possible.

The revision control function initializes the DLL so that the functions are interpreted according to the format of the revision you wrote and used to compile the program.

- [cbDeclareRevision\(\)](#) – Declares the revision number of the Universal Library that your program was written with.
- [cbGetRevision\(\)](#) – Returns the version number of the installed Universal Library.

## Streamer File Functions

The streamer file functions explained below create, fill, and read streamer files.

- [cbFileAInScan\(\)](#) - Transfer analog input data directly to file. Very similar to [cbAInScan\(\)](#) except that the data is stored in a file instead of memory.
- [cbFilePretrig\(\)](#) - Pre-triggered analog input to a file. Very similar to [cbAPretrig\(\)](#) except that the data is stored in a file instead of memory.
- [cbFileGetInfo\(\)](#) - Reads streamer file information on how much data is in the file, and the conditions under which it was collected (sampling rate, channels, etc.).
- [cbFileRead\(\)](#) - Reads a selected number of data points from a streamer file into an array.

## Synchronous Functions

The synchronous functions synchronously read data from analog, counter, thermocouple, and digital input channels, write data to analog or digital output channels, or configure devices that support synchronous I/O.

- [cbDaqInScan\(\)](#) – Scans analog, digital, temperature, and counter inputs synchronously, and stores the values in memory.
- [cbDaqOutScan\(\)](#) – Outputs values synchronously to analog output channels and digital output ports.
- [cbDaqSetSetpoints\(\)](#) – Configures up to 16 detection setpoints associated with the input channels within a scan group.
- [cbDaqSetTrigger\(\)](#) – Selects a trigger source and sets up its parameters. This method starts or stops a synchronous data acquisition operation using `cbDaqInScan()` with the `EXTTRIGGER` option

## Temperature Input Functions

The temperature sensor functions convert a raw analog input from an EXP or other temperature sensor board to temperature.

- [`cbTIn\(\)`](#) - Reads a channel from a digital input board, filters it (if specified), does the cold junction compensation, linearizes and converts it to temperature.
- [`cbTInScan\(\)`](#) - Scans a range of temperature inputs. Reads input temperatures from a range of channels, and returns the temperature values in an array

## Windows Memory Management Functions

The Windows memory management functions take care of allocating, freeing and copying to/from Windows global memory buffers.

- [cbWinBufAlloc\(\)](#) - Allocates a Windows global memory buffer.
- [cbWinBufAlloc32\(\)](#) - Allocates a Windows global memory buffer for use with 32-bit scan functions, and returns a memory handle for the buffer.
- [cbWinBufAlloc64\(\)](#) - Allocates a Windows memory buffer large enough for double precision data values, and returns a memory handle for the buffer.
- [cbWinBufFree\(\)](#) - Frees a Windows buffer.
- [cbWinArrayToBuf\(\)](#) - Copies data from an array to a Windows buffer.
- [cbWinBufToArray\(\)](#) - Copies data from a Windows buffer to an array.
- [cbWinBufToArray32\(\)](#) - Copies 32-bit data from a Windows global memory buffer into an array. This function is typically used to retrieve data from the buffer after executing an input scan function.
- [cbScaledWinArrayToBuf\(\)](#) - Copies double precision values from an array into a Windows memory buffer.
- [cbScaledWinBufAlloc\(\)](#) - Allocates a Windows global memory buffer large enough to hold scaled data obtained from scan operations in which the SCALEDATA option is selected, and returns a memory handle for the buffer.
- [cbScaledWinBufToArray\(\)](#) - Copies double precision values from a Windows memory buffer into an array.

## Miscellaneous Functions

These functions do not as a group fit into a single category. They get and set board information, convert units, manage events and background operations, and perform serial communication operations.

- [cbDeviceLogin\(\)](#) - Opens a device session with a shared device.
- [cbDeviceLogout\(\)](#) - Releases the device session with a shared device.
- [cbEnableEvent\(\)](#) - Installs an event handler, or *callback function*, to be called when a specified condition occurs.
- User Callback Function – Defines the prototype for the user function for [cbEnableEvent\(\)](#). This defines the format for the user-defined handlers to be called when the events set up using [cbEnableEvent\(\)](#) occurs.
- [cbDisableEvent\(\)](#) - Disables the notification of an event handler for a specified condition.
- [cbFlashLED\(\)](#) - Causes the LED on a USB device to flash.
- [cbFromEngUnits\(\)](#) - Converts a single precision voltage (or current) value in engineering units to an integer D/A count value for output to a D/A.
- [cbGetBoardName\(\)](#) - Returns the name of a specified board.
- [cbGetStatus\(\)](#) - Returns the status of a background operation. Once a background operation starts, your program needs to periodically check on its progress. This function returns the current status of the process.
- [cbGetTCValues\(\)](#) - Converts raw thermocouple data gathered with [cbDagInScan\(\)](#) to data on a temperature scale (Celsius, Fahrenheit, or Kelvin).
- [cbInByte\(\)](#) - Reads a byte from a hardware register on a board.
- [cbInWord\(\)](#) - Reads a word from a hardware register on a board.
- [cbOutByte\(\)](#) - Writes a byte to a hardware register on a board.
- [cbOutWord\(\)](#) - Writes a word to a hardware register on a board.
- [cbRS485\(\)](#) - Sets the transmit and receive buffers on an RS485 port.
- [cbStopBackground\(\)](#) - Stop a background process. It is sometimes necessary to stop a background process even though the process has been set up to run continuously. This function stops a background process that is running. [cbStopBackground\(\)](#) should be executed after normal termination of all background functions in order to clear variables and flags.
- [cbTEDSRead\(\)](#) - Reads data from a TEDS sensor into an array.
- [cbToEngUnits\(\)](#) - Converts an integer A/D count value to an equivalent single precision voltage (or current) value.
- [cbToEngUnits32\(\)](#) - Converts an integer A/D count value to an equivalent double precision voltage (or current) value.



## Analog I/O Methods

The analog I/O methods available from the [MccBoard class](#) are explained below. These methods perform analog input and output and convert analog data.

- [MccBoard.AIn\(\)](#) - Takes a single reading from an analog input channel (A/D), and returns a 16-bit integer value.
- [MccBoard.AIn32\(\)](#) - Takes a single reading from an analog input channel (A/D), and returns a 32-bit integer value.
- [MccBoard.AInScan\(\)](#) - Repeatedly scans a range of analog input (A/D) channels. You can specify the channel range, the number of samples, the sampling rate, and the A/D range. The data that is collected is stored in memory for later transfer to an array.
- [MccBoard.ALoadQueue\(\)](#) - Loads a series of channel/gain pairs into an A/D board's queue. These channel/gains are used with all subsequent analog input methods.
- [MccBoard.AOut\(\)](#) - Outputs a single value to an analog output (D/A).
- [MccBoard.AOutScan\(\)](#) - Repeatedly scans a range of analog output (D/A) channels. You can specify the channel range, the number of samples, the update rate, and the D/A range. The data values from consecutive elements of a memory buffer are sent to each D/A channel in the scan.
- [MccBoard.APretrig\(\)](#) - 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 before the trigger occurred, and fills the remainder of the buffer with post-trigger samples. You can specify the channel range, the sampling rate, the A/D range, and the number of pre-trigger samples. All of the data that is collected is stored in memory for later transfer to an array.
- [MccBoard.ATrig\(\)](#) - Reads analog input and waits until it goes above or below a specified threshold. When the trigger condition is met, the current sample is returned.
- [MccBoard.AConvertData\(\)](#) - Converts analog data from data plus channel tags to separate data and channel tags when using 12-bit devices that support channel tags.
- [MccBoard.ACalibrateData\(\)](#) - Calibrates analog data after an acquisition using the NoCalibrateData option is complete.
- [MccBoard.AConvertPretrigData\(\)](#) - Used to properly arrange data and separate channel tags (if necessary) after a pre-trigger acquisition is complete, when data was acquired without using the ConvertData option.
- [MccBoard.VIn\(\)](#) - Reads an A/D input channel, and returns a single precision voltage value.
- [MccBoard.VIn32\(\)](#) - Reads an A/D input channel, and returns a double precision voltage value.
- [MccBoard.VOut\(\)](#) - Sets the value of a D/A output.

## Configuration Methods and Properties

The configuration methods and properties available from the MccBoard class, cBoardConfig class, cCtrConfig class, cDioConfig class, and cExpansionConfig class are explained below.

The configuration information for all boards is stored in the configuration file CB.CFG . This information is loaded from CB.CFG by all programs that use the library. The Library includes the following classes and methods that retrieve or change configuration options.

- [MccBoard.BoardNum](#) property – Retrieves the number of the board associated with an instance of the MccBoard class.
- [MccBoard.GetSignal\(\)](#) – Retrieves the configured auxiliary or DAQ Sync connection and polarity for the specified timing and control signal. This method is intended for advanced users.
- [MccBoard.SelectSignal\(\)](#) – Configures timing and control signals to use specific auxiliary or DAQ Sync connections as a source or destination. This method is intended for advanced users.
- [MccBoard.SetTrigger\(\)](#) – Sets up trigger parameters used with the ExtTrigger option for [MccBoard.AInScan\(\)](#).
- [MccBoard.BoardConfig](#) property – Gets an instance of a [cBoardConfig](#) object.
- [MccBoard.BoardConfig.DACUpdate\(\)](#) – Updates the voltage values on analog output channels.
- [MccBoard.BoardConfig.GetAdRetrigCount\(\)](#) – Gets the number of samples to acquire during a trigger event when ScanOptions.RetrigMode is set.
- [MccBoard.BoardConfig.GetBaseAdr\(\)](#) – Gets the base address of a board.
- [MccBoard.BoardConfig.GetBoardType\(\)](#) – Gets the unique number (device ID) assigned to the board (between 0 and 8000h) indicating the type of board installed.
- [MccBoard.BoardConfig.GetCiNumDevs\(\)](#) – Gets the number of counter devices on the board.
- [MccBoard.BoardConfig.GetClock\(\)](#) – Gets the clock frequency in MHz (40, 10, 8, 6, 5, 4, 3, 2, 1), or 0 for not supported.
- [MccBoard.BoardConfig.GetDACStartup\(\)](#) – Gets the board's configuration register STARTUP bit setting.
- [MccBoard.BoardConfig.GetDACUpdateMode\(\)](#) – Gets the update mode for a digital-to-analog converter (DAC).
- [MccBoard.BoardConfig.GetDeviceId\(\)](#) – Gets the name that identifies the instance of a device.
- [MccBoard.BoardConfig.GetDeviceNotes\(\)](#) – Gets the device notes that are stored in the device's memory.
- [MccBoard.BoardConfig.GetDInMask\(\)](#) – Determines the bits on a specified port that are configured for input.
- [MccBoard.BoardConfig.GetDiNumDevs\(\)](#) – Gets the number of digital devices on the board.
- [MccBoard.BoardConfig.GetDmaChan\(\)](#) – Gets the DMA channel (0, 1 or 3) set for the board.
- [MccBoard.BoardConfig.GetDOutMask\(\)](#) – Determines the bits on a specified port that are configured for output.
- [MccBoard.BoardConfig.GetDtBoard\(\)](#) – Gets the number of the board with the DT connector used to connect to external memory boards.
- [MccBoard.BoardConfig.GetIntLevel\(\)](#) – Gets the interrupt level set for the board (0 for none, or 1 to 15).
- [MccBoard.BoardConfig.GetNumAdChans\(\)](#) – Gets the number of A/D channels
- [MccBoard.BoardConfig.GetNumDaChans\(\)](#) – Gets the number of D/A channels.
- [MccBoard.BoardConfig.GetNumExps\(\)](#) – Gets the number of expansion boards.
- [MccBoard.BoardConfig.GetNumIoPorts\(\)](#) – Gets the number of I/O ports used by the board.
- [MccBoard.BoardConfig.GetPANID\(\)](#) – Gets the Personal Area Network (PAN) identifier for wireless communication.
- [MccBoard.BoardConfig.GetRange\(\)](#) – Gets the selected voltage range.
- [MccBoard.BoardConfig.GetRFChannel\(\)](#) – Gets the RF channel number that a wireless device uses to communicate.
- [MccBoard.BoardConfig.GetRSS\(\)](#) – Gets the signal strength in dBm of a signal received by a remote device.
- [MccBoard.BoardConfig.GetSignal\(\)](#) – Retrieves the configured Auxiliary or DAQ Sync connection and polarity for the specified timing and control signal.
- [MccBoard.BoardConfig.GetUsesExps\(\)](#) – Gets the True/False value indicating support of expansion boards.
- [MccBoard.BoardConfig.GetWaitState\(\)](#) – Gets the value of the Wait State jumper (1-enabled, 0-disabled).
- [MccBoard.BoardConfig.SetAdRetrigCount\(\)](#) – Sets the number of samples to acquire during a trigger event when ScanOptions.RetrigMode is set.

- [MccBoard.BoardConfig.SetBaseAdr\(\)](#) – Sets the base address of a board
- [MccBoard.BoardConfig.SetClock\(\)](#) – Sets the clock source by the frequency (40, 10, 8, 6, 5, 4, 3, 2, 1), or 0 for not supported.
- [MccBoard.BoardConfig.SetDACStartup\(\)](#) – Sets the board's configuration register STARTUP bit to 0 or 1 to enable/disable the storing of digital-to-analog converter (DAC) startup values.
- [MccBoard.BoardConfig.SetDACUpdateMode\(\)](#) – Sets the update mode for a digital-to-analog converter (DAC).
- [MccBoard.BoardConfig.SetDeviceId\(\)](#) – Sets the name that identifies the instance of a device.
- [MccBoard.BoardConfig.SetDeviceNotes\(\)](#) – Sets the device notes to store in a device's memory.
- [MccBoard.BoardConfig.SetDmaChan\(\)](#) – Sets the DMA channel (0, 1 or 3).
- [MccBoard.BoardConfig.SetIntLevel\(\)](#) – Sets the interrupt level: 0 for none, or 1 to 15.
- [MccBoard.BoardConfig.SetNumAdChans\(\)](#) – Sets the number of A/D channels available on the board.
- [MccBoard.BoardConfig.SetPanID\(\)](#) – Sets the Personal Area Network (PAN) identifier used for wireless communication.
- [MccBoard.BoardConfig.SetRange\(\)](#) – Sets the selected voltage range.
- [MccBoard.BoardConfig.SetRFChannel\(\)](#) – Sets the RF channel number used for wireless communications.
- [MccBoard.BoardConfig.SetWaitState\(\)](#) – Sets the value of the Wait State jumper (1 = enabled, 0 = disabled).
- [MccBoard.CtrConfig](#) property – Gets an instance of a [cCtrConfig](#) object.
- [MccBoard.CtrConfig.GetCtrType\(\)](#) – Gets the counter device number of counter type specified with the configVal parameter.
- [MccBoard.DioConfig](#) property – Gets an instance of a [cDioConfig](#) object.
- [MccBoard.DioConfig.GetConfig\(\)](#) – Gets the configuration of a digital device (digital input or digital output).
- [MccBoard.DioConfig.GetCurVal\(\)](#) – Gets the current value of digital outputs.
- [MccBoard.DioConfig.GetDevType\(\)](#) – Gets the device type of the digital port (AUXPORT, FIRSTPORTA, etc.).
- [MccBoard.DioConfig.GetDInMask\(\)](#) – Determines the bits on a specified port that are configured for input.
- [MccBoard.DioConfig.GetDOutMask\(\)](#) – Determines the bits on a specified port that are configured for output.
- [MccBoard.DioConfig.GetNumBits\(\)](#) – Gets the number of bits in the digital port value.
- [MccBoard.ExpansionConfig](#) property – Gets an instance of a [cExpansionConfig](#) object.
- [MccBoard.ExpansionConfig.GetBoardType\(\)](#) – Gets the expansion board type.
- [MccBoard.ExpansionConfig.GetCjcChan\(\)](#) – Gets the channel that the CJC is connected to.
- [MccBoard.ExpansionConfig.GetMuxAdChan1\(\)](#) – Gets the first A/D channel that the board is connected to.
- [MccBoard.ExpansionConfig.GetMuxAdChan2\(\)](#) – Gets the second A/D channel that the board is connected to.
- [MccBoard.ExpansionConfig.GetNumExpChans\(\)](#) – Gets the number of expansion board channels.
- [MccBoard.ExpansionConfig.GetRange1\(\)](#) – Gets the range/gain of the low 16 channels.
- [MccBoard.ExpansionConfig.GetRange2\(\)](#) – Gets the range/gain of the high 16 channels.
- [MccBoard.ExpansionConfig.GetThermType\(\)](#) – Gets the type of thermocouple configuration for the board (J, K, E, T, R, S, and B types).
- [MccBoard.ExpansionConfig.SetCjcChan\(\)](#) – Sets the channel that the CJC is connected to.
- [MccBoard.ExpansionConfig.SetMuxAdChan1\(\)](#) – Sets the first A/D channel that the board is connected to.
- [MccBoard.ExpansionConfig.SetMuxAdChan2\(\)](#) – Sets the second A/D channel that the board is connected to.
- [MccBoard.ExpansionConfig.SetRange1\(\)](#) – Sets the range/gain of the low 16 channels.
- [MccBoard.ExpansionConfig.SetRange2\(\)](#) – Sets the range/gain of the high 16 channels.
- [MccBoard.ExpansionConfig.SetThermType\(\)](#) – Sets the type of thermocouple configuration for the board (J, K, E, T, R, S, and B types).
- [GlobalConfig.NumBoards](#) property – Returns the maximum number of boards you can install at one time.
- [GlobalConfig.NumExpBoards](#) property – Returns the maximum number of expansion boards you can install on a board.

- [GlobalConfig.Version property](#) – Information used by the library to determine compatibility.

## Counter Methods

The counter methods available from the [MccBoard class](#) load, read, and configure counters. There are several types of counters used in Measurement Computing devices. Some of the counter commands only apply to one type of counter.

- [MccBoard.C7266Config\(\)](#) - Selects the basic operating mode of an LS7266 counter.
- [MccBoard.C8254Config\(\)](#) - Selects the basic operating mode of an 8254 counter.
- [MccBoard.C8536Config\(\)](#) - Selects the basic operating mode of an 8536 counter chip.
- [MccBoard.C8536Init\(\)](#) - 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.
- [MccBoard.C9513Config\(\)](#) - Sets the basic operating mode of a 9513 counter. This method sets all of the programmable options that are associated with a 9513 counter. It is similar in purpose to C8254Config() except that it is used with a 9513 counter.
- [MccBoard.C9513Init\(\)](#) - 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 each counter chip, not the individual counters within it.
- [MccBoard.CClear\(\)](#) - Clears a scan counter value (sets it to zero).
- [MccBoard.CConfigScan\(\)](#) - Configures a scan counter channel. This method only works with counter boards that have counter scan capability.
- [MccBoard.CFreqIn\(\)](#) - Measures the frequency of a signal by counting it for a specified period of time (gateInterval), and then converting the count to count/sec (Hz). This method only works with 9513 counters.
- [MccBoard.CIn\(\)](#) - Reads a counter's current value as a 16-bit integer.
- [MccBoard.CIn32\(\)](#) - Reads a counter's current value as a 32-bit integer.
- [MccBoard.CIn64\(\)](#) - Reads a counter's current value as a 64-bit integer.
- [MccBoard.CInScan\(\)](#) - Scans a range of scan counter channels, and stores the samples in memory for later transfer to an array.
- [MccBoard.CLoad\(\)](#) - Loads a counter with a 16-bit integer initial value.
- [MccBoard.CLoad32\(\)](#) - Loads a counter with a 32-bit integer initial value.
- [MccBoard.CLoad64\(\)](#) - Loads a counter with a 64-bit integer initial value.
- [MccBoard.CStatus\(\)](#) - Read the counter status of a counter. Returns various bits that indicate the current state of a counter (currently only applies to LS7266 counters).
- [MccBoard.CStoreOnInt\(\)](#) - Installs an interrupt handler that stores the current count whenever an interrupt occurs. This method only works with 9513 counters.
- [MccBoard.PulseOutStart\(\)](#) - Starts a timer to generate digital pulses at a specified frequency and duty cycle.
- [MccBoard.PulseOutStop\(\)](#) - Stops a timer output.
- [MccBoard.TimerOutStart\(\)](#) - Starts a timer square wave output.
- [MccBoard.TimerOutStop\(\)](#) - Stops a timer square wave output.

## Data Logger Methods and Property

The methods and property available from the [DataLogger class](#) are explained below. These class members read and convert binary log files.

- [DataLogger.ConvertFile\(\)](#) - Converts a binary log file to a comma-separated values (.CSV) text file or another text file format that you specify.
- [DataLogger.GetAIChannelCount\(\)](#) - Retrieves the total number of analog input channels logged in a binary file.
- [DataLogger.GetAIInfo\(\)](#) - Retrieves the channel number and unit value of each analog input channel logged in a binary file.
- [DataLogger.GetCJCInfo\(\)](#) - Retrieves the number of CJC temperature channels logged in a binary file.
- [DataLogger.GetDIOInfo\(\)](#) - Retrieves the number of digital I/O channels logged in a binary file.
- [DataLogger.GetFileInfo\(\)](#) - Retrieves the version level and byte size of a binary file.
- [DataLogger.GetFileName\(\)](#) - Retrieves the name of the n<sup>th</sup> file in the directory containing binary log files.
- [DataLogger.GetPreferences\(\)](#) - Retrieves API preference settings for time stamp data, analog temperature data, and CJC temperature data. Returns the default values unless changed using [SetPreferences\(\)](#).
- [DataLogger.GetSampleInfo\(\)](#) - retrieves the sample interval, sample count, and the date and time of the first data point in a binary file.
- [DataLogger.ReadAIChannels\(\)](#) - Retrieves analog input data from a binary file, and stores the values in an array.
- [DataLogger.ReadCJCChannels\(\)](#) - Retrieves CJC temperature data from a binary file, and stores the values in an array.
- [DataLogger.ReadDIOChannels\(\)](#) - Retrieves digital I/O channel data from a binary file, and stores the values in an array.
- [DataLogger.ReadTimeTags\(\)](#) - Retrieves date and time values logged in a binary file. This method stores date values in the dateTags array, and time values in the timeTags array.
- [DataLogger.SetPreferences\(\)](#) - Sets preferences for returned time stamp data, analog temperature data, and CJC temperature data.
- [DataLogger.FileName property](#) - Returns the file name associated with an instance of the DataLogger class.

## Digital I/O Methods

The digital methods available from the MccBoard class are explained below. These methods perform digital input and output on various types of digital I/O ports.

- [MccBoard.DBitIn\(\)](#) - Reads a single bit from a digital input port.
- [MccBoard.DBitOut\(\)](#) - Sets a single bit on a digital output port.
- [MccBoard.DConfigBit\(\)](#) - Configures a specific digital bit as input or output.
- [MccBoard.DConfigPort\(\)](#) - Selects whether a digital port is an input or an output.
- [MccBoard.DIn\(\)](#) - Reads a specified digital input port.
- [MccBoard.DInScan\(\)](#) - Reads a set number of bytes or words from a digital input port at a specified rate.
- [MccBoard.DOut\(\)](#) - Writes a byte to a digital output port.
- [MccBoard.DOutScan\(\)](#) - Writes a series of bytes or words to a digital output port at a specified rate.

## Error Handling Methods and Properties

Most UL for .NET methods return `ErrorInfo` objects. The [MccService](#) class includes one method that determines how errors are handled internally by the library. The [ErrorInfo](#) class includes properties that provide information returned by the method called.

- [MccService.ErrHandling\(\)](#) - Sets the manner of reporting and handling errors for all method calls.
- [ErrorInfo.Message property](#) - Gets the text of the error message associated with a specific error code.
- [ErrorInfo.Value property](#) - Gets the error constant associated with an [ErrorInfo](#) object.
- [ErrorInfo.LogToFile property](#) - When set *true*, records time-stamped error codes to a file.



## Memory Board Methods

The memory board methods available from the [MccBoard class](#) read and write data to and from a memory board, and also set modes that control memory boards (MEGA-FIFO).

The most common use for 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, use the ExtMemory option with the [MccBoard.AInScan\(\)](#) or [MccBoard.APretrig\(\)](#) methods.

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

- [MccBoard.MemSetDTMode\(\)](#) - Set DT-Connect mode on a memory board. 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 method configures the port.
- [MccBoard.MemReset\(\)](#) - 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 method resets the current address to the location 0.
- [MccBoard.MemRead\(\)](#) - Reads a specified number of points from a memory board starting at a specified address.
- [MccBoard.MemWrite\(\)](#) - Writes a specified number of points to a memory board starting at a specified address.
- [MccBoard.MemReadPretrig\(\)](#) - Reads data collected with [MccBoard.APretrig\(\)](#). The [MccBoard.APretrig\(\)](#) method writes the pre-triggered data to the memory board in a scrambled order. This method unscrambles the data and returns it in the correct order.

## Revision Control Methods

The revision control methods and property explained below are available from the [MccBoard class](#). As new revisions of the library are released, bugs from previous revisions are fixed, and occasionally new properties and methods are added. It is our goal to preserve the programs you have written so that you never change the order or number of arguments in a method. However, sometimes this is not possible.

The revision control methods initialize the DLL so that the functions are interpreted according to the format of the revision you wrote and used to compile the program.

- [MccBoard.DeclareRevision\(\)](#) – Declares the revision number of the Universal Library for .NET that your program was written with.
- [MccBoard.GetRevision\(\)](#) – Returns the version number of the installed Universal Library for .NET.

## Streamer File Methods

The streamer file methods available from the [MccBoard class](#) create, fill, and read streamer files.

- [MccBoard.FileAInScan\(\)](#) - Transfer analog input data directly to file. Very similar to [AInScan\(\)](#) except that the data is stored in a file instead of memory.
- [MccBoard.FilePretrig\(\)](#) - Pre-triggered analog input to a file. Very similar to [APretrig\(\)](#) except that the data is stored in a file instead of memory.
- [MccBoard.FileGetInfo\(\)](#) - Reads streamer file information on how much data is in the file, and the conditions under which it was collected (sampling rate, channels, etc.).
- [MccBoard.FileRead\(\)](#) - Reads a selected number of data points from a streamer file into a one-dimensional or two-dimensional array.

## Synchronous Methods

The synchronous methods available from the [MccBoard class](#) synchronously read data from analog, counter, thermocouple, and digital input channels, write data to analog or digital output channels, or configure devices that support synchronous I/O.

- [MccBoard.DaqInScan\(\)](#) – Scans analog, digital, temperature, and counter inputs synchronously, and stores the values in memory.
- [MccBoard.DaqOutScan\(\)](#) – Outputs values synchronously to analog output channels and digital output ports.
- [MccBoard.DaqSetSetpoints\(\)](#) – Configures up to 16 detection setpoints associated with the input channels within a scan group.
- [MccBoard.DaqSetTrigger\(\)](#) – Selects a trigger source and sets up its parameters. This method starts or stops a synchronous data acquisition operation using [DaqInScan\(\)](#) with the `ExtTrigger` option.

## Temperature Input Methods

The temperature input methods available from the [MccBoard class](#) convert a raw analog input from an EXP or other temperature sensor board to temperature.

- [MccBoard.TIn\(\)](#) - Reads a channel from a digital input board, filters it (if specified), does the cold junction compensation, linearizes and converts it to temperature.
- [MccBoard.TInScan\(\)](#) - Scans a range of temperature inputs. Reads temperatures from a range of channels, and returns the temperature values to an array.

## Windows Memory Management Methods

The Windows memory management methods available from the [MccService class](#) take care of allocating, freeing, and copying to/from Windows global memory buffers.

- [MccService.WinBufAlloc\(\)](#) - Allocates a Windows memory buffer.
- [MccService.WinBufAlloc32\(\)](#) - Allocates a Windows global memory buffer for use with 32-bit scan functions, and returns a memory handle for the buffer.
- [MccService.WinBufAlloc64\(\)](#) - Allocates a Windows global memory buffer large enough for double precision data values, and returns a memory handle for the buffer.
- [MccService.WinBufFree\(\)](#) - Frees a Windows buffer.
- [MccService.WinArrayToBuf\(\)](#) - Copies data from a one-dimensional or two-dimensional array into a Windows buffer.
- [MccService.WinBufToArray\(\)](#) - Copies data from a Windows memory buffer into a one-dimensional or two-dimensional array.
- [MccService.WinBufToArray32\(\)](#) - Copies 32-bit data from a Windows global memory buffer into an array. This method is typically used to retrieve data from the buffer after executing an input scan method.
- [MccService.ScaledWinArrayToBuf\(\)](#) - Copies double precision values from an array into a Windows memory buffer.
- [MccService.ScaledWinBufAlloc\(\)](#) - Allocates a Windows global memory buffer large enough to hold scaled data obtained from scan operations in which the ScaleData option is selected, and returns a memory handle for the buffer.
- [MccService.ScaledWinBufToArray\(\)](#) - Copies double precision values from a Windows memory buffer into an array.

## Miscellaneous Methods

The methods explained below are available from the [MccBoard class](#). These methods do not as a group fit into a single category. They get and set board information, convert units, manage events and background operations, copy two-dimensional arrays to/from Windows global memory buffers, and perform serial communication operations.

- [MccBoard.DeviceLogin\(\)](#) - Opens a device session with a shared device.
- [MccBoard.DeviceLogout\(\)](#) - Releases the device session with a shared device.
- [MccBoard.DisableEvent\(\)](#) - Disables one or more events set up with [EnableEvent\(\)](#) and disconnects their user-defined handlers
- [MccBoard.EnableEvent\(\)](#) - Binds one or more event conditions to a user-defined callback function.
- [EventCallback](#) delegate - Defines the prototype for the user function for [EnableEvent\(\)](#). This defines the format for the user-defined handlers to be called when the events set up using [EnableEvent\(\)](#) occurs.
- [MccBoard.EngArrayToWinBuf\(\)](#) - Transfers a 2D array of engineering unit values to a Windows buffer as integer values.
- [MccBoard.FlashLED\(\)](#) - Causes the LED on a USB device to flash.
- [MccBoard.FromEngUnits\(\)](#) - Converts a single precision voltage (or current) value in engineering units to an integer D/A count value for output to a D/A.
- [MccBoard.GetBoardName\(\)](#) - Returns the name of a specified board.
- [MccBoard.GetStatus\(\)](#) - Returns the status of a background operation. Once a background operation starts, your program must periodically check on its progress. This method returns the current status of the operation.
- [MccBoard.GetTCValues\(\)](#) - Converts raw thermocouple data gathered with [DaqInScan\(\)](#) to Celsius, Fahrenheit or Kelvin.
- [MccBoard.HideLoginDialog\(\)](#) - Prevents the default login dialog from being shown when a protected function is called while not logged in.
- [MccBoard.InByte\(\)](#) - Reads a byte from a hardware register on a board.
- [MccBoard.InWord\(\)](#) - Reads a word from a hardware register on a board.
- [MccBoard.OutByte\(\)](#) - Writes a byte to a hardware register on a board.
- [MccBoard.OutWord\(\)](#) - Writes a byte or word to a hardware register on a board.
- [MccBoard.RS485\(\)](#) - Sets the transmit and receive buffers on an RS485 port.
- [MccBoard.StopBackground\(\)](#) - Stop a background process. It is sometimes necessary to stop a background process even though the process has been set up to run continuously. This method stops a background process that is running. [StopBackground\(\)](#) should be executed after normal termination of all background functions in order to clear variables and flags.
- [MccBoard.TEDSRead\(\)](#) - Reads data from a TEDS sensor into an array.
- [MccBoard.ToEngUnits\(\)](#) - Converts an integer A/D count value to an equivalent single precision voltage (or current) value.
- [MccBoard.ToEngUnits32\(\)](#) - Converts an integer count value to an equivalent double precision voltage (or current) value.
- [MccBoard.WinBufToEngArray\(\)](#) - Transfers integer values from a Windows buffer to a 2D array as engineering unit values.
- [MccBoard.BoardName property](#) - Name of the board associated with an instance of the [MccBoard class](#).

## cbAConvertData() function

### Changed R3.3 RW

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 is used. For many 12-bit A/D boards, the raw data is a 16-bit value that contains a 12-bit A/D value and a 4 bit channel tag (refer to board-specific information in the *Universal Library User's Guide*). The data returned to ADData consists of just the 12-bit A/D value. The data returned to ChanTags consists of just the channel numbers.

## Function Prototype

C/C++

```
int cbAConvertData(int BoardNum, long NumPoints, unsigned short ADData[], unsigned short ChanTags[])
```

Visual Basic

```
Function cbAConvertData(ByVal BoardNum%, ByVal NumPoints%, ADData%, ChanTags%) As Long
```

## Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal.

*NumPoints*

Number of samples to convert.

*ADData*

Pointer or reference to start of data array.

*ChanTags*

Pointer or reference to the start of the channel tag array.

When collecting data using [cbAInScan\(\)](#) without the CONVERTDATA option, use this function to convert the data after it has been collected. There are cases where the CONVERTDATA option is not allowed. For example - if you are using both the DMAIO and BACKGROUND option with [cbAInScan\(\)](#) on some devices, the CONVERTDATA option is not allowed. In those cases this function should be used to convert the data after the data collection is complete.

For some boards, 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

## Notes

- 12-bit A/D Boards

The name of the array must match that used in [cbAInScan\(\)](#) or [cbWinBufToArray\(\)](#).

Upon returning from cbAConvertData(), ADData array contains only 12-bit A/D data.

- 16-bit A/D Boards

This function is not for use with 16-bit A/D boards because 16-bit boards do not have channel tags. The argument BoardNum was added in revision 3.3 to prevent applying this function to 16-bit data. If you wrote your program for a 12-bit board then later upgrade to a 16-bit board, all you need change is the InstaCal configuration file. If this function is called for a 16-bit board, it is simply ignored, and no errors are generated.



## cbAConvertPretrigData() function

### Changed R3.3 RW

For products with pretrigger implemented in hardware (most products), this function converts and aligns the raw data collected by [cbAPretrig\(\)](#). The [cbAPretrig\(\)](#) 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. This function correctly orders the data also, starting with the first pretrigger data point and ending with the last post-trigger point.

Change at revision 3.3 is to support multiple background tasks. It is now possible to run two boards with DMA or REP-INSW background convert-and-transfer features active, therefore, the convert function must know which board the data came from. The data value assigned to BoardNum should be assigned in the header file so it is easy to locate if a change is needed.

## Function Prototype

C/C++

```
int cbAConvertPretrigData(int BoardNum, long PretrigCount, long TotalCount, unsigned short ADData[], unsigned short ChanTags[])
```

Visual Basic

```
Function cbAConvertPretrigData(ByVal BoardNum&, ByVal PretrigCount&, ByVal TotalCount&, ADData%, ChanTags%) As Long
```

## Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal.

*PretrigCount*

Number of pre-trigger samples. This value must match the value returned by the PretrigCount argument in the [cbAPretrig\(\)](#) function.

*TotalCount*

Total number of samples that were collected.

*ADDData*

Pointer to the data array. this value must match the array name used in the [cbAPretrig\(\)](#) function.

*ChanTags*

Pointer to the channel tag array or a NULL pointer may be passed if using 16-bit boards or if channel tags are not desired; see the note regarding 16-bit boards below.

## Returns

- [Error code](#) or 0 if no errors.
- ADDData - converted data.

## Notes

- When you collect data with [cbAPretrig\(\)](#) and you don't use the CONVERTDATA option, 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\(\)](#) on some devices, the CONVERTDATA option is not allowed. In those cases this function should be used to convert the data after the data collection is complete.

- 12-Bit A/D boards

On some 12-bit boards, 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 ADDData and the channel number into the ChanTags array.

The name of the ADDData array must match that used in [cbAInScan\(\)](#) or [cbWinBufToArray\(\)](#).

Upon returning from cbAConvertPretrigData(), ADDData array contains only 12-bit A/D data.

- 16-Bit A/D boards

This function is for use with 16-bit A/D boards only insofar as ordering the data. No channel tags are returned.

## Visual Basic Programmers

After the data is collected with [cbAPretrig\(\)](#) it must be copied to an array with [cbWinBufToArray\(\)](#).

**Important:** The entire array must be copied. This array includes the extra 512 samples needed by [cbAPretrig\(\)](#). Example code is given below:

```
Count& = 10000  
  
Dim ADData%(Count& + 512)  
Dim ChanTags%(Count& + 512)  
  
cbAPretrig%(BoardNum, LowChan, HighChan, PretrigCount&, Count&...)  
cbWinBufToArray%(MemHandle%, ADData%, Count& + 512)  
cbAConvertPretrigData%(PretrigCount&, Count&, ADData%, ChanTags%)
```

## cbACalibrateData() function

### New R3.3

Calibrates the raw data collected by [cbAInScan\(\)](#) from boards with real time software calibration when the real time calibration has been turned off. The [cbAInScan\(\)](#) function can return either raw A/D data or calibrated data, depending on whether or not the NOCALIBRATEDATA option was used.

## Function Prototype

C/C++

```
int cbACalibrateData(int BoardNum, long NumPoints, int Range, unsigned ADData[ ])
```

Visual Basic

```
Function cbACalibrateData(ByVal BoardNum&, ByVal NumPoints&, ByVal Range&, ADData%) As Long
```

## Arguments

*BoardNum*

May be 0 to 99. Refers to the number associated with the board when it was installed with InstaCal.

*NumPoints*

Number of samples to convert

*Range*

The programmable gain/range used when the data was collected. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*ADDData*

Pointer to data array.

## Returns

- [Error code](#) or 0 if no errors.
- ADDData - converted data.

## Notes

- When collecting data using [cbAInScan\(\)](#) with the NOCALIBRATEDATA option, use this function to calibrate the data once collected.
- The name of the array must match that used in [cbAInScan\(\)](#) or [cbWinBufToArray\(\)](#).
- Applying software calibration factors in real time on a per sample basis eats up machine cycles. If your CPU is slow, or if processing time is at a premium, do not calibrate until the acquisition run finishes. Turn off real time software calibration to save CPU time during high speed acquisitions by using the NOCALIBRATEDATA option to a turn off real-time software calibration. After the acquisition is run, calibrate the data with [cbACalibrateData\(\)](#).

## cbAIn() function

Reads an A/D input channel from the specified board, and returns a 16-bit unsigned integer value. 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.

## Function Prototype

C/C++

```
int cbAIn(int BoardNum, int Channel, int Range, unsigned short *DataValue);
```

Visual Basic

```
Function cbAIn(ByVal BoardNum&, ByVal Channel&, ByVal Range&, DataValue%) As Long
```

## Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed using InstaCal. The specified board must have an A/D.

*Channel*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured. For example, a USB-1608GX device has 8 differential or 16 single-ended analog input channels. Expansion boards also support this function, so this argument can contain values up to 272. See board specific information for EXP boards if you are using an expansion board.

*Range*

A/D Range code. If the selected A/D board does not have a programmable gain feature, this argument is ignored. If the A/D board does have programmable gain, set the Range argument to the desired A/D range. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*DataValue*

Pointer or reference to the data value.

## Returns

- [Error code](#) or 0 if no errors.
- DataValue - Returns the value of the A/D sample.

## cbAIn32() function

Reads an A/D input channel from the specified board, and returns a 32-bit unsigned integer value. 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. In general, this function should be used with devices with a resolution higher than 16-bits.

### Function Prototype

C/C++

```
int cbAIn32(int BoardNum, int Chan, int Gain, ULONG *DataValue, int Options);
```

Visual Basic

```
Function cbAIn32(ByVal BoardNum&, ByVal Chan&, ByVal Gain&, DataValue&, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed using InstaCal. The specified board must have an A/D.

*Chan*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured. For example, a USB-2416 device has 16 differential or 32 single-ended analog input channels. Expansion boards also support this function, so this argument can contain values up to 272. See board specific information for EXP boards if you are using an expansion board.

*Gain*

A/D Range code. If the selected A/D board does not have a programmable gain feature, this argument is ignored. If the A/D board does have programmable gain, set the Range argument to the desired A/D range. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*DataValue*

Pointer or reference to the data value.

*Options*

Reserved for future use.

### Returns

- [Error code](#) or 0 if no errors.
- DataValue - Returns the value of the A/D sample.

## cbAInScan() function

### Changed R3.3 ID

Scans a range of A/D channels and stores the samples in an array. cbAInScan() 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 the data array.

**Changes:** Revision 3.3 added a "no real time calibration" option.

## Function Prototype

C/C++

```
int cbAInScan(int BoardNum, int LowChan, int HighChan, long Count, long *Rate, int Range, int MemHandle, int Options)
```

Visual Basic

```
Function cbAInScan(ByVal BoardNum&, ByVal LowChan&, ByVal HighChan&, ByVal Count&, Rate&, ByVal Range&, ByVal MemHandle&, ByVal Options&) As Long
```

## Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed using InstaCal. The specified board must have an A/D.

*LowChan*

The first A/D channel of scan. When [cbALoadQueue\(\)](#) is used, the channel count is determined by the total number of entries in the channel gain queue, and LowChan is ignored.

*HighChan*

The last A/D channel of scan. When [cbALoadQueue\(\)](#) is used, the channel count is determined by the total number of entries in the channel gain queue, and HighChan is ignored.

Low / High Channel number: 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 channels for differential, 16 for single ended.

*Count*

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

*Rate*

The rate at which samples are acquired, in samples per second per channel.

For example, if you sample four channels, 0-3, at a rate of 10,000 scans per second (10 kHz), the resulting A/D converter rate is 40 kHz: four 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.

The channel count is determined by the LowChan and HighChan parameters. Channel Count = (HighChan - LowChan + 1).

When cbALoadQueue is used, the channel count is determined by the total number of entries in the channel gain queue. LowChan and HighChan are ignored.

Rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*Range*

A/D range code. If the selected A/D board does not have a programmable range feature, this argument is ignored. Otherwise, set the Range argument to any range that is supported by the selected A/D board. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*MemHandle*

Handle for Windows buffer to store data in (Windows). This buffer must have been previously allocated with either [cbWinBufAlloc\(\)](#) or [cbWinBufAlloc32\(\)](#).

*Options*

Bit fields that control various options. This field may contain any combination of non-contradictory choices from the values listed in the [Options argument values](#) section below.

## Returns

- [Error code](#) or 0 if no errors.

- Rate - Actual sampling rate used.
- MemHandle - Collected A/D data returned via the Windows buffer.

## Options argument values

Transfer method options	<p>The following four options determine how data is transferred from the board to PC memory. If none of these options are specified (recommended), the optimum sampling mode is automatically chosen based on board type and sampling speed. Use the default method unless you have a reason to select a specific transfer method.</p> <ul style="list-style-type: none"> <li>■ SINGLEIO A/D transfers to memory are initiated by an interrupt. One interrupt per conversion. Rates attainable using SINGLEIO are PC-dependent and generally less than 10 kHz.</li> <li>■ DMAIO A/D transfers are initiated by a DMA request.</li> <li>■ BLOCKIO A/D transfers are handled in blocks (by REP-INSW for example). <b>BLOCKIO is not recommended for slow acquisition rates.</b> If the rate of acquisition is very slow (for example less than 200 Hz) BLOCKIO is probably not the best choice for transfer mode. The reason for this is that status for the operation is not available until one packet of data has been collected (typically 512 samples). The implication is that if acquiring 100 samples at 100 Hz using BLOCKIO, the operation will not complete until 5.12 seconds has elapsed.</li> <li>■ BURSTIO Allows higher sampling rates for sample counts up to full FIFO. Data is collected into the local FIFO. Data transfers to the PC are held off until after the scan is complete. For BACKGROUND scans, the count and index returned by <a href="#">cbGetStatus()</a> remain 0 and the status equals RUNNING until the scan finishes. When the scan is complete and the data is retrieved, the count and index are updated and the status equals IDLE. BURSTIO is the default mode for non-Continuous fast scans (aggregate sample rates above 1000 Hz) with sample counts up to full FIFO. To avoid the BURSTIO default, specify BLOCKIO. BURSTIO is not a valid option for most boards. It is used mainly for USB products.</li> </ul>
BACKGROUND	<p>If the BACKGROUND option is not used then the <a href="#">cbAInScan()</a> function will not return to your program until all of the requested data has been collected and returned to the buffer. 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 the buffer will continue in the background. Use <a href="#">cbGetStatus()</a> with AIFUNCTION to check on the status of the background operation. Alternatively, some boards support <a href="#">cbEnableEvent()</a> for event notification of changes in status of BACKGROUND scans. Use <a href="#">cbStopBackground()</a> with AIFUNCTION to terminate the background process before it has completed. <a href="#">cbStopBackground()</a> should be executed after normal termination of all background functions in order to clear variables and flags.</p>
BURSTMODE	<p>Enables burst mode sampling. Scans from LowChan to HighChan are clocked at the maximum A/D rate in order to minimize channel to channel skew. Scans are initiated at the rate specified by the <i>Rate</i> argument.</p> <p>BURSTMODE is not recommended for use with the SINGLEIO option. If this combination is used, the Count value should be set as low as possible, preferably to the number of channels in the scan. Otherwise, overruns may occur.</p>
CONVERTDATA	<p>This option is used to align data, either within each byte (in the case of some 12-bit devices) or within the buffer (see the <a href="#">cbAPreTrig()</a> function). Only the former case applies for <a href="#">cbAInScan()</a>. The data stored on some 12-bit devices is offset in the devices data register. For these devices, the CONVERTDATA option converts the data to 12-bit A/D values by shifting the data to the first 12 bits within the byte. For devices that store the data without an offset and for all 16-bit devices, this option is ignored.</p> <p>Use of CONVERTDATA is recommended unless one of the following two conditions exist: 1) On some devices, CONVERTDATA may not be specified if you are using the BACKGROUND option and DMA transfers. In this case, if data conversion is required, use <a href="#">cbAConvertData()</a> to re-align the data. 2) Some 12-bit boards store the data as a 12-bit A/D value and a 4-bit channel number. Using CONVERTDATA will strip out the channel number from the data. If you prefer to store the channel number as well as the data, call <a href="#">cbAConvertData()</a> to retrieve the data and the channel number from the buffer after the data acquisition to the buffer is complete.</p>
CONTINUOUS	<p>This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of the buffer and begins again. The only way to stop this operation is with <a href="#">cbStopBackground()</a>. Normally this option should be used in combination with BACKGROUND so that your program will regain control.</p> <p><b>Count argument settings in CONTINUOUS mode:</b> For some DAQ hardware, Count must</p>

	<p>be an integer multiple of the packet size. Packet size is the amount of data that a DAQ device transmits back to the PC's memory buffer during each data transfer. Packet size can differ among DAQ hardware, and can even differ on the same DAQ product depending on the transfer method.</p> <p>In some cases, the minimum value for the Count argument may change when the CONTINUOUS option is used. This can occur for several reasons; the most common is that in order to trigger an interrupt on boards with FIFOs, the circular buffer must occupy at least half the FIFO. Typical half-FIFO sizes are 256, 512 and 1,024.</p> <p>Another reason for a minimum Count value is that the buffer in memory must be periodically transferred to the user buffer. If the buffer is too small, data will be overwritten during the transfer resulting in garbled data.</p> <p>Refer to board-specific information in the <i>Universal Library User's Guide</i> for packet size information for your particular DAQ hardware.</p>
DTCONNECT	<p>All A/D values will be sent to the A/D board's DT-Connect port. This option is incorporated into the EXTMEMORY option. Use DTCNNCT only if the external board is not supported by Universal Library.</p>
EXTCLOCK	<p>If this option is used, conversions will be controlled by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal (refer to the board-specific information in the <i>Universal Library User's Guide</i>). In most cases, when this option is used the Rate argument is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to a transfer mode that will allow the maximum conversion rate to be attained unless otherwise specified.</p> <p>In some cases, such as with the PCI-DAS4020/12, an approximation of the rate is used to determine the size of the packets to transfer from the board. Set the Rate argument to an approximate maximum value.</p> <p><b>SINGLEIO is recommended for slow external clock rates:</b> If the rate of the external clock is very slow (for example less than 200 Hz) and the board you are using supports BLOCKIO, you may want to include the SINGLEIO option. The reason for this is that the status for the operation is not available until one packet of data has been collected (typically 512 samples). The implication is that, if acquiring 100 samples at 100 Hz using BLOCKIO (the default for boards that support it if EXTCLOCK is used), the operation will not complete until 5.12 seconds has elapsed.</p>
EXTMEMORY	<p>Causes the command to send the data to a connected memory board via the DT Connect interface rather than returning the data to the buffer. Data for each call to this function will be appended unless <a href="#">cbMemReset()</a> is called. The data should be unloaded with the <a href="#">cbMemRead()</a> function before collecting new data. When EXTMEMORY option is used, the MemHandle argument can be set to null or 0. CONTINUOUS option cannot be used with EXTMEMORY. Do not use EXTMEMORY and DTCNNCT together. The transfer modes DMAIO, SINGLEIO, BLOCKIO and BURSTIO have no meaning when used with this option.</p>
EXTTRIGGER	<p>If this option is specified, the sampling will not begin until the trigger condition is met. On many boards, this trigger condition is programmable (refer to the <a href="#">cbSetTrigger()</a> function and board-specific information for details) and can be programmed for rising or falling edge or an analog level.</p> <p>On other boards, only 'polled gate' triggering is supported. In this case, assuming active high operation, data acquisition will commence immediately if the trigger input is high. If the trigger input is low, acquisition will be held off until it goes high. Acquisition will then continue until NumPoints&amp; samples have been taken regardless of the state of the trigger input. For "polled gate" triggering, this option is most useful if the signal is a pulse with a very low duty cycle (trigger signal in TTL low state most of the time) so that triggering will be held off until the occurrence of the pulse.</p>
HIGHRESRATE	<p>Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>Rate</i> argument above).</p>
NOCALIBRATEDATA	<p>Turns off real-time software calibration for boards which are software calibrated. This is done by applying calibration factors to the data on a sample by sample basis as it is acquired. Examples are the PCM-DAS16/330 and PCM-DAS16x/12. Turning off software calibration saves CPU time during a high speed acquisition run. This may be required if your processor is less than a 150 MHz Pentium and you desire an acquisition speed in excess of 200 kHz. These numbers may not apply to your system. Only trial will tell for sure. DO NOT use this option if you do not have to. If this option is used, the data must be calibrated after the acquisition run with the <a href="#">cbACalibrateData()</a> function.</p>
NOTODINTS	<p>If this option is specified, the system's time-of-day interrupts are disabled for the duration of the scan. These interrupts are used to update the systems real time clock and are also used by various other programs. These interrupts can limit the maximum sampling speed of some boards - particularly the PCM-DAS08. If the interrupts are turned off using this option, the real-time clock will fall behind by the length of time that the scan takes.</p>
RETRIGMODE	<p>Re-arms the trigger after a trigger event is performed. With this mode, the scan begins when a trigger event occurs. When the scan completes, the trigger is re-armed to acquire the next the batch of data. You can specify the number of samples in the scan for each</p>



	<p>trigger event (described below). The RETRIGMODE option can be used with the CONTINUOUS option to continue arming the trigger until <code>cbStopBackground()</code> is called.</p> <p>You can specify the number of samples to acquire with each trigger event. This is the trigger count. Use the <a href="#">cbSetConfig()</a> ConfigItem option BIADCTRIGCOUNT to set the trigger count. If you specify a trigger count that is either zero or greater than the value of the <code>cbAInScan()</code> Count argument, the trigger count will be set to the value of the Count argument.</p> <p>Specify the CONTINUOUS option with the trigger count set to zero to fill the buffer with Count samples, re-arm the trigger, and refill the buffer upon the next trigger.</p>
SCALEDATA	<p>Converts raw scan data — to voltage, temperature, and so on, depending upon the selected channel sensor category — during the analog input scan, and puts the scaled data directly into the user buffer. The user buffer should have been allocated with <a href="#">cbScaledWinBufAlloc()</a>.</p>

### 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 `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  $8 \times 20,000 = 160,000$ , but the CIO-DAS16/JR is capable of converting only 120,000 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.

### Important

In order to understand the functions, you must read the board-specific information found in the *Universal Library User's Guide*. The example programs should be examined and run prior to attempting any programming of your own. Following this advice will save you hours of frustration, and possibly time wasted holding for technical support.

This note, which appears elsewhere, is especially applicable to this function. Now is the time to read the board specific information for your board that is contained in the *Universal Library User's Guide*. We suggest that you make a copy of this information for reference as you read this manual and examine the example programs.

## cbALoadQueue() function

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

Some products do not support channel/gain queue, and some that do support it are limited on the order of elements, number of elements, and gain values that can be included, etc. Refer to the device-specific information in the *Universal Library User's Guide* to find details for your particular product.

### Function Prototype

C/C++

```
int cbALoadQueue(int BoardNum, short ChanArray[], short GainArray[], int Count)
```

Visual Basic

```
Function cbALoadQueue(ByVal BoardNum&, ChanArray%, GainArray%, ByVal Count&) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed using InstaCal. The specified board must have an A/D and a channel/gain queue.

*ChanArray*

Array containing channel values. This array should contain all of the channels that will be loaded into the channel gain queue.

*GainArray*

Array containing A/D range values. This array should contain each of the A/D ranges that will be loaded into the channel gain queue.

*Count*

Number of elements in ChanArray and GainArray or 0 to disable channel/gain queue. Specifies the total number of channel/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 channel/gain queue. The maximum value is specific to the queue size of the A/D boards channel gain queue.

### Returns

- [Error code](#) or 0 if no errors.

### Notes

- 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 channel/range pairs that you have loaded into the queue.

## cbAOut() function

Sets the value of a D/A output.

### Function Prototype

C/C++

```
int cbAOut(int BoardNum, int Channel, int Range, unsigned short DataValue)
```

Visual Basic

```
Function cbAOut(ByVal BoardNum&, ByVal Channel&, ByVal Range&, ByVal DataValue%) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed using InstaCal. The specified board must have a D/A.

*Channel*

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

*Range*

D/A range code. The output range of the D/A channel can be set to any of those supported by the board. If the D/A board does not have programmable ranges then this argument will be ignored. Refer to board specific information for a list of the supported A/D ranges.

*DataValue*

Value to set D/A to. Must be in the range 0 -  $n$  where  $n$  is the value  $2^{\text{Resolution}} - 1$  of the converter.

Exception: using 16-bit boards with Basic range is -32,768 to 32,767. Refer to the discussion on [16-bit values using a signed integer data type](#) for more information.

### Returns

- [Error code](#) or 0 if no errors.

### Notes

- "Simultaneous Update" or "Zero Power-Up" boards: If you set the simultaneous update jumper for simultaneous operation, 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.

## cbAPretrig() function

Waits for a trigger to occur and then returns a specified number of analog samples before and after the trigger occurred. If only 'polled gate' triggering is supported, the trigger input line (refer to the user's manual for the board) must be at TTL low before this function is called, or a [TRIGSTATE](#) error will occur. The trigger occurs when the trigger condition is met. Refer to the [cbSetTrigger\(\)](#) function for more details.

## Function Prototype

C/C++

```
int cbAPretrig(int BoardNum, int LowChan, int HighChan, long *PretrigCount, long *TotalCount, long *Rate, int Range, int MemHandle, int Options)
```

Visual Basic

```
Function cbAPretrig(ByVal BoardNum&, ByVal LowChan&, ByVal HighChan&, PretrigCount&, TotalCount&, Rate&, ByVal Range&, ByVal MemHandle&, ByVal Options&) As Long
```

## Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99. The specified board must have an A/D.

*LowChan*

First A/D channel of scan.

*HighChan*

Last A/D channel of scan.

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

*PretrigCount*

Number of pre-trigger A/D samples to collect. Specifies the number of samples to collect before the trigger occurs.

For products using a hardware implementation of pretrigger (most products), PretrigCount must be less than (TotalCount - 512). For these devices, if the trigger occurs too early, fewer than the requested number of pre-trigger samples will be collected, and a TOOFEW error will occur. The PretrigCount will be set to indicate how many samples were actually collected. The post trigger samples will still be collected.

For software implementations of pretrigger, PretrigCount must be less than TotalCount. For these devices, triggers that occur before the requested number of pre-trigger samples are collected are ignored. See board-specific information contained in the *UL Users Guide* for details.

*TotalCount*

Total number of A/D samples to collect. Specifies the total number of samples that will be collected and stored in the buffer.

For products using a hardware implementation of pretrigger (most products), TotalCount must be greater than or equal to the PretrigCount + 512. If the trigger occurs too early, fewer than the requested number of samples will be collected, and a TOOFEW error will occur. The TotalCount will be set to indicate how many samples were actually collected.

For software implementations of pretrigger, TotalCount must be greater than PretrigCount. For these devices, triggers that occur before the requested number of pre-trigger samples are collected are ignored. See board-specific information.

TotalCount must be evenly divisible by the number of channels being scanned. If it is not, this function will adjust the number (down) to the next valid value and return that value to the TotalCount argument.

PretrigCount must also be evenly divisible by the number of channels being scanned. If it is not, this function will adjust the number (up) to the next valid value and return that value to the PretrigCount argument.

*Rate*

Sample rate in scans per second.

*Range*

A/D Range code. If the selected A/D board does not have a programmable gain feature, this argument is ignored. Otherwise, set to any range that is supported by the selected A/D board. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

## MemHandle

Handle for Windows buffer to store data. This buffer must have been previously allocated with the [cbWinBufAlloc\(\)](#) function.

For hardware trigger types, the buffer referenced by MemHandle must be big enough to hold at least TotalCount + 512 integers.

## Options

Bit fields that control various options. This field may contain any combination of non-contradictory choices from the values listed in the [Options argument values](#) section below.

## Returns

- [Error code](#) or 0 if no errors.
- PretrigCount - Number of pre-trigger samples.
- TotalCount - Total number of samples collected.
- Rate - The actual sampling rate.
- MemHandle - Collected A/D data returned via the Windows buffer.

## Options argument values

BACKGROUND	<p>If the BACKGROUND option is not used, the cbAPretrig() function will not return to your program until all of the requested data has been collected and returned to the buffer. When the BACKGROUND option is used, control returns immediately to the next line in your program, and the data collection from the A/D into the buffer will continue in the background. Use <a href="#">cbGetStatus()</a> with AIFUNCTION to check on the status of the background operation. Alternatively, some boards support <a href="#">cbEnableEvent()</a> for event notification of changes in status of BACKGROUND scans. Use <a href="#">cbStopBackground()</a> with AIFUNCTION to terminate the background process before it has completed.</p> <p>Call cbStopBackground() after normal termination of all background functions to clear variables and flags.</p> <p>For hardware trigger types, you cannot use the CONVERTDATA option in combination with the BACKGROUND option for this function. To correctly order and parse the data, use <a href="#">cbAConvertPretrigData()</a> after the function completes</p>
CONVERTDATA	<p>For hardware trigger types, the data is collected into a "circular" buffer. The CONVERTDATA option is used to align data within the buffer when the data acquisition is complete. This option is ignored for all 16-bit devices, and for 12-bit devices that store the data without an offset (refer to <a href="#">cbAInScan()</a>). Note that you can also call <a href="#">cbAConvertPretrigData()</a> to align data within the buffer when the data acquisition is complete.</p> <p>Use of CONVERTDATA is recommended unless one of the following two conditions exist: 1) On some devices, CONVERTDATA may not be specified if you are using the BACKGROUND option and DMA transfers. In this case, if data conversion is required, use <a href="#">cbAConvertData()</a> to re-align the data. 2) Some 12-bit boards store the data as a 12-bit A/D value and a 4-bit channel number. Using CONVERTDATA will strip out the channel number from the data. If you prefer to store the channel number as well as the data, call cbAConvertData() to retrieve the data and the channel number from the buffer after the data acquisition to the buffer is complete.</p> <p>The CONVERTDATA option is not required for software triggered types.</p>
EXTCLOCK	<p>This option is available only for boards that have separate inputs for external pacer and external trigger. See your hardware manual or board-specific information.</p>
EXTMEMORY	<p>Causes this function to send the data to a connected memory board via the DT-Connect interface rather than returning the data to the buffer. If you use this option to send the data to a MEGA-FIFO memory board, then you must use <a href="#">cbMemReadPretrig()</a> to later read the pre-trigger data from the memory board. If you use <a href="#">cbMemRead()</a>, the data will NOT be in the correct order.</p> <p>Every time this option is used, it overwrites any data already stored in the memory board. All data should be read from the board (with <a href="#">cbMemReadPretrig()</a>) before collecting any new data. When this option is used, the MemHandle argument is ignored. The MEGA-FIFO memory must be fully populated in order to use the cbAPretrig() function with the EXTMEMORY option.</p>
DTCONNECT	<p>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 Measurement Computing memory board plugged into the DT-Connect interface then you should use EXTMEMORY option rather than this option.</p>

## Important

For hardware trigger types, the buffer referenced by MemHandle must be big enough to hold at least TotalCount + 512 integers.

## cbAOutScan() function

Outputs values to a range of D/A channels. This function can be used for paced analog output on hardware that supports paced output. It can also be used to update all analog outputs at the same time when the SIMULTANEOUS option is used.

### Function Prototype

C/C++

```
int cbAOutScan(int BoardNum, int LowChan, int HighChan, long NumPoints, long *Rate, int Range, int MemHandle, int Options)
```

Visual Basic

```
Function cbAOutScan(ByVal BoardNum&, ByVal LowChan&, ByVal HighChan&, ByVal NumPoints&, Rate&, ByVal Range&, ByVal MemHandle&, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The board number associated with the board when it was installed with InstaCal. The specified board must have a D/A. BoardNum may be 0 to 99.

*LowChan*

First D/A channel of scan.

*HighChan*

Last D/A channel of scan.

**LowChan/HighChan:** The maximum allowable channel depends on which type of D/A board is being used.

*NumPoints*

Number of D/A values to output. Specifies the total number of D/A values that will be output. Most D/A boards do not support timed outputs. For these boards, set the count to the number of channels in the scan.

*Rate*

Sample rate in scans per second. For many 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 CIO-DAC04/12-HS, Rate should be set to the D/A output rate (in scans/sec). This argument returns the value of the actual rate set. This value may be different from the user specified rate due to pacer limitations.

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 (10 kHz) will result in the D/A converter rates of 10 kHz (one D/A per channel). The data transfer rate is 40,000 words per second; 4 channels \* 10,000 updates per scan.

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

*Range*

D/A range code. The output range of the D/A channel can be set to any of those supported by the board. If the D/A board does not have a programmable gain, this argument is ignored.

*MemHandle*

Handle for Windows buffer from which data will be output. This buffer must have been previously allocated with the [cbWinBufAlloc\(\)](#) function and data values loaded (perhaps using [cbWinArrayToBuf\(\)](#)).

*Options*

Bit fields that control various options. This field may contain any combination of non-contradictory choices from the values listed in the [Options argument values](#) section below.

### Returns

- [Error code](#) or 0 if no errors.
- Rate - Actual sampling rate used.

## Options argument values

ADCCLOCK	Paces the data output operation using the ADC clock.
ADCCLOCKTRIG	Triggers a data output operation when the ADC clock starts.
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 <a href="#">cbGetStatus()</a> with AOFUNCTION to check the status of background operation. Alternatively, some boards support <a href="#">cbEnableEvent()</a> for event notification of changes in status of BACKGROUND scans. Use <a href="#">cbStopBackground()</a> with AOFUNCTION to terminate background operations before they are completed. <a href="#">cbStopBackground()</a> should be executed after normal termination of all background functions in order to clear variables and flags.
CONTINUOUS	This option may only be used with boards which support interrupt, DMA or REP INSW transfer methods. This option puts the method in an endless loop.  Once it outputs the specified number (NumPoints) of D/A values, it resets to the start of the buffer and begins again. The only way to stop this operation is by calling <a href="#">cbStopBackground()</a> with AOFUNCTION. This option should only be used in combination with BACKGROUND so that your program can regain control.
EXTCLOCK	If this option is specified, conversions will be paced by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal (refer to board-specific information contained in the <i>Universal Library Users Guide</i> ).  When this option is used the Rate argument is ignored. The sampling rate is dependent on the clock signal. Options for the board default to transfer types that allow the maximum conversion rate to be attained unless otherwise specified.
EXTTRIGGER	If this option is specified the sampling will not begin until the trigger condition is met. On many boards, this trigger condition is programmable (refer to the <a href="#">cbSetTrigger()</a> function and board-specific information contained in the <i>Universal Library Users Guide</i> for details).
NONSTREAMEDIO	When this option is used, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample updates are transferred or downloaded to the device, the device is responsible for outputting the data. You can't make any changes to the output buffer once the output begins.  With NONSTREAMEDIO mode, you do not have to periodically feed output data through the program to the device for the data output to continue. However, the size of the buffer is limited.  NONSTREAMEDIO can only be used with the number of samples (Count) set equal to the size of the FIFO or less.
RETRIGMODE	Re-arms the trigger after a trigger event is performed. With this mode, the scan begins when a trigger event occurs. When the scan completes, the trigger is re-armed to generate the next the batch of data. You can specify the number of samples to generate for each trigger event (described below). The RETRIGMODE option can be used with the CONTINUOUS option to continue arming the trigger until <a href="#">cbStopBackground()</a> is called.  You can specify the number of samples to generate with each trigger event. This is the trigger count. Use the <a href="#">cbSetConfig()</a> ConfigItem option BIDACTRIGCOUNT to set the trigger count. If you specify a trigger count that is either zero or greater than the value of the NumPoints argument, the trigger count will be set to the value of NumPoints.
SCALEDATA	Gets scaled data, such as voltage, temperature, and so on, from the user buffer, and converts it to raw data. The user buffer should have been allocated with <a href="#">cbScaledWinBufAlloc()</a> .
SIMULTANEOUS	When this option is used (if the board supports it 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 updated. 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.

## 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 LowChan = 0 and 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  $4 \times 100,000 = 400,000$ . 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.

## cbATrig() function

Waits for a specified analog input channel to go above or below a specified value. cbATrig() continuously reads the specified channel and compares its value to TrigValue. Depending on whether TrigType is set to TRIGABOVE or TRIGBELOW, it waits for the first A/D sample that is above or below TrigValue. The first sample that meets the trigger criteria is returned to DataValue.

### Function Prototype

C/C++

```
int cbATrig(int BoardNum, int Channel, int TrigType, int TrigValue, int Range, unsigned short *DataValue)
```

Visual Basic

```
Function cbATrig(ByVal BoardNum&, ByVal Channel&, ByVal TrigType&, ByVal TrigValue%, ByVal Range&, DataValue%) As Long
```

### Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with the InstaCal. BoardNum may be 0 to 99. The specified board must have an A/D.

*Channel*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with 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 channels for differential inputs and 16 channels for single ended inputs.

*TrigType*

TRIGABOVE or TRIGBELOW. Specifies whether to wait 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. Must be in the range 0 –4,095 for 12-bit A/D boards, or 0–65,535 for 16-bit A/D boards. Refer to your BASIC manual for information on signed BASIC integer data types.

*Range*

Gain code. If the selected A/D board does not have a programmable gain feature, this argument is ignored. Otherwise, set to any range that is supported by the selected A/D board. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*DataValue*

Returns the value of the first A/D sample to meet the trigger criteria.

### Returns

- [Error code](#) or 0 if no errors
- DataValue - Value of the first A/D sample to match the trigger criteria.

### Notes

- Pressing Ctrl-C will not terminate the wait for an analog trigger that meets the specified condition. There are only two ways to terminate this call: satisfy the trigger condition or reset the computer.

### Caution!

Use caution when using this function in Windows programs. All active windows will lock on the screen until the trigger condition is satisfied. The keyboard and mouse activity will also lock until the trigger condition is satisfied.



## cbVIn() function

Reads an A/D input channel, and returns a voltage value. If the specified A/D board has programmable gain, then this function sets the gain to the specified range. The voltage value is returned to DataValue.

### Function Prototype

C/C++

```
int cbVIn(int BoardNum, int Channel, int Range, float *DataValue, int Options);
```

Visual Basic

```
Function cbVIn(ByVal BoardNum&, ByVal Channel&, ByVal Range&, DataValue!, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The board number associated with the board used to collect the data when it was installed with InstaCal. BoardNum may be 0 to 99. The specified board must have an A/D.

*Channel*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single-ended and differential inputs, the maximum allowable channel number also depends on how the board is configured.

*Range*

A/D range code. If the board has a programmable gain, it will be set according to this argument value. Keep in mind that some A/D boards have a programmable gain feature, and others set the gain via switches on the board. In either case, the range that the board is configured for must be passed to this function. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*DataValue*

A pointer or reference to the data value.

*Options*

Reserved for future use.

### Returns

- [Error code](#) or 0 if no errors
- DataValue - Returns the value in volts of the A/D sample.

### Options argument values

*Default*

Reserved for future use.

## cbVIn32() function

Reads an A/D input channel, and returns a voltage value. This function is similar to [cbVIn\(\)](#), but returns a double precision float value instead of a single precision float value. If the specified A/D board has programmable gain, then this function sets the gain to the specified range. The voltage value is returned to DataValue.

### Function Prototype

C/C++

```
int cbVIn32(int BoardNum, int Chan, int Gain, double* DataValue, int Options);
```

Visual Basic

```
Function cbVIn32(ByVal BoardNum&, ByVal Chan&, ByVal Gain&, DataValue#, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The board number associated with the board used to collect the data when it was installed with InstaCal. BoardNum may be 0 to 99. The specified board must have an A/D.

*Chan*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single-ended and differential inputs, the maximum allowable channel number also depends on how the board is configured.

*Gain*

A/D range code. If the board has a programmable gain, it will be set according to this argument value. Keep in mind that some A/D boards have a programmable gain feature, and others set the gain via switches on the board. In either case, the range that the board is configured for must be passed to this function. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*DataValue*

A pointer or reference to the data value.

*Options*

Board-specific operations to apply to the operation.

### Returns

- [Error code](#) or 0 if no errors
- DataValue - Returns the value in volts of the A/D sample.

### Options argument values

*Default*

Reserved for future use.

# cbVOut() function

Sets the value of a D/A channel. This function cannot be used for current output.

## Function Prototype

C/C++

```
int cbVOut(int BoardNum, int Channel, int Range, float DataValue, int Options);
```

Visual Basic

```
Function cbVOut (ByVal BoardNum&, ByVal Channel&, ByVal Range&, ByVal DataValue!, ByVal Options&) As Long
```

## Arguments

*BoardNum*

The board number associated with the board used to collect the data when it was installed with InstaCal. BoardNum may be 0 to 99. The specified board must have a D/A.

*Channel*

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

*Range*

The D/A range code. If the device has a programmable gain, it is set according to this argument value. If the range specified isn't supported, the function return a BADRANGE error.

If the gain is fixed or manually selectable, make sure that this argument matches the gain configured for the device. If it doesn't, the output voltage will not match the voltage specified in the DataValue argument.

*DataValue*

The voltage value to be written.

*Options*

Reserved for future use.

## Returns

- [Error code](#) or 0 if no errors

### options parameter values

Default	Reserved for future use.
---------	--------------------------

## cbGetConfig() function

Returns a configuration option for a board. The configuration information for all boards is stored in the CB.CFG file. This information is loaded from CB.CFG by all programs that use the library. You can change the current configuration within a running program with the [cbSetConfig\(\)](#) function. The `cbGetConfig()` function returns the current configuration information.

### Function Prototype

C/C++

```
int cbGetConfig(int InfoType, int BoardNum, int DevNum, int ConfigItem, int *ConfigVal)
```

Visual Basic

```
Function cbGetConfig(ByVal InfoType&, ByVal BoardNum&, ByVal DevNum&, ByVal ConfigItem&, ConfigVal&) As Long
```

### Arguments

*InfoType*

The configuration information for each board is grouped into different categories. This argument specifies which category you want. Set it to one of the constants listed in the "[InfoType argument values](#)" below.

*BoardNum*

Refers to the board number associated with a board when it was installed with InstaCal. BoardNum may be 0 to 99.

*DevNum*

Selects a particular device. If InfoType = DIGITALINFO, then DevNum specifies which of the board's digital devices you want information on. If InfoType = COUNTERINFO, then DevNum specifies which of the board's counter devices you want information from.

*ConfigItem*

Specifies which configuration item you wish to retrieve. Set it in conjunction with the InfoType argument using one of the constants listed in the "[ConfigItem argument values](#)" below.

*ConfigVal*

The specified configuration item is returned to this variable.

### Returns

- [Error code](#) or 0 if no errors.
- ConfigVal - returns the value of the specified configuration item here.

### InfoType argument values

InfoType	Description
GLOBALINFO	Information about the configuration file.
BOARDINFO	General information about a board.
DIGITALINFO	Information about a digital device.
COUNTERINFO	Information about a counter device.
EXPANSIONINFO	Information about an expansion device.
MISCINFO	One of the miscellaneous options for the board.

## ConfigItem argument values

Valid ConfigItem constant settings for each InfoType constant are as follows:

InfoType	ConfigItem	Description
GlobalInfo	GIVERSION	CB.CFG file format - used by the library to determine compatibility.
	GINUMBOARDS	Maximum number of boards that can be installed.
	GINUMEXPBOARDS	Maximum number of expansion boards that can be installed.
BOARDINFO	BIADCSETTLETIME	ADC settling time
	BIBASEADR	Base address of the device
	BIBOARDTYPE	Returns a unique number in the range of 0 to 8000 Hex describing the board type installed.
	BICINUMDEVS	Number of counter devices
	BICLOCK	Clock frequency in megahertz (MHz) (40, 10, 8, 6, 5, 4, 3, 2, 1) or 0 for not supported.
	BIDACSTARTUP	Returns the setting of a DAC board's configuration register STARTUP bit. Refer to the <a href="#">Notes</a> section below for more information.
	BIDACUPDATEMODE	Returns the setting of the update mode for a digital-to-analog converter (DAC). Refer to the <a href="#">Notes</a> section below for more information.
	BIDINUMDEVS	Number of digital devices
	BIDMACHAN	DMA channel — 0, 1 or 3
	BIDTBOARD	Board number of the connected DT board
	BIFACTORYID	The factory serial number of a USB device, or the MAC address of a WEB device.
	BIINTLEVEL	Interrupt level. 0 for none, or 1 - 15
	BINETIOTIMEOUT	The amount of time (in milliseconds) to wait for a WEB device to acknowledge a command or query sent to the device over a network connection. If no acknowledgement is received in this time a timeout occurs.
	BINUMADCHANS	Number of A/D channels
	BINUMDACHANS	Number of D/A channels
	BINUMIOPORTS	Number of I/O ports used by the device
	BINUMTEMPCHANS	Number of temperature channels
	BIPANID	Sets the Personal Area Network (PAN) identifier for a USB device that supports wireless communication.
	BIRANGE	Selected voltage range. For switch selectable gains only.  If the selected A/D board does not have a programmable gain feature, this argument returns the range as defined by the installed InstaCal settings. If InstaCal and the board are installed correctly, the returned range will correspond to the input range as set via the switches on the board. Refer to board specific information for a list of the <a href="#">A/D ranges</a> supported by each board.
	BIRFCHANNEL	Sets the RF channel number used to transmit/receive data by a USB device that supports wireless communication.
	BIRSS	Returns the received signal strength in dBm of a remote device.
	BISERIALNUM	Returns the serial number assigned by a user to a USB device in InstaCal. This ConfigItem does not return the factory serial number.
	BIWAITSTATE	Setting of the Wait State jumper. 1 = enabled, 0 = disabled
	BIUSESEXP	Supports expansion boards. TRUE/FALSE value is returned.
DIGITALINFO	DIBASEADR	Base address (16-bit library only)
	DIDEVTYPE	Device Type - AUXPORT, FIRSTPORTA etc.
	DICONFIG	Current configuration INPUT or OUTPUT.
	DINUMBITS	Number of bits in the port.
	DICURVAL	Current value of outputs.
	DIINMASK	Returns the bit configuration of the specified port. Any bits that return a value of 1 are configured for input. Refer to the <a href="#">Notes</a> section below for more information.
	DIOUTMASK	Returns the bit configuration of the specified port. Any bits that return a value of 1 are configured for output. Refer to the <a href="#">Notes</a> section below for more information.

COUNTERINFO	CICTRNUM	The counter number referred to by devnum.
	CICTRTYPE	The counter type, where: 1 = 8254, 2 = 9513, 3 = 8536, 4 = 7266, 5 = event counter, 6 = scan counter, 7 = timer counter, 8 = quadrature counter, and 9 = pulse counter.
EXPANSIONINFO	XIBOARDTYPE	Board type (refer to the " <a href="#">Measurement Computing Device IDs</a> " topic in the <i>Universal Library User's Guide</i> )
	XIMUXADCHAN1	First A/D channel that the EXP board is connected to
	XIMUXADCHAN2	Second A/D channel that the EXP board is connected to
	XIRANGE1	Range (gain) of the low 16 channels
	XIRANGE2	Range (gain) of the high 16 channels
	XICJCCHAN	A/D channel that the CJC is connected to
	XITHERMTYPE	Sensor type. Use one of the sensor types listed below: <ul style="list-style-type: none"> <li>■ J = 1</li> <li>■ K = 2</li> <li>■ T = 3</li> <li>■ E = 4</li> <li>■ R = 5</li> <li>■ S = 6</li> <li>■ B = 7</li> <li>■ Platinum .00392 = 257</li> <li>■ Platinum .00391 = 258</li> <li>■ Platinum .00385 = 259</li> <li>■ Copper .00427 = 260</li> <li>■ Nickel/Iron .00581 = 261</li> <li>■ Nickel/Iron .00527 = 262</li> </ul>
	XINUMEXPCHANS	Number of channels on the expansion board
	XIPARENTBOARD	Board number of the parent A/D board

## Notes

- Use the DIINMASK and DIOUTMASK options to determine if an AUXPORT is configurable. Execute cbGetConfig() twice to the same port – once using DIINMASK and once using DIOUTMASK. If both of the ConfigValue arguments returned have input and output bits that overlap, the port is not configurable.

You can determine overlapping bits by *Anding* both arguments.

Example: for a device with seven bits of digital I/O (four outputs and three inputs), the ConfigValue returned by DIINMASK is always 7 (0000 0111), while the ConfigValue argument returned by DIOUTMASK is always 15 (0000 1111). When you *And* both ConfigValue arguments together, you get a non-zero number (7). Any non-zero number indicates that input and output bits overlap for the specified port, and the port is a non-configurable AUXPORT.

- Use the BIDACSTARTUP option to determine whether a board's DAC values before the last power down are stored.

With ConfigItem set to BIDACSTARTUP, Configval returns 0 when the startup bit is *disabled*. Current DAC settings are stored as startup values.

ConfigVal returns 1 when the startup bit is *enabled*. The last DAC values are stored as startup values.

Refer to the cbSetConfig() [Notes](#) section for information about how to store the current or last DAC values as start-up values.

- Use the BIDACUPDATEMODE option to check the update mode for a DAC board.

With ConfigItem set to BIDACUPDATEMODE, ConfigVal returns 0 when the DAC update mode is *immediate*. Values written with [cbAOut\(\)](#) are automatically output by the DAC channels.

ConfigVal returns 1 when the DAC update mode is set to *on command*. Values written with [cbAOut\(\)](#) are not output by the DAC channels until a [cbSetConfig\(\)](#) call is made with its ConfigItem argument set to BIDACUPDTECMD.

# cbGetConfigString() function

Retrieves configuration or device information as a null-terminated string.

## Function Prototype

C/C++

```
int cbGetConfigString(int InfoType, int BoardNum, int ItemIndex, int ConfigItem, char *ConfigVal, int* maxConfigLen)
```

Visual Basic

```
Function cbGetConfigString(ByVal InfoType, ByVal BoardNum&, ByVal ItemIndex&, ByVal ConfigItem&, ByVal ConfigVal$, ByRef maxConfigLen&) As Long
```

## Arguments

*InfoType*

The configuration information for each board is grouped into different categories. This argument specifies which category you want. Always set this argument to BOARDINFO.

*BoardNum*

Refers to the board number associated with a board when it was installed with InstaCal. BoardNum may be 0 to 99.

*ItemIndex*

The location in the device memory (specified by ConfigItem) at which to start reading.

*ConfigItem*

Specifies the type of information (or memory area) to read from the device. Set it to one of the constants listed in the ["ConfigItem argument values"](#) section below.

*ConfigVal*

Pointer to a user-allocated buffer where the configuration string is copied.

*maxConfigLen*

Pointer to the value holding the maximum number of bytes to be read from the device into ConfigVal.

## Returns

- [Error code](#) or 0 if no errors.
- maxConfigLen - The number of bytes read from the device into ConfigVal.
- ConfigVal - The string read from the device.

## ConfigItem argument values

ConfigItem	Description
BIDEVNOTES	Reads up to maxConfigLen characters/bytes from the device notes memory, starting at the location defined by ItemIndex. Currently supported only for WLS Series devices.
BIFACTORYID	Reads the factory serial number of a USB device, or the MAC address of a WEB device.
BINODEID	Reads up to maxConfigLen character/bytes from the string identifier memory. Note that ItemIndex is not used for this ConfigItem.

## cbGetSignal() function

Retrieves the configured Auxiliary or DAQ Sync connection and polarity for the specified timing and control signal.

This function is intended for advanced users. Except for the SYNC\_CLK input, you can easily view the settings for the timing and control signals using InstaCal.

Note: This function is not supported by all board types.

### Function Prototype

C/C++

```
int cbGetSignal(int BoardNum, int Direction, int Signal, int Index, int* Connection, int* Polarity)
```

Visual Basic

```
Function cbGetSignal(ByVal BoardNum&, ByVal Direction&, ByVal Signal&, ByVal Index&, ByRef Connection, ByRef Polarity) As Long
```

### Arguments

*BoardNum*

Refers to the board number associated with the A/D board when it was installed with InstaCal. The specified board must have configurable signal inputs and outputs.

*Direction*

Specifies whether retrieving the source (SIGNAL\_IN) or destination (SIGNAL\_OUT) of the specified signal.

*Signal*

Signal type whose connection is to be retrieved. See [cbSelectSignal](#) for valid signal types.

*Index*

Used to indicate which connection to reference when there is more than one connection associated with the output Signal type. When querying output signals, increment this value until BADINDEX is returned or 0 is returned via the Connection parameter to determine all the output Connections for the specified output Signal. The first Connection is indexed by 0.

For input signals (Direction=SIGNAL\_IN), this should always be set to 0.

*Connection*

The specified connection is returned through this variable. This is set to 0 if no connection is associated with the Signal, or if the Index is set to an invalid value.

*Polarity*

Holds the polarity for the associated Signal and Connection.

For output Signals assigned an AUXOUT Connection, the return value is either INVERTED or NONINVERTED.

For Signal settings of ADC\_CONVERT, DAC\_UPDATE, ADC\_TB\_SRC and DAC\_TB\_SRC input signals, either POSITIVEEDGE or NEGATIVEEDGE is returned.

All other signals return 0.

### Returns

- [Error code](#) or 0 if no errors.

### Notes

Timing and control configuration information can be viewed and edited inside InstaCal. Do the following:

1. Run InstaCal.
2. Click on the board and press the **Configure...** button or menu item. If the board supports DAQ Sync and Auxiliary Input/Output signal connections, a button labeled **Advanced Timing & Control Configuration** displays.
3. Press this button to open a display for viewing and modifying the above timing and control signals.



## cbSelectSignal() function

Configures timing and control signals to use specific Auxiliary or DAQ Sync connections as a source or destination.

This function is intended for advanced users. Except for the SYNC\_CLK input, you can easily configure all the timing and control signals using InstaCal.

Note: This function is not supported by all board types. Please refer to board specific information for details.

### Function Prototype

C/C++

```
int cbSelectSignal(int BoardNum, int Direction, int Signal, int Connection, int Polarity);
```

Visual Basic

```
Function cbSelectSignal(ByVal BoardNum&, ByVal Direction&, ByVal Signal&, ByVal Connection&, ByVal Polarity&) as Long
```

### Arguments

*BoardNum*

Refers to the board number associated with the A/D board when it was installed. The specified board must have configurable signal inputs and outputs.

*Direction*

Direction of the specified signal type to be assigned a connector pin. For most signal types, this should be either SIGNAL\_IN or SIGNAL\_OUT. For the SYNC\_CLK, ADC\_TB\_SRC and DAC\_TB\_SRC signals, the external source can also be disabled by specifying DISABLED(=0) such that it is neither input nor output. Set it in conjunction with the Signal, Connection, and Polarity arguments using the information listed in the [Direction argument values](#) section below.

*Signal*

Signal type to be associated with a connector pin. Set it to one of the constants listed in the "[Signal argument values](#)" section below.

*Connection*

Designates the connector pin to associate the signal type and direction. Since individual pin selection is not allowed for the DAQ-Sync connectors, all DAQ-Sync pin connections are referred to as DS\_CONNECTOR. The AUXIN and AUXOUT settings match their corresponding hardware pin names.

*Polarity*

ADC\_TB\_SRC and DAC\_TB\_SRC input signals (SIGNAL\_IN) can be set for either rising edge (POSITIVEEDGE) or falling edge (NEGATIVEEDGE) signals. The AUXOUT connections can be set to INVERTED or NONINVERTED from their internal polarity.

### Returns

- [Error code](#) or 0 if no errors.

## Signal argument values

Signal	Connection
ADC_CONVERT	A/D conversion pulse or clock.
ADC_GATE	External gate for A/D conversions.
ADC_SCANCLK	A/D channel scan signal.
ADC_SCAN_STOP	A/D scan completion signal.
ADC_SSH	A/D simultaneous sample and hold signal.
ADC_STARTSCAN	Start of the A/D channel-scan sequence signal.
ADC_START_TRIG	A/D scan start trigger.
ADC_STOP_TRIG	A/D stop- or pre- trigger.
ADC_TB_SRC	A/D pacer timebase source.
CTR1_CLK	CTR1 clock source.
CTR2_CLK	CTR2 clock source.
DAC_START_TRIG	D/A start trigger.
DAC_TB_SRC	D/A pacer timebase source.
DAC_UPDATE	D/A update signal.
DGND	Digital ground.
SYNC_CLK	STC timebase signal.

## Direction argument values

Valid input (**Direction=SIGNA\_IN**) connection pin and polarity settings include:

Signal	Connection	Polarity
ADC_CONVERT	AUXIN0..AUXIN5 DS_CONNECTOR	POSITIVEEDGE or NEGATIVEEDGE
ADC_GATE	AUXIN0..AUXIN5	See <a href="#">cbSetTrigger()</a>
ADC_START_TRIG	AUXIN0..AUXIN5 DS_CONNECTOR	See <a href="#">cbSetTrigger()</a>
ADC_STOP_TRIG	AUXIN0..AUXIN5 DS_CONNECTOR	See <a href="#">cbSetTrigger()</a>
ADC_TB_SRC	AUXIN0..AUXIN5	POSITIVEEDGE or NEGATIVEEDGE
DAC_START_TRIG	AUXIN0..AUXIN5 DS_CONNECTOR	Not assigned here.
DAC_TB_SRC	AUXIN0..AUXIN5	POSITIVEEDGE or NEGATIVEEDGE
DAC_UPDATE	AUXIN0..AUXIN5 DS_CONNECTOR	POSITIVEEDGE or NEGATIVEEDGE
SYNC_CLK	DS_CONNECTOR	Not assigned here.

Valid output (**Direction=SIGNA\_OUT**) connection pin and polarity settings include:

Signal	Connection	Polarity
ADC_CONVERT	AUXIN0..AUXIN2 DS_CONNECTOR	INVERTED* or NONINVERTED
ADC_SCANCLK	AUXIN0..AUXIN2	
ADC_SCAN_STOP	AUXIN0..AUXIN2	
ADC_SSH	AUXIN0..AUXIN2	
ADC_STARTSCAN	AUXIN0..AUXIN2	
ADC_START_TRIG	AUXIN0..AUXIN2 DS_CONNECTOR	
ADC_STOP_TRIG	AUXIN0..AUXIN2 DS_CONNECTOR	
CTR1_CLK	AUXIN0..AUXIN2	
CTR2_CLK	AUXIN0..AUXIN2	
DAC_START_TRIG	AUXIN0..AUXIN2 DS_CONNECTOR	
DAC_UPDATE	AUXIN0..AUXIN2 DS_CONNECTOR	
DGND	AUXIN0..AUXIN2	Not assigned here.
SYNC_CLK	DS_CONNECTOR	

\* INVERTED is only valid for Auxiliary Output (AUXOUT) connections.

Valid disabled settings (Direction=DISABLED):

Signal	Connection	Polarity
<a href="#">ADC_TB_SRC</a>	Not assigned here.	Not assigned here.
<a href="#">DAC_TB_SRC</a>		
SYNC_CLK		

## Notes

- You can view and edit the above timing and control configuration information from InstaCal. Open InstaCal, click on the board, and press the "Configure..." button or menu item. If the board supports DAQ Sync and Auxiliary Input/Output signal connections, a button labeled "Advanced Timing & Control Configuration" displays. Press that button to open a display for viewing and modifying the above timing and control signals.
- Except for the ADC\_TB\_SRC, DAC\_TB\_SRC and SYNC\_CLK signals, selecting an input signal connection does not necessarily activate it. However, assigning an output signal to a connection does activate the signal upon performing the respective operation. For instance, when running an EXTCLOCK cbAInScan(), ADC\_CONVERT SIGNAL\_IN selects the connection to use as an external clock to pace the A/D conversions; if cbAInScan() is run without setting the EXTCLOCK option, however, the selected connection is not activated and the signal at that connection is ignored. In both cases, the ADC\_CONVERT signal is output via the connection(s) selected for the ADC\_CONVERT SIGNAL\_OUT. Since there are no scan options for enabling the Timebase Source and the SYNC\_CLK, selecting an input for the A/D or D/A Timebase Source, or SYNC\_CLK does activate the input source for the next respective operations.
- Multiple input signals can be mapped to the same AUXIN<sub>n</sub> connection by successive calls to cbSelectSignal; however, only one connection can be mapped to each input signal. If another connection had already been assigned to an input signal, the former selection is de-assigned and the new connection is assigned.
- Only one output signal can be mapped to the same AUXOUT<sub>n</sub> connection; however, multiple connections can be mapped to the same output signal by successive calls to cbSelectSignal. If an output signal had already been assigned to a connection, then the former output signal is de-assigned and the new output signal is assigned to the connection. Note that there are at most MAX\_CONNECTIONS(=4) connections that can be assigned to each output signal.
- When selecting DS\_CONNECTOR for a signal, only one direction per signal type can be defined at a given time. Attempting to assign both directions of a signal to the DS\_CONNECTOR results in only the latest selection being applied. If the signal type had formerly been assigned an input direction from the DS\_CONNECTOR, assigning the output direction for that signal type results in the input signal being reassigned to its default connection:

Default input signal connections	
Input signal	Default connection
ADC_CONVERT	AUXIN0
ADC_GATE	AUXIN5
ADC_START_TRIG	AUXIN1
ADC_STOP_TRIG	AUXIN2
DAC_UPDATE	AUXIN3
DAC_START_TRIG	AUXIN3

- **ADC\_TB\_SRC** and **DAC\_TB\_SRC** are intended to synchronize the timebase of the analog input and output pacers across two or more boards. Internal calculations of sampling and update rates assume that the external timebase has the same frequency as its internal clock. Adjust sample rates to compensate for differences in clock frequencies.

For instance, if the external timebase has a frequency of 10 MHz on a board that has a internal clock frequency of 40 MHz, the scan function samples or updates at a rate of about 1/4 the rate entered. However, while compensating for differences in external timebase and internal clock frequency, if the rate entered results in an invalid pacer count, the function returns a BADRATE error.

# cbSetConfig() function

Sets a configuration option for a board. The configuration information for all boards is stored in the CB.CFG file. All programs that use the library read this file. You can use this function to override the configuration information stored in the CB.CFG file.

## Function prototype

C/C++

```
int cbSetConfig(int InfoType, int BoardNum, int DevNum, int ConfigItem, int ConfigVal)
```

Visual Basic

```
Function cbSetConfig(ByVal InfoType&, ByVal BoardNum&, ByVal DevNum&, ByVal ConfigItem&, ByVal ConfigVal&) As Long
```

## Arguments

*InfoType*

The configuration information for each board is grouped into different categories. InfoType specifies which category you want. Set it to one of the constants listed in the [InfoType argument values](#) section below.

*BoardNum*

Refers to the board number associated with a board when it was installed. BoardNum may be 0 to 99.

*DevNum*

Selects a particular device. If InfoType = DIGITALINFO, then DevNum specifies which of the board's digital devices you want to set information on. If InfoType = COUNTERINFO then DevNum specifies which of the board's counter devices you want to set information on.

*ConfigItem*

Specifies which configuration item you wish to set. Set it in conjunction with the InfoType argument using the table under [ConfigItem argument values](#) section below.

*ConfigVal*

The value to set the specified configuration item to.

## Returns

- [Error code](#) or 0 if no errors.

## InfoType argument values

InfoType	Description
BOARDINFO	General information about a board.
DIGITALINFO	Information about a digital device.
COUNTERINFO	Information about a counter device.
EXPANSIONINFO	Information about an expansion device.
MISCINFO	One of the miscellaneous options for the board.

## ConfigItem argument values

InfoType	ConfigItem	Description
BoardInfo	BIADTRIGCOUNT	ADC trigger count. For use with the cbAInScan()/AInScan() <a href="#">RETRIGMODE</a> option to set up repetitive trigger events.
	BIBASEADR	Base address of the board
	BICALOUTPUT	Sets the voltage for the CAL pin on supported USB devices
	BICLOCK	Clock frequency in megahertz (MHz) (1, 4, 6 or 10)
	BIDACSTARTUP	Sets the board's configuration register STARTUP bit to 0 or 1 to enable/disable the storing of digital-to-analog converter (DAC) startup values. Each time the board is powered up, the stored values are written to the DACs. Refer to the <a href="#">"Notes"</a> section below for more information.
	BIDACTRIGCOUNT	DAC trigger count. For use with the cbAOutScan()/AOutScan() <a href="#">RETRIGMODE</a> option to set up repetitive trigger events.
	BIDACUPDATECMD	Updates all analog output channels.  When ConfigItem is set to BIDACUPDATECMD, the DevNum and ConfigVal arguments are not used and can be set to 0. Refer to the <a href="#">"Notes"</a> section below for more information.
	BIDACUPDATESMODE	Sets the update mode for a digital-to-analog converter (DAC). Use this setting in conjunction with one of these ConfigVal settings: <ul style="list-style-type: none"> <li>■ UPDATEIMMEDIATE</li> <li>■ UPDATEONCOMMAND</li> </ul> Refer to the <a href="#">"Notes"</a> section below for more information.
	BIDACSETTLTIME	ADC settling time
	BIDIDEBOUNGESTATE	The state of the digital inputs when debounce timing is set
	BIDIDEBOUNCETIME	Sets the debounce time of digital inputs
	BIDMACHAN	DMA channel
	BIINTLEVEL	Interrupt level
	BIHIDELOGINDLG	Enables or disables the Device Login dialog. Set to a nonzero value to disable the dialog. When disabled, the <a href="#">cbDeviceLogin()</a> function must be used to log in to a device session.
	BINETIOTIMEOUT	Sets the amount of time (in milliseconds) to wait for a WEB device to acknowledge a command or query sent to the device over a network connection. If no acknowledgement is received in this time a timeout occurs.
	BINUMADCHANS	Number of A/D channels
	BIPANID	Sets the Personal Area Network (PAN) identifier of a USB device that supports wireless communication.
	BIRANGE	<a href="#">Selected voltage range</a>
	BIRFCHANNEL	Sets the RF channel number used to transmit/receive data by a USB device that supports wireless communication.
	BIRSS	The received signal strength in dBm of a remote device.
	BISRCADPACER	Outputs the A/D pacer signal to the SYNC pin on supported USB devices.
	BIWAITSTATE	Setting of the Wait State jumper
EXPANSIONINFO	XIMUXADCHAN1	The first A/D channel that the board is connected to.
	XIMUXADCHAN2	The second A/D channel that the board is connected to.
	XIRANGE1	Range (gain) of the low 16 channels.
	XIRANGE2	Range (gain) of the high 16 channels.
	XICJCCHAN	A/D channel that the CJC is connected to.
	XITHERMTYPE	Thermocouple type

## Notes

- Use the ConfigItem option BIDACSTARTUP to store either the current DAC values or the DAC values before the board was last powered down.

To store the current DAC values as start-up values, call `cbSetConfig()` with a value of 1 for the BIDACSTARTUP value. Then, call `cbAOut()` or `cbAOutScan()` for each channel (), and call `cbSetConfig()` again with a value of 0 for the BIDACSTARTUP value.

### Example:

```
cbSetConfig(BOARDINFO, boardNumber, 0, BIDACSTARTUP, 1);
    for (int i =1; i <8; i++)
    {
        cbAOut(boardNumber, i, BIP5VOLTS, DACValue[i]);
    }
cbSetConfig(BOARDINFO, boardNumber, 0, BIDACSTARTUP, 0);
```

To store the DAC's last settings, call `cbSetConfig()` with a BIDACSTARTUP value of 1. Leave this bit turned on until the application exits. The next time the board is powered up, it restores the values last written to the DACs.

- Use the BIDACUPDATEMODE option to set the update mode for a DAC board.

With ConfigItem set to BIDACUPDATEMODE and ConfigVal set to 0, the DAC update mode is *immediate*. Values written with `cbAOut()` or `cbAOutScan()` are automatically output by the DAC channels.

With ConfigItem set to BIDACUPDATEMODE and ConfigVal set to 1, the DAC update mode is *on command*. Values written with `cbAOut()` or `cbAOutScan()` are not output by the DAC channels until another `cbSetConfig()` call is made with ConfigItem set to BIDACUPDATECMD.

# cbSetConfigString() function

Sets the configuration or device information as a null-terminated string.

## Function Prototype

C/C++:

```
int cbSetConfigString(int InfoType, int BoardNum, int ItemIndex, int ConfigItem, char *ConfigVal, int* maxConfigLen)
```

Visual Basic:

```
Function cbSetConfigString(ByVal InfoType, ByVal BoardNum&, ByVal ItemIndex&, ByVal ConfigItem&, ByVal ConfigVal$, ByRef maxConfigLen&) As Long
```

## Arguments

*InfoType*

The configuration information for each board is grouped into different categories. This argument specifies which category you want. Always set this argument to BOARDINFO.

*BoardNum*

Refers to the board number associated with a board when it was installed. BoardNum may be 0 to 99.

*ItemIndex*

The location in the device memory (specified by ConfigItem) at which to start writing.

*ConfigItem*

The type of information (or memory area) to write to the device. Set it to one of the constants listed in the "[ConfigItem argument values](#)" section below.

*ConfigVal*

Pointer to the user-allocated buffer containing the string to copy to the device's memory.

*maxConfigLen*

Pointer to the value specifying the number of bytes to be written to the device from ConfigVal.

## Returns

- [Error code](#) or 0 if no errors.
- maxConfigLen - The number of bytes written to the device.

## ConfigItem argument values

ConfigItem	Description
BIDEVNOTES	Writes up to maxConfigLen characters/bytes from the ConfigVal buffer to the device notes memory, beginning at the location defined by ItemIndex. Currently supported only for WLS Series devices.
BINODEID	Writes up to maxConfigLen characters/bytes from the ConfigVal buffer to the string identifier memory on the device. Note that ItemIndex is not used for this ConfigItem.

## cbSetTrigger() function

Selects the trigger source and sets up its parameters. This trigger is used to initiate analog to digital conversions using the following Universal Library functions:

- [cbAInScan\(\)](#), if the EXTTRIGGER option is selected.
- [cbDInScan\(\)](#), if the EXTTRIGGER option is selected.
- [cbCInScan\(\)](#), if the EXTTRIGGER option is selected.
- [cbAPretrig\(\)](#)
- [cbFilePretrig\(\)](#)

## Function prototype

C/C++

```
int cbSetTrigger(int BoardNum, int TrigType, unsigned short LowThreshold, unsigned short HighThreshold);
```

Visual Basic

```
Function cbSetTrigger(ByVal BoardNum&, ByVal TrigType&, ByVal LowThreshold%, ByVal HighThreshold%) As Long
```

## Arguments

*BoardNum*

Specifies the board number associated with the board when it was installed with the configuration program. The board must have the software selectable triggering source and/or options. BoardNum may be 0 to 99.

*TrigType*

Specifies the type of triggering based on the external trigger source. Set it to one of the constants in the [TrigType argument values](#) section below.

*LowThreshold*

Selects the low threshold used when the trigger input is analog. The range depends upon the resolution of the trigger circuitry. Must be 0 to 255 for 8-bit trigger circuits, 0 to 4,095 for 12-bit trigger circuits, and 0 to 65,535 for 16-bit trigger circuits. Refer to the [Notes](#) section below.

*HighThreshold*

Selects the high threshold used when the trigger input is analog. The range depends upon the resolution of the trigger circuitry. Must be 0 to 255 for 8-bit trigger circuits, 0 to 4,095 for 12-bit trigger circuits, and 0 to 65,535 for 16-bit trigger circuits. Refer to the [Notes](#) section below.

## Returns

- [Error code](#) or 0 if no errors.



## TrigType argument values

Trigger source	Type	Explanation
Analog	GATE_NEG_HYS	AD conversions are enabled when the external analog trigger input is more positive than HighThreshold. AD conversions are disabled when the external analog trigger input is more negative than LowThreshold. Hysteresis is the level between LowThreshold and HighThreshold.
	GATE_POS_HYS	AD conversions are enabled when the external analog trigger input is more negative than LowThreshold. AD conversions are disabled when the external analog trigger input is more positive than HighThreshold. Hysteresis is the level between LowThreshold and HighThreshold.
	GATE_ABOVE	AD conversions are enabled as long as the external analog trigger input is more positive than HighThreshold.
	GATE_BELOW	AD conversions are enabled as long as the external analog trigger input is more negative than LowThreshold.
	GATE_IN_WINDOW	AD conversions are enabled as long as the external analog trigger is inside the region defined by LowThreshold and HighThreshold.
	GATE_OUT_WINDOW	AD conversions are enabled as long as the external analog trigger is outside the region defined by LowThreshold and HighThreshold.
	TRIG_ABOVE	AD conversions are enabled when the external analog trigger input transitions from below HighThreshold to above. Once conversions are enabled, the external trigger is ignored.
	TRIG_BELOW	AD conversions are enabled when the external analog trigger input transitions from above LowThreshold to below. Once conversions are enabled, the external trigger is ignored.
Digital	GATE_HIGH	AD conversions are enabled as long as the external digital trigger input is 5V (logic HIGH or '1').
	GATE_LOW	AD conversions are enabled as long as the external digital trigger input is 0V (logic LOW or '0').
	TRIG_HIGH	AD conversions are enabled when the external digital trigger is 5V (logic HIGH or '1'). Once conversions are enabled, the external trigger is ignored.
	TRIG_LOW	AD conversions are enabled when the external digital trigger is 0V (logic LOW or '0'). Once conversions are enabled, the external trigger is ignored.
	TRIG_POS_EDGE	AD conversions are enabled when the external digital trigger makes a transition from 0V to 5V (logic LOW to HIGH). Once conversions are enabled, the external trigger is ignored.
	TRIG_NEG_EDGE	AD conversions are enabled when the external digital trigger makes a transition from 5V to 0V (logic HIGH to LOW). Once conversions are enabled, the external trigger is ignored.

## Notes

- The threshold value must be within the range of the analog trigger circuit associated with the board. Refer to the board-specific information. For example, on the PCI-DAS 1602/16 the analog trigger circuit handles  $\pm 10\text{V}$ . A value of 0 corresponds to  $-10\text{V}$ , whereas a value of 65,535 corresponds to  $+10\text{V}$ .

Since Visual Basic does not support unsigned integer types, the thresholds range from  $-32,768$  to  $32,767$  for 16 bit boards, instead of from 0 to 65,535. In this case, the unsigned value of 65,535 corresponds to a value of  $-1$ , 65,534 corresponds to  $-2$ , ...,  $32,768$  corresponds to  $-32,768$ .

- For most boards that support analog triggering, you can pass the required trigger voltage level and the appropriate Range to `cbFromEngUnits()` to calculate the HighThreshold and LowThreshold values.

For some boards, you must **manually calculate the threshold** by first calculating the least significant bit (LSB) for a particular range for the trigger resolution of your hardware. You then use the LSB to find the threshold in counts based on an analog voltage trigger threshold. Refer to the following procedure for details. For board-specific information, refer to your hardware in the "Analog Input Boards" section of the *Universal Library User's Guide*.

### Manually calculating the threshold

To calculate the threshold, do the following:

1. Calculate the LSB by dividing the full scale range (FSR) by  $2^{\text{resolution}}$ . FSR is the entire span from  $-FS$  to  $+FS$  of your hardware for a particular range. For example, the full scale range of  $\pm 10\text{ V}$  is  $20\text{ V}$ .
2. Calculate how many times you need to add the LSB calculated in step 1 to the negative full scale ( $-FS$ ) to reach the trigger threshold value.

The maximum threshold value is  $2^{\text{resolution}} - 1$ . The formula is shown here:

$$\text{Abs}(-FS - \text{threshold in volts}) \div (\text{LSB}) = \text{threshold in counts}$$

Here are two examples that use this formula — one for 8-bit trigger resolution, and one for 12-bit trigger resolution:

- 8-bit example using the  $\pm 10$  V range with a  $-5$  V threshold:

Calculate the LSB:  $\text{LSB} = 20 \div 2^8 = 20 \div 256 = 0.078125$

Calculate the threshold:  $\text{Abs}(-10 - (-5)) \div 0.078125 = 5 \div 0.078125 = 64$  (round this result if it is not an integer). A count of 64 translates to a voltage threshold of  $-5.0$  V.

- 12-bit example using the  $\pm 10$  V range with a  $+1$  V threshold:

Calculate the LSB:  $\text{LSB} = 20 \div 2^{12} = 20 \div 4096 = 0.00488$

Calculate the threshold:  $\text{Abs}(-10 - 1) \div 0.00488 = 11 \div 0.00488 = 2254$  (rounded from 2254.1). A count of 2254 translates to a voltage threshold of  $0.99952$  V.

## cbC7266Config() function

Configures 7266 counter for desired operation. This function can only be used with boards that contain a 7266 counter chip (Quadrature Encoder boards). For more information, see LS7266R1 data sheet in accompanying [ls7266r1.pdf](#) file located in the *Documents* subdirectory where the UL is installed (C:\Program files\Measurement Computing\DAQ by default).



## Function Prototype

C/C++

```
int cbC7266Config(int BoardNum, int CounterNum, int Quadrature, int CountingMode, int DataEncoding, int IndexMode, int InvertIndex, int FlagPins, int Gating)
```

Visual Basic

```
Function cbC7266Config(ByVal BoardNum&, ByVal CounterNum&, ByVal Quadrature&, ByVal CountingMode&, ByVal DataEncoding&, ByVal IndexMode&, ByVal InvertIndex&, ByVal FlagPins&, ByVal Gating&) As Long
```

## Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with InstaCal. The specified board must have an LS7266 counter. BoardNum may be 0 to 99.

*CounterNum*

Counter Number (1 - n), where n is the number of counters on the board.

*Quadrature*

Selects the resolution multiplier for quadrature input, or disables quadrature input (NO\_QUAD) so that the counters can be used as standard TTL counters. NO\_QUAD, X1\_QUAD, X2\_QUAD, or X4\_QUAD.

*CountingMode*

Selects operating mode for the counter. NORMAL\_MODE, RANGE\_LIMIT, NO\_RECYCLE, MODULO\_N. Set it to one of the constants in the [CountingMode argument values](#) section below.

*DataEncoding*

Selects the format of the data that is returned by the counter - either Binary or BCD format. BCD\_ENCODING, BINARY\_ENCODING.

*IndexMode*

Selects which action will be taken when the Index signal is received. The IndexMode must be set to INDEX\_DISABLED whenever a Quadrature is set to NON\_QUAD or when Gate is set to ENABLED. Set it to one of the constants in the [IndexMode argument values](#) section below.

*InvertIndex*

Selects the polarity of the Index signal. If set to DISABLED, the Index signal is assumed to be positive polarity. If set to ENABLED, the Index signal is assumed to be negative polarity.

*FlagPins*

Selects which signals will be routed to the FLG1 and FLG2 pins. Set it to one of the constants in the [FlagPins argument values](#) section below.

*Gating*

If gating is set to ENABLED, then the channel INDEX input is routed to the RCNTR pin of the LS7266 chip, and is used as a gating signal for the counter. Whenever Gating = ENABLED the IndexMode must be set to INDEX\_DISABLED.

## Returns

- [Error code](#) or 0 if no error occurs

### CountingMode argument values

NORMAL_MODE	Each counter operates as a 24 bit counter that rolls over to 0 when the maximum count is reached.
RANGE_LIMIT	In range limit count mode, an upper and lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by loading the PRESET register with the <a href="#">cbCLoad()</a> function after the counter has been configured. The lower limit is always 0. When counting up, the counter freezes whenever the count reaches the value that was loaded into the PRESET register. When counting down, the counter freezes at 0. In either case the counting is resumed only when the count direction is reversed.
NO_RECYCLE	In non-recycle mode the counter is disabled whenever a count overflow or underflow takes place. The counter is re-enabled when a reset or load operation is performed on the counter.
MODULO_N	In modulo-n mode, an upper limit is set by loading the PRESET register with a maximum count. Whenever counting up, when the maximum count is reached, the counter will roll-over to 0 and continue counting up. Likewise when counting down, whenever the count reaches 0, it will roll over to the maximum count (in the PRESET register) and continue counting down.

### IndexMode argument values

INDEX_DISABLED	The Index signal is ignored.
LOAD_CTR	The channel INDEX input is routed to the LCNTR pin of the LS7266 counter chip. The counter is loaded whenever the signal occurs.
LOAD_OUT_LATCH	The channel INDEX input is routed to the LCNTR pin of the LS7266 counter chip. The current count is latched whenever the signal occurs. When this mode is selected, the <a href="#">cbCIn()</a> function will return the same count value each time it is called until the Index signal occurs.
RESET_CTR	The channel INDEX input is routed to the RCNTR pin of the LS7266 counter chip. The counter is reset whenever the signal occurs.

### FlagPins argument values

CARRY_BORROW	FLG1 pin is CARRY output, FLG2 is BORROW output.
COMPARE_BORROW	FLG1 pin is COMPARE output, FLG2 is BORROW output.
CARRYBORROW_UPDOWN	FLG1 pin is CARRY/BORROW output, FLG2 is UP/DOWN signal.
INDEX_ERROR	FLG1 pin is INDEX output, FLG2 is error output.

## cbC8254Config() function

Configures 8254 counter for desired operation. This function can only be used with 8254 counters.

For more information, see the 82C54 data sheet in accompanying [82C54.pdf](#) file located in the *Documents* subdirectory where the UL is installed (C:\Program files\Measurement Computing\DAQ by default).



## Function Prototype

C/C++

```
int cbC8254Config(int BoardNum, int CounterNum, int Config)
```

Visual Basic

```
Function cbC8254Config(ByVal BoardNum&, ByVal CounterNum&, ByVal Config&) As Long
```

## Arguments

*BoardNum*

Refers to the number associated with the board when it was installed with InstaCal. Board must have an 82C54 installed. BoardNum may be 0 to 99.

*CounterNum*

Selects one of the counter channels. An 8254 has 3 counters. The value may be 1 - n, where n is the number of 8254 counters on the board (see board-specific information in the *Universal Library User's Guide*).

*Config*

Refer to the 8254 data sheet for a detailed description of each of the configurations. Set it to one of the constants in the "[Config argument values](#)" below.

## Returns

- [Error code](#) or 0 if no errors

## Config argument values

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 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory where you installed the UL.
HIGHONLASTCOUNT	Output of counter (OUT N) transitions from low to high on terminal count and remains high until reset. See Mode 0 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory where you installed the UL (C:\Program Files\Measurement Computing\DAQ by default).
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 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory where you installed the UL.
RATEGENERATOR	Output of counter (OUT N) pulses low for one clock cycle on terminal count, reloads counter and recycles. See Mode 2 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory where you installed the UL.
SOFTWARESTROBE	Output of counter (OUT N) pulses low for one clock cycle on terminal count. Count starts after counter is loaded. See Mode 4 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory where you installed the UL.
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 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory where you installed the UL.

## cbC8536Config() function

Configures an 8536 counter for desired operation. This function can only be used with 8536 counters.

For more information, refer to the *Zilog 8536 product specification*. The document is available on our web site at <http://www.mccdaq.com/PDFmanuals/Z8536.pdf>.



## Function Prototype

C/C++

```
int cbC8536Config(int BoardNum, int CounterNum, int OutputControl, int RecycleMode, int TrigType)
```

Visual Basic

```
Function cbC8536Config(ByVal BoardNum&, ByVal CounterNum&, ByVal OutputControl&, ByVal RecycleMode&,  
ByVal TrigType&) As Long
```

## Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with InstaCal. The board must have an 8536 counter. BoardNum may be 0 to 99.

*CounterNum*

Selects one of the counter channels. An 8536 has three counters. The value may be 1, 2 or 3. INT32 Series boards have two chips installed, so the CounterNum value may be 1 to 6.

*OutputControl*

Specifies the action of the output signal. Set it to one of the constants in the "[OutputControl argument values](#)" section below.

*RecycleMode*

If set to RECYCLE (as opposed to ONETIME), the counter automatically reloads to the starting count every time it reaches 0, then counting continues.

*TrigType*

Specifies the trigger type. Set it to one of the constants in the "[TrigType argument values](#)" section below.

## Returns

- [Error code](#) or 0 if no errors

## OutputControl argument values

HIGHPULSEONTC	Output transitions from low to high for one clock pulse on the terminal count.
TOGGLEONTC	Output changes state on the terminal count.
HIGHUNTILTC	Output transitions to high at the start of counting, then goes low on the terminal count.

## TrigType argument values

HW_START_TRIG	The first trigger on the counter's trigger input initiates loading of the initial count. Counting proceeds from the initial count.
HW_RETRIG	Every trigger on the counter's trigger input initiates loading of the initial count. Counting proceeds from the initial count.
SW_START_TRIG	The <a href="#">cbCLoad()</a> function initiates loading of the initial count. Counting proceeds from the initial count.

# cbC8536Init() function

Initializes the counter linking features of an 8536 counter chip. The linking of counters 1 & 2 must be accomplished prior to enabling the counters.

Refer to the Zilog 8536 product specification for a description of the hardware affected by this mode. The document is available on our web site at <http://www.mccdcaq.com/PDFmanuals/Z8536.pdf>.



## Function Prototype

```
C/C++
int cbC8536Init(int BoardNum, int ChipNum, int CtrlOutput)

Visual Basic
Function cbC8536Init(ByVal BoardNum&, ByVal ChipNum&, ByVal CtrlOutput&) As Long
```

## Arguments

- BoardNum*  
Refers to the board number associated with the board when it was installed with InstaCal. The specified board must have an 8536. BoardNum may be 0 to 99.
- 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. Set it to one of the constants in the "[CtrlOutput argument values](#)" section below.

## Returns

- [Error code](#) or 0 if no errors.

## CtrlOutput argument values

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 the counter #2 clock input.

## cbC9513Config() function

Sets all of the configurable options of a 9513 counter. For more information, refer to the AM9513A data sheet in accompanying [9513A.pdf](#) file located in the *Documents* subdirectory where you installed the UL (C:\Program files\MeasurementComputing\DAQ by default).



## Function Prototype

C/C++

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

Visual Basic

```
Function cbC9513Config(ByVal BoardNum&, ByVal CounterNum&, ByVal GateControl&, ByVal CounterEdge&, ByVal CountSource&, ByVal SpecialGate&, ByVal Reload&, ByVal RecycleMode&, ByVal BCDMode&, ByVal CountDirection&, ByVal OutputControl&) As Long
```

## Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with InstaCal. The specified board must have a 9513 counter. BoardNum may be 0 to 99.

*CounterNum*

Counter number (1 - n) where n is the number of counters on the board. For example, a CIO-CTR5 has 5, a CIO-CTR10 has 10, etc. See board-specific information in the *UL User's Guide*.

*GateControl*

Sets the gating response for level, edge, etc. Set it to one of the constants in the "[GateControl argument values](#)" section below.

*CounterEdge*

Which edge to count. Referred to as "Source Edge" in 9513 data book. Can be set to POSITIVEEDGE (count on rising edge) or NEGATIVEEDGE (count on falling edge).

*CountSource*

Each counter may be set to count from one of 16 internal or external sources. Set it to one of the constants in the "[CountSource argument values](#)" section below.

*SpecialGate*

Special gate may be enabled or disabled (CBENABLED or CBDISABLED in Visual Basic).

*Reload*

Reload the counter from the load register (Reload = LOADREG) or alternately load from the load register, then the hold register (Reload = LOADANDHOLDREG).

*RecycleMode*

Execute once (RecycleMode = ONETIME) or reload and recycle (RecycleMode = RECYCLE).

*BCDMode*

Counter may operate in binary coded decimal count (ENABLED) or binary count (DISABLED) (CBENABLED or CBDISABLED in Visual Basic).

*CountDirection*

AM9513 may count up (COUNTUP) or down (COUNTDOWN).

*OutputControl*

The type of output desired. Set it to one of the constants in the "[OutputControl argument values](#)" section below.

## Returns

- [Error code](#) or 0 if no errors

## GateControl argument values

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

### CountSource argument values

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
ALWAYSLOW	Inactive, Output Low

### OutputControl argument values

HIGHPULSEONTCTC	High pulse on Terminal Count
TOGGLEONTCTC	TC Toggled
DISCONNECTED	Inactive, Output High Impedance
LOWPULSEONTCTC	Active Low Terminal Count Pulse
3, 6, 7	(numeric values) Illegal

### Notes

- The information provided here and in [cbC9513Config\(\)](#) will help you understand how Universal Library syntax corresponds to the 9513 data sheet, but is not a substitute for the data sheet. You cannot program and use a 9513 without this data sheet. Refer to the accompanying [9513A.pdf](#) datasheet located in the *Documents* subdirectory where you installed the UL (C:\Program Files\Measurement Computing\DAQ by default).

## cbC9513Init() function

Initializes all of the chip level features of a 9513 counter chip. This function can only be used with 9513 counters. For more information refer to the AM9513A data sheet in the accompanying [9513A.pdf](#) file located in the *Documents* subdirectory where you installed the UL (C:\Program files\Measurement Computing\DAQ by default).



## Function Prototype

C/C++

```
int cbC9513Init(int BoardNum, int ChipNum, int FOutDivider, int FOutSource, int Compare1, int Compare2,
int TimeOfDay)
```

Visual Basic

```
Function cbC9513Init(ByVal BoardNum&, ByVal ChipNum&, ByVal FOutDivider&, ByVal FOutSource&, ByVal
Compare1&, ByVal Compare2&, ByVal TimeOfDay&) As Long
```

## Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with InstaCal. The specified board must have a 9513 counter. BoardNum may be 0 to 99.

*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, and for a CTR20 it should be 1-4.

*FOutDivider*

F-Out divider (0-15). If set to 0, FoutDivider is the rate of FoutSource divided by 16. If set to a number between 1 and 15, FoutDivider is the rate of FoutSource divided by FoutDivider.

*FOutSource*

Specifies source of the signal for F-Out signal. Set it to one of the constants in the "[FOutSource argument values](#)" section below.

*Compare1*

Compare1 ENABLED or Compare1 DISABLED. (CBENABLED or CBDISABLED in Visual Basic).

*Compare2*

Compare2 ENABLED or Compare2 DISABLED. (CBENABLED or CBDISABLED in Visual Basic).

*TimeOfDay*

TimeOfDay ENABLED or TimeOfDay DISABLED. (CBENABLED or CBDISABLED in Visual Basic). The options for this argument are listed in the "[TimeOfDay argument values](#)" section below.

## Returns

- [Error code](#) or 0 if no errors

## FOutSource argument values and 9513 Data Sheet Equivalent

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

## TimeOfDay argument values and 9513 Data Sheet Equivalent

CBDISABLED	TOD Disabled
1	TOD Enabled / 5 Input
2	TOD Enabled / 6 Input
3	TOD Enabled / 10 Input

No Arguments - For:

- 0 (FOUT on): FOUT Gate
- 0 (Data bus matches board): Data Bus Width
- 1 (Disable Increment): Data Pointer Control
- 1 (BCD Scaling): Scalar Control

## Notes

- The information provided here and in [cbC9513Config\(\)](#) will help you understand how Universal Library syntax corresponds to the 9513 data sheet, but is not a substitute for the data sheet. You cannot program and use a 9513 without this data sheet.  
Refer to the accompanying [9513A.pdf](#) file located in the Documents subdirectory where you installed the UL (C:\Program files\Measurement Computing\DAQ by default).

## cbCClear() function

Clears a scan counter value (sets it to zero). This function only works with counter boards that have counter scan capability.

### Function Prototype

C/C++

```
int cbCClear(int BoardNum, int CounterNum)
```

Visual Basic

```
Function cbCClear(ByVal BoardNum&, ByVal CounterNum&) As Long
```

### Arguments

*BoardNum*

The board number associated with the board when it was installed with InstaCal. The specified board must have a counter. BoardNum may be 0 to 99.

*CounterNum*

The counter to clear. Note: This argument is zero-based (the first counter number to clear is "0").

### Returns

- [Error code](#) or 0 if no errors

## cbCConfigScan() function

Configures a counter channel. This function only works with counter boards that have counter scan capability.

### Function Prototype

C/C++

```
int cbCConfigScan(int BoardNum, int CounterNum, int Mode, int DebounceTime, int DebounceMode, int EdgeDetection, int TickSize, int MappedChannel)
```

Visual Basic

```
Function cbCConfigScan(ByVal BoardNum&, ByVal CounterNum&, ByVal Mode&, ByVal DebounceTime&, ByVal DebounceMode&, ByVal EdgeDetection&, ByVal TickSize&, ByVal MappedChannel&) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal. The specified board must have a scan counter.

*CounterNum*

The counter to set up. **Note:** This argument is zero-based (the first counter number to set up is "0").

*Mode*

Bit fields that control various options. This field may contain any combination of non-contradictory choices in the "[Mode argument values](#)" section below.

*DebounceTime*

Used to bypass the debounce mode, or to set a channel's comparator output to one of 16 debounce times. Debounce is used to eliminate switch-induced transients typically associated with electromechanical devices including relays, proximity switches, and encoders. The choices are:

- CTR\_DEBOUNCE500ns
- CTR\_DEBOUNCE1500ns
- CTR\_DEBOUNCE3500ns
- CTR\_DEBOUNCE7500ns
- CTR\_DEBOUNCE15500ns
- CTR\_DEBOUNCE31500ns
- CTR\_DEBOUNCE63500ns
- CTR\_DEBOUNCE127500ns
- CTR\_DEBOUNCE100us
- CTR\_DEBOUNCE300us
- CTR\_DEBOUNCE700us
- CTR\_DEBOUNCE1500us
- CTR\_DEBOUNCE3100us
- CTR\_DEBOUNCE6300us
- CTR\_DEBOUNCE12700us
- CTR\_DEBOUNCE25500us
- CTR\_DEBOUNCE\_NONE

*DebounceMode*

Sets the mode of the debounce module to CTR\_TRIGGER\_AFTER\_STABLE or to CTR\_TRIGGER\_BEFORE\_STABLE.

**CTR\_TRIGGER\_AFTER\_STABLE:** This mode rejects glitches and only passes state transitions after a specified period of stability (the debounce time). This mode is used with electromechanical devices like encoders and mechanical switches to reject switch bounce and disturbances due to a vibrating encoder that is not otherwise moving. The debounce time should be set short enough to accept the desired input pulse but longer than the period of the undesired disturbance.

**CTR\_TRIGGER\_BEFORE\_STABLE:** Use this mode when the input signal has groups of glitches and each group is to be counted as one. The trigger before stable mode will recognize and count the first glitch within a group but reject the subsequent glitches within the group if the debounce time is set accordingly. In this case the debounce time should be set to encompass

one entire group of glitches

#### EdgeDetection

Selects whether to detect rising edge or falling edge. Choices are: CTR\_RISING\_EDGE and CTR\_FALLING\_EDGE.

If a counter is configured for CTR\_FALLING\_EDGE, calling [cbCIn\(\)](#) or [cbCIn32\(\)](#) for that counter will result in a [BADCOUNTERMODE](#) error.

#### TickSize

Sets the tick size, which is the fundamental unit of time for period, pulsewidth, and timing measurements. The choices are:

- CTR\_TICK20PT83ns
- CTR\_TICK208PT3ns
- CTR\_TICK2083PT3ns
- CTR\_TICK20833PT3ns

#### MappedChannel

Used to select the mapped channel. A mapped channel is one of the input channels on a counter other than CounterNum that can participate with the input signal of the counter defined by CounterNum by gating the counter or decrementing the counter.

## Returns

- [Error code](#) or 0 if no errors

## Mode argument values

### ■ TOTALIZE mode

Sets the specified counter to totalize mode. This mode may contain any combination of non-contradictory choices from the following list of options:

CLEAR_ON_READ	The counter counts up and is cleared at the beginning of every sample. By default, the counter counts up and only clears the counter at the start of a new scan command.
STOP_AT_MAX	The counter will stop at the top of its count. For the <a href="#">cbCIn32()</a> function, the top of the count depends on whether the BIT_32 option is used. If it is, the top of the count is FFFFFFFF hex. If not, the top of the count is FFFF hex. By default, the counter counts upward and rolls over on the 32-bit boundary.
DECREMENT_ON	Allows the mapped channel to decrement the counter. With this option, the main counter input channel will increment the counter, and the mapped channel can be used to decrement the counter. By default, the counter decrement option is set to "off."  This mode is not compatible with <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> . If a counter is configured for DECREMENT_ON, calling <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> for that counter will result in a BADCOUNTERMODE error.
GATING_ON	Selects gating "on." When "on", the counter is enabled when the mapped channel used to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value. By default, the counter gating option is set to "off."  This mode is not compatible with <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> . If a counter is configured for GATING_ON, calling <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> for that counter will result in a BADCOUNTERMODE error.
LATCH_ON_MAP	Causes the count to be latched by the signal on the mapped channel. By default, the count is latched by the internal "start of scan" signal, so the count is updated each time it's read.  This mode is not compatible with <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> . If a counter is configured for LATCH_ON_MAP, calling <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> for that counter will result in a BADCOUNTERMODE error.
BIT_32	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn32()</a> . (Using the BIT_32 option with <a href="#">cbCIn()</a> is not very useful, since the value returned by <a href="#">cbCIn()</a> is only 16 bits. The effect is that the value returned by <a href="#">cbCIn()</a> rolls over 65,535 times before stopping.) Refer to board-specific information for the product you are using for details on how this affects asynchronous reads on a specific device.
BIT_48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Using the BIT_48 option with <a href="#">cbCIn()</a> and <a href="#">cbCIn32()</a> is not very useful, since the value returned by

	cbCIn() is only 16 bits, and the value returned by cbCIn32() is only 32 bits. The effect is that the value returned by cbCIn() rolls over 4,294,967,295 times before stopping, and the value returned by cbCIn32() rolls over 65,535 times before stopping.) Refer to board-specific information for the product you are using for details on how this affects asynchronous reads on a specific device.
UPDOWN_ON	Enables Up/down counting mode.
RANGE_LIMIT_ON	Enables Range Limit counting mode. In Range Limit mode, an upper and lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by loading the max limit register with the cbCLoad, cbCLoad32 or cbCLoad64 functions. The lower limit is always 0. When counting up, the counter freezes whenever the count reaches the value that was loaded into the max limit register.
NO_RECYCLE_ON	Enables Non-recycle counting mode. In Non-recycle mode, the counter is disabled whenever a count overflow or underflow takes place. The counter is re-enabled when a clear or a load operation is performed on the counter
MODULO_N_ON	Enables Modulo-N counting mode. In Modulo-N mode, an upper limit is set by loading the max limit register with a maximum count. When counting up, the counter will roll-over to 0 when the maximum count is reached, and then continue counting up. Likewise when counting down, the counter will roll over to the maximum count (in the max limit register) whenever the count reaches 0, and then continue counting down.

## ■ ENCODER mode

Sets the specified counter to encoder measurement mode. This mode may contain any combination of non-contradictory choices from the following list of options:

ENCODER_MODE_X1	Sets the encoder measurement mode to X1.
ENCODER_MODE_X2	Sets the encoder measurement mode to X2.
ENCODER_MODE_X4	Sets the encoder measurement mode to X4.
ENCODER_MODE_LATCH_ON_Z	Selects the Encoder Z mapped signal to latch the counter outputs. This allows the user to know the exact counter value when an edge is present on another counter.
ENCODER_MODE_CLEAR_ON_Z_ON	Selects "clear on Z" on. The counter is cleared on the rising edge of the mapped (Z) channel. By default, the "clear on Z" option is off, and the counter is not cleared.
ENCODER_MODE_BIT_16	Selects a 16-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with cbCIn().
ENCODER_MODE_BIT_32	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with cbCIn32(). (Using the ENCODER_MODE_BIT_32 option with cbCIn() is not very useful, since the value returned by cbCIn() is only 16 bits. The effect is that the value returned by cbCIn() rolls over 65,535 times before stopping.)
ENCODER_MODE_BIT_48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with cbCIn64(). (Using the ENCODER_MODE_BIT_48 option with cbCIn() and cbCIn32() is not very useful, since the value returned by cbCIn() is only 16 bits, and the value returned by cbCIn32() is only 32 bits. The effect is that the value returned by cbCIn() rolls over 4,294,967,295 times before stopping, and the value returned by cbCIn32() rolls over 65,535 times before stopping.)
ENCODER_MODE_RANGE_LIMIT_ON	Enables Range Limit counting mode. In Range Limit mode, an upper and lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by loading the max limit register with the cbCLoad, cbCLoad32 or cbCLoad64 functions. The lower limit is always 0. When counting up, the counter freezes whenever the count reaches the value that was loaded into the max limit register.
ENCODER_MODE_NO_RECYCLE_ON	Enables Non-recycle counting mode. In Non-recycle mode, the counter is disabled whenever a count overflow or underflow takes place. The counter is re-enabled when a clear or a load operation is performed on the counter
ENCODER_MODE_MODULO_N_ON	Enables Modulo-N counting mode. In Modulo-N mode, an upper limit is set by loading the max limit register with a maximum count. When counting up, the counter will roll-over to 0 when the maximum count is reached, and then continue counting up. Likewise when counting down, the counter will roll over to the maximum count (in the max limit register) whenever the count reaches 0, and then continue counting down.

## ■ PERIOD mode

Sets the specified counter to period measurement mode. This mode may contain any combination of non-contradictory choices from the following list of options:

PERIOD_MODE_X1	The measurement is latched each time one complete period is observed.
PERIOD_MODE_X10	The measurement is latched each time 10 complete periods are observed.
PERIOD_MODE_X100	The measurement is latched each time 100 complete periods are observed.
PERIOD_MODE_X1000	The measurement is latched each time 1000 complete periods are observed.
PERIOD_MODE_GATING_ON	<p>Selects gating "on." When "on", the counter is enabled when the mapped channel used to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value. By default, the counter gating option is set to "off."</p> <p>This mode is not compatible with <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a>. If a counter is configured for PERIOD_MODE_GATING_ON, calling <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> for that counter will result in a BADCOUNTERMODE error.</p>
PERIOD_MODE_BIT_16	Selects a 16-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn()</a> .
PERIOD_MODE_BIT_32	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn32()</a> . (Using the PERIOD_MODE_BIT_32 option with <a href="#">cbCIn()</a> is not very useful, since the value returned by <a href="#">cbCIn()</a> is only 16 bits. The effect is that the value returned by <a href="#">cbCIn()</a> rolls over at 65,535 times before stopping.)
PERIOD_MODE_BIT_48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn64()</a> . (Using the PERIOD_MODE_BIT_48 option with <a href="#">cbCIn()</a> and <a href="#">cbCIn32()</a> is not very useful, since the value returned by <a href="#">cbCIn()</a> is only 16 bits, and the value returned by <a href="#">cbCIn32()</a> is only 32 bits. The effect is that the value returned by <a href="#">cbCIn()</a> rolls over 4,294,967,295 times before stopping, and the value returned by <a href="#">cbCIn32()</a> rolls over 65,535 times before stopping.)

## ■ PULSEWIDTH mode

Sets the specified counter to pulsewidth measurement mode. This mode may contain any combination of non-contradictory choices from the following list of options:

PULSEWIDTH_MODE_GATING_ON	<p>Selects gating "on." When "on", the counter is enabled when the mapped channel used to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value. By default, the counter gating option is set to "off."</p> <p>This mode is not compatible with <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a>. If a counter is configured for PULSEWIDTH_MODE_GATING_ON, calling <a href="#">cbCIn()</a> or <a href="#">cbCIn32()</a> for that counter will result in a BADCOUNTERMODE error.</p>
PULSEWIDTH_MODE_BIT_16	Selects a 16-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn()</a> .
PULSEWIDTH_MODE_BIT_32	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn32()</a> . (Using the PULSEWIDTH_MODE_BIT_32 option with <a href="#">cbCIn()</a> is not very useful, since the value returned by <a href="#">cbCIn()</a> is only 16 bits. The effect is that the value returned by <a href="#">cbCIn()</a> rolls over 65,535 times before stopping.)
PULSEWIDTH_MODE_BIT_48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn64()</a> . (Using the PULSEWIDTH_MODE_BIT_48 option with <a href="#">cbCIn()</a> and <a href="#">cbCIn32()</a> is not very useful, since the value returned by <a href="#">cbCIn()</a> is only 16 bits, and the value returned by <a href="#">cbCIn32()</a> is only 32 bits. The effect is that the value returned by <a href="#">cbCIn()</a> rolls over 4,294,967,295 times before stopping, and the value returned by <a href="#">cbCIn32()</a> rolls over 65,535 times before stopping.)

## ■ TIMING mode

Sets the specified counter to timing mode. This mode supports the following options:

TIMING_MODE_BIT_16	Selects a 16-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with <a href="#">cbCIn()</a> .
TIMING_MODE_BIT_32	Selects a 32-bit counter for asynchronous mode. This argument value only



	affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with cbCIn32(). (Using the TIMING_MODE_BIT_32 option with cbCIn() is not very useful, since the value returned by cbCIn() is only 16 bits. The effect is that the value returned by cbCIn() rolls over 65,535 times before stopping.)
TIMING_MODE_BIT_48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">cbCIn()</a> , <a href="#">cbCIn32()</a> and <a href="#">cbCIn64()</a> . Recommended for use only with cbCIn64(). (Using the TIMING_MODE_BIT_48 option with cbCIn() and cbCIn32() is not very useful, since the value returned by cbCIn() is only 16 bits, and the value returned by cbCIn32() is only 32 bits. The effect is that the value returned by cbCIn() rolls over 4,294,967,295 times before stopping, and the value returned by cbCIn32() rolls over 65,535 times before stopping.)

## cbCFreqIn() function

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

### Function Prototype

C/C++

```
int cbCFreqIn(int BoardNum, int SigSource, int GateInterval, unsigned short *Count, long *Freq)
```

Visual Basic

```
Function cbCFreqIn(ByVal BoardNum&, ByVal SigSource&, ByVal GateInterval&, Count%, Freq&) As Long
```

### Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with InstaCal. The specified board must have a 9513 counter. BoardNum may be 0 to 99.

*SigSource*

Specifies the source of the signal from which the frequency is calculated.

The signal to be measured is routed internally from the source specified by SigSource to the clock input of counter 5. On boards with more than one 9513 chip, there is more than one counter 5. Which counter 5 is used is also determined by SigSource. Set it to one of the constants in the "[SigSource argument values](#)" section below.

The value of SigSource determines which chip will be used. CTRINPUT6 through CTRINPUT10, FREQ6 through FREQ10 and GATE6 through GATE9 indicate chip two will be used. The signal to be measured must be present at the chip two input specified by SigSource. Also, the gating connection from counter 4 output to counter 5 gate must be made between counters 4 and 5 OF THIS CHIP (see below).

Refer to board-specific information in the *UL Users Guide* to determine valid values for your board.

*GateInterval*

Gating interval in milliseconds (must be > 0). 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 65,535.

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 is too long the counter will overflow and a [FREQOVERRUN](#) error will occur.

The cbCFreqIn function does not return until the GateInterval has expired. There is no background option. Under Windows, this means that window activity will stop for the duration of the call. Adjust the GateInterval so this does not pose a problem to your user interface.

*Count*

The raw count is returned here.

*Freq*

The measured frequency in Hz is returned here.

### Returns

- [Error code](#) or 0 if no errors
- Count - Count that the frequency calculation is based on is returned here
- Freq - Measured frequency in Hz is returned here

## SigSource argument values

One 9513 chip (Chip 1 used):	CTRINPUT1 through CTRINPUT5
	GATE1 through GATE4
	FREQ1 through FREQ5
Two 9513 chips (Chip 1 or Chip 2 used):	CTRINPUT1 through CTRINPUT10
	GATE 1 through GATE 9 (excluding gate 5)
	FREQ1 through FREQ10
Four 9513 chips (Chips 1- 4 may be used):	CTRINPUT1 through CTRINPUT20
	GATE1 through GATE19 (excluding gates 5, 10, and 15)
	FREQ1 through FREQ20

## Notes

- This function requires an electrical connection between counter 4 output and counter 5 gate. This connection must be made between counters 4 and 5 *on the chip determined* by SigSource.
- [cbC9513Init\(\)](#) must be called for each ChipNum that will be used by this function. The values of FOutDivider, FOutSource, Compare1, Compare2, and TimeOfDay are irrelevant to this function and may be any value shown in the [cbC9513Init\(\)](#) function description.
- If you select an external clock source for the counters, the GateInterval, Count, and Freq settings are only valid if the external source is 1 MHz. Otherwise, you need to scale the values according to the frequency of the external clock source.  
For example, for an external clock source of 2 MHz, increase your GateInterval setting by a factor of 2, and also double the Count and Freq values returned when analyzing your results.

## cbCIn() function

Reads the current count from a counter channel.

### Function Prototype

C/C++

```
int cbCIn(int BoardNum, int CounterNum, unsigned short *Count)
```

Visual Basic

```
Function cbCIn(ByVal BoardNum&, ByVal CounterNum&, Count%) As Long
```

### Arguments

*BoardNum*

The board number associated with the board when it was installed with InstaCal. The specified board must have a counter. BoardNum may be 0 to 99.

*CounterNum*

The counter to read the current count from. Valid values are in the range of 0 to 20, depending on the device and the number of counters available on the device. See product-specific information in the *Universal Library User's Guide*.

*Count*

Counter value is returned here. Refer to the [Notes](#) section below.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- Count: The range of counter values returned are: 0 to 65,535 for C or PASCAL languages.

Refer to your BASIC manual for information on BASIC integer data types. -32,768 to 32,767 for BASIC languages. BASIC reads counters as:

- 65,535 reads as -1
- 32,768 reads as -32,768
- 32,767 reads as 32,767
- 2 reads as 2
- 0 reads as 0
- cbCIn() vs [cbCIn32\(\)](#) vs [cbCIn64\(\)](#)

Although the cbCIn(), cbCIn32(), and cbCIn64() functions perform the same operation, cbCIn32() is the preferred method to use in most situations.

The only difference between the three is that cbCIn() returns a 16-bit count value, cbCIn32() returns a 32-bit value, and cbCIn64() returns a 64-bit value. Both cbCIn() and cbCIn32() can be used, but cbCIn64() is required whenever you need to read count values greater than 32-bits (counts >4,294,967,295) or the upper (more significant) bits will be truncated.

## cbCIn32() function

Reads the current count from a counter, and returns it as a 32 bit integer.

### Function Prototype

C/C++:

```
int cbCIn32(int BoardNum, int CounterNum, unsigned long *Count)
```

Visual Basic:

```
Function cbCIn32(ByVal BoardNum&, ByVal CounterNum&, Count&) As Long
```

### Arguments

*BoardNum*

The board number associated with the board when it was installed with InstaCal. The specified board must have a counter. BoardNum may be 0 to 99.

*CounterNum*

The counter to read the current count from. Valid values are dependent on the device and the number of counters available on the device. See product-specific information in the *Universal Library User's Guide*.

*Count*

Current count value from selected counter is returned here.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- [cbCIn\(\)](#) vs [cbCIn32\(\)](#) vs [cbCIn64\(\)](#)

Although the [cbCIn\(\)](#), [cbCIn32\(\)](#), and [cbCIn64\(\)](#) functions perform the same operation, [cbCIn32\(\)](#) is the preferred method to use in most situations.

The only difference between the three is that [cbCIn\(\)](#) returns a 16-bit count value, [cbCIn32\(\)](#) returns a 32-bit value, and [cbCIn64\(\)](#) returns a 64-bit value. Both [cbCIn\(\)](#) and [cbCIn32\(\)](#) can be used, but [cbCIn64\(\)](#) is required whenever you need to read count values greater than 32-bits (counts >4,294,967,295) or the upper (more significant) bits will be truncated.

## cbCIn64() function

Reads the current count from a counter, and returns it as a 64-bit double word. This function is not supported in Visual Basic, since no appropriate data type is available to accept the Count argument in those languages.

### Function Prototype

C/C++:

```
int cbCIn64(int BoardNum, int CounterNum, ULONGLONG *Count)
```

### Arguments

*BoardNum*

The board number associated with the board when it was installed with InstaCal. The specified board must have a counter. BoardNum may be 0 to 99.

*CounterNum*

The counter to read the current count from. Valid values are dependent on the device and the number of counters available on the device. See product-specific information in the *Universal Library User's Guide*.

*Count*

Current count value from selected counter is returned here.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- [cbCIn\(\)](#) vs [cbCIn32\(\)](#) vs cbCIn64()

Although the cbCIn(), cbCIn32(), and cbCIn64() functions perform the same operation, cbCIn32() is the preferred method to use in most situations.

The only difference between the three is that cbCIn() returns a 16-bit count value, cbCIn32() returns a 32-bit value, and cbCIn64() returns a 64-bit value. Both cbCIn() and cbCIn32() can be used, but cbCIn64() is required whenever you need to read count values greater than 32-bits (counts >4,294,967,295) or the upper (more significant) bits will be truncated.

## cbCInScan() function

Scans a range of scan counter channels, and stores the samples in an array.

### Function Prototype

C/C++

```
int cbCInScan(int BoardNum, int FirstCtr, int LastCtr, long Count, long *Rate, int MemHandle, int Options)
```

Visual Basic

```
Function cbCInScan(ByVal BoardNum&, ByVal FirstCtr&, ByVal LastCtr&, ByVal Count&, Rate&, ByVal MemHandle&, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal. The specified board must have a counter with scan capability.

*FirstCtr*

First counter channel of the scan. This argument is zero-based, so the first counter number is "0".

*LastCtr*

Last counter channel of the scan. This argument is zero-based, so the first counter number is "0".

The maximum allowable channel for both FirstCtr and LastCtr depends on how many scan counters are available on the Measurement Computing device in use.

*Count*

The total number of counter samples to collect. If more than one channel is being sampled then the number of samples collected per channel is equal to Count / (LastCtr – FirstCtr + 1).

*Rate*

The rate at which samples are taken in samples per second.

Rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*MemHandle*

The handle for the Windows buffer to store data. This buffer must have been previously allocated with the [cbWinBufAlloc32\(\)](#) function.

*Options*

Bit fields that control various options. This field may contain any combination of non-contradictory choices in the "[Options argument values](#)" section below.

### Returns

- [Error code](#) or 0 if no errors
- Rate – the actual sampling rate used.
- MemHandle – the collected counter data returned via the Windows buffer.

## Options argument values

BACKGROUND	<p>When the BACKGROUND option is used, control returns immediately to the next line in your program and the data collection from the counters into the buffer continues in the background. If the BACKGROUND option is not used, the <code>cbCInScan()</code> function does not return to your program until all of the requested data has been collected and returned to the buffer.</p> <p>Use <a href="#">cbGetStatus()</a> with CTRFUNCTION to check on the status of the background operation. Use <a href="#">cbStopBackground()</a> with CTRFUNCTION to terminate the background process before it has completed. Execute <code>cbStopBackground()</code> after normal termination of all background functions in order to clear variables and flags.</p>
CONTINUOUS	<p>This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of the buffer and begins again. The only way to stop this operation is by using <a href="#">cbStopBackground()</a> with CTRFUNCTION. Normally, you should use this option with BACKGROUND so that your program regains control.</p>
CTR16BIT	<p>Sets the counter resolution to 16-bits. When using devices that return data in a 16-bit format, create the buffer using <a href="#">cbWinBufAlloc()</a>.</p>
CTR32BIT	<p>Sets the counter resolution to 32-bits. When using devices that return data in a 32-bit format, create the buffer using <a href="#">cbWinBufAlloc32()</a>.</p>
CTR48BIT	<p>Sets the counter resolution to 48-bits. When using devices that return data in a 64-bit format, create the buffer using <a href="#">cbWinBufAlloc64()</a>.</p>
EXTCLOCK	<p>If this option is specified, conversions will be controlled by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal (refer to board-specific information in the <i>UL User's Guide</i>). When this option is used the Rate argument is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to a transfer mode that will allow the maximum conversion rate to be attained unless otherwise specified.</p>
EXTTRIGGER	<p>If this option is specified, sampling does not begin until the trigger condition is met. You can set the trigger condition to rising edge, falling edge, or the level of the digital trigger input with the <a href="#">cbSetTrigger()</a> function. Refer to board-specific information in the <i>UL User's Guide</i>.</p>
HIGHRESRATE	<p>Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>Rate</i> argument above).</p>



## cbCLoad() function

Loads the specified counter's LOAD, HOLD, ALARM, COUNT, PRESET or PRESCALER register with a count. When loading a counter with a starting value, it is never loaded directly into the counter's count register. Rather, it is loaded into the load or hold register. From there, the counter, after being enabled, loads the count from the appropriate register, generally on the first valid pulse.

## Function Prototype

C/C++

```
int cbCLoad(int BoardNum, int RegNum, unsigned LoadValue)
```

Visual Basic

```
Function cbCLoad(ByVal BoardNum&, ByVal RegNum&, ByVal LoadValue&) As Long
```

## Arguments

*BoardNum*

The board number associated with the board when it was installed with InstaCal. The specified board must have a counter. BoardNum may be 0 to 99.

*RegNum*

The register to load the count to. Set it to one of the constants in the "[RegNum argument values](#)" section below.

*LoadValue*

The value to be loaded. Must be between 0 and  $2^{\text{resolution}} - 1$  of the counter. For example, a 16-bit counter is  $2^{16} - 1$ , or 65,535. Refer to the [Visual Basic signed integers](#) information in the *Introduction: Counter Boards* topic.

## Returns

- [Error code](#) or 0 if no errors

## RegNum argument values

LOADREG0 to LOADREG20	Load registers 0 through 20. This may span several chips.
HOLDREG1 to HOLDREG20	Hold registers 1 through 20. This may span several chips. (9513 only)
ALARM1CHIP1	Alarm register 1 of the first counter chip. (9513 only)
ALARM2CHIP1	Alarm register 2 of the first counter chip. (9513 only)
ALARM1CHIP2	Alarm register 1 of the second counter chip. (9513 only)
ALARM2CHIP2	Alarm register 2 of the second counter chip. (9513 only)
ALARM1CHIP3	Alarm register 1 of the third counter chip. (9513 only)
ALARM2CHIP3	Alarm register 2 of the third counter chip. (9513 only)
ALARM1CHIP4	Alarm register 1 of the fourth counter chip. (9513 only)
ALARM2CHIP4	Alarm register 2 of the fourth counter chip. (9513 only)
COUNT1 to COUNT4	Used to initialize the counter. (LS7266 only)
PRESET1 to PRESET4	Used to set the upper limit of the counter in some modes. (LS7266 only)
PRESCALER1 to PRESCALER4	Used for clock filtering (valid values: 0 to 255). (LS7266 only)
MAXLIMITREG0 to MAXLIMITREG7	Max limit register (USB-QUAD08 only)

## Notes

- You cannot load a count-down-only counter with less than 2.
- Counter Types: There are several counter types supported. Refer to the counter chip's data sheet for the registers that are available.
- cbCLoad() vs [cbCLoad32\(\)](#)

Although the cbCLoad() and cbCLoad32() functions perform the same operation, cbCLoad32() is the preferred function to use.

The only difference between the two is that cbCLoad() loads a 16-bit count value, and cbCLoad32() loads a 32-bit value. The only time you need to use cbCLoad32() is to load counts that are larger than 32-bits (counts >4,294,967,295).

# cbCLoad32() function

Loads the specified counter's COUNT, PRESET or PRESCALER register with a count.

## Function Prototype

C/C++

```
int cbCLoad32(int BoardNum, int RegNum, unsigned long LoadValue)
```

Visual Basic

```
Function cbCLoad32(ByVal BoardNum&, ByVal RegNum&, ByVal LoadValue&) As Long
```

## Arguments

*BoardNum*

Refers to the board number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*RegNum*

The register to load the value into. Set it to one of the constants in the "[RegNum argument values](#)" section below.

*LoadValue*

Value to be loaded into RegNum.

## Returns

- [Error code](#) or 0 if no error occurs

### RegNum argument values

LOADREG1 to LOADREG20	Load registers 1 through 20. This may span several chips.
HOLDREG0 to HOLDREG20	Hold registers 0 through 20. This may span several chips. (9513 only)
ALARM1CHIP1	Alarm register 1 of the first counter chip. (9513 only)
ALARM2CHIP1	Alarm register 2 of the first counter chip. (9513 only)
ALARM1CHIP2	Alarm register 1 of the second counter chip. (9513 only)
ALARM2CHIP2	Alarm register 2 of the second counter chip. (9513 only)
ALARM1CHIP3	Alarm register 1 of the third counter chip. (9513 only)
ALARM2CHIP3	Alarm register 2 of the third counter chip. (9513 only)
ALARM1CHIP4	Alarm register 1 of the fourth counter chip. (9513 only)
ALARM2CHIP4	Alarm register 2 of the fourth counter chip. (9513 only)
COUNT1 to COUNT4	Used to initialize the counter. (LS7266 only)
PRESET1 to PRESET4	Used to set the upper limit of the counter in some modes. (LS7266 only)
PRESCALER1 to PRESCALER4	Used for clock filtering (valid values: 0 to 255). (LS7266 only)
MAXLIMITREG0 to MAXLIMITREG7	Max limit register (USB-QUAD08 only)

## Notes

- [cbCLoad\(\)](#) vs cbCLoad32()  
Although the cbCLoad() and cbCLoad32() functions perform the same operation, cbCLoad32() is the preferred function to use. The only difference between the two is that cbCLoad() loads a 16-bit count value, and cbCLoad32() loads a 32-bit value. The only time you need to use cbCLoad32() is to load counts that are larger than 16-bits (counts > 65,535).

# cbCLoad64() function

Loads the specified counter's COUNT, PRESET, or PRESCALER register with a count.

## Function Prototype

```
C/C++
int cbCLoad64(int BoardNum, int RegNum, ULONGLONG LoadValue)
```

## Arguments

- BoardNum*  
Refers to the board number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.
- RegNum*  
The register to load the value into. Set it to one of the constants in the "[RegNum argument values](#)" section below.
- LoadValue*  
Value to be loaded into RegNum.

## Returns

- [Error code](#) or 0 if no error occurs

## RegNum argument values

LOADREG0 to LOADREG20	Load registers 0 through 20. This may span several chips.
HOLDREG1 to HOLDREG20	Hold registers 1 through 20. This may span several chips. (9513 only)
ALARM1CHIP1	Alarm register 1 of the first counter chip. (9513 only)
ALARM2CHIP1	Alarm register 2 of the first counter chip. (9513 only)
ALARM1CHIP2	Alarm register 1 of the second counter chip. (9513 only)
ALARM2CHIP2	Alarm register 2 of the second counter chip. (9513 only)
ALARM1CHIP3	Alarm register 1 of the third counter chip. (9513 only)
ALARM2CHIP3	Alarm register 2 of the third counter chip. (9513 only)
ALARM1CHIP4	Alarm register 1 of the fourth counter chip. (9513 only)
ALARM2CHIP4	Alarm register 2 of the fourth counter chip. (9513 only)
COUNT1 to COUNT4	Used to initialize the counter. (LS7266 only)
PRESET1 to PRESET4	Used to set the upper limit of the counter in some modes. (LS7266 only)
PRESCALER1 to PRESCALER4	Used for clock filtering (valid values: 0 to 255). (LS7266 only)
MAXLIMITREG0 to MAXLIMITREG7	Max limit register (USB-QUAD08 only)

## Notes

- [cbCLoad\(\)](#) vs cbCLoad64()  
Although the cbCLoad() and cbCLoad64() functions perform the same operation, cbCLoad64() is the preferred function to use. The only difference between the two is that cbCLoad() loads a 16-bit count value, and cbCLoad64() loads a 64-bit value. The only time you need to use cbCLoad64() is to load counts that are larger than 32-bits (counts >4,294,967,295).

# cbCStatus() function

Returns status information about the specified counter (7266 counters only). For more information, refer to the LS7261 data sheet in the [LS7266R1.pdf](#) file located in the *Documents* subdirectory where you installed the UL (C:\Program files\Measurement Computing\DAQ by default).

## Function Prototype

```
C/C++
int cbCStatus(int BoardNum, int CounterNum, unsigned long *StatusBits)

Visual Basic
Function cbCStatus(ByVal BoardNum&, ByVal CounterNum&, StatusBits&) As Long
```

## Arguments

- BoardNum*
- The board number associated with the board when it was installed with InstaCal. The specified board must have an LS7266 counter. BoardNum may be 0 to 99.
- CounterNum*
- Number of the counter whose status bits you want to read. Valid values are 1 to N, where N is the number of counters on the board.
- StatusBits*
- Current status from selected counter is returned here. The status consists of individual bits that indicate various conditions within the counter. Set it to one of the constants in the "[StatusBits argument values](#)" section below.

## Returns

- [Error code](#) or 0 if no error occurs.

## StatusBits argument values

C_UNDERFLOW	Set to 1 whenever the count decrements past 0. Is cleared to 0 whenever cbCStatus() is called.
C_OVERFLOW	Set to 1 whenever the count increments past it's upper limit. Is cleared to 0 whenever cbCStatus() is called.
C_COMPARE	Set to 1 whenever the count matches the preset register. Is cleared to 0 whenever cbCStatus() is called.
C_SIGN	Set to 1 when the MSB of the count is 1. Is cleared to 0 whenever the MSB of the count is set to 0.
C_ERROR	Set to 1 whenever an error occurs due to excessive noise on the input. Is cleared to 0 by calling <a href="#">cbC7266Config()</a>
C_UP_DOWN	Set to 1 when counting up. Is cleared to 0 when counting down.
C_INDEX	Set to 1 when index is valid. Is cleared to 0 when index is not valid.

## cbCStoreOnInt() function

### Changed R4.0 RW

Installs an interrupt handler that will store the current count whenever an interrupt occurs. This function can only be used with 9513 counters. This function will continue to operate in the background until either IntCount is satisfied or [cbStopBackground\(\)](#) with CTRFUNCTION is called.

## Function Prototype

C/C++

```
int cbCStoreOnInt(int BoardNum, int IntCount, short CntrControl[], int MemHandle)
```

Visual Basic

```
Function cbCStoreOnInt(ByVal BoardNum%, ByVal IntCount%, CntrControl%, ByVal MemHandle%) As Long
```

## Arguments

*BoardNum*

The board number associated with the board when it was installed with InstaCal. The specified board must have a 9513 counter. BoardNum may be 0 to 99.

*IntCount*

The counters will be read every time an interrupt occurs until IntCount number of interrupts have occurred. If IntCount is = 0, the function will run until [cbStopBackground\(\)](#) is called. (Refer below to the *MemHandle* argument).

*CntrControl*

The array should have an element for each counter on the board. (5 elements for CTR05 device, 10 elements for a CTR10 device, and so on). Each element corresponds to a counter channel. Each element should be set to either CBDISABLED or CBENABLED. All channels that are set to CBENABLED will be read when an interrupt occurs.

*MemHandle*

Handle for Windows buffer. If IntCount is non-zero, the buffer referenced by MemHandle must be of sufficient size to hold (IntCount × Number of Counters) points.

## Returns

- [Error code](#) or 0 if no errors

## Notes

New functionality: If the Library Revision is set to 4.0 or greater, the following code changes are required:

- If IntCount is non-zero, the buffer referenced by MemHandle must be able to hold (IntCount × Number of Counters) points.

For example, if you set IntCount to 100 for a CTR05 device, allocate the size of the buffer to be  $(100 \times 5) = 500$ . This new functionality keeps the user application from having to move the data out of the buffer for every interrupt before it is overwritten. For each interrupt, the counter values will be stored in adjacent memory locations in the buffer.

Allocate the proper buffer size for non-zero IntCount settings: Specifying IntCount as a non-zero value and failing to allocate the proper sized buffer results in a runtime error. There is no way for the Universal Library to determine if the buffer has been allocated with the proper size.

- If IntCount = 0, the functionality is unchanged.

## cbPulseOutStart() function

Starts a timer to generate digital pulses at a specified frequency and duty cycle. Use [cbPulseOutStop\(\)](#) to stop the output.

### Function Prototype

C/C++

```
int cbPulseOutStart (int BoardNum, int TimerNum, double *Frequency, double *DutyCycle, unsigned int PulseCount, double *InitialDelay, int IdleState, int Options);
```

Visual Basic

```
Function cbPulseOutStart (ByVal BoardNum&, ByVal TimerNum&, Frequency#, DutyCycle#, ByVal PulseCount&, InitialDelay#, ByVal IdleState&, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. The specified board must have a pulse-type counter. BoardNum may be 0 to 99.

*TimerNum*

The timer to start output pulses. Valid values are zero (0) up to the number of timers on the board – 1.

*Frequency*

The desired square wave frequency. The timer clock will be divided down by integer values to produce the frequency. The actual frequency output will be returned. Valid values are dependent on the timer's clock and the timer resolution.

*DutyCycle*

The width of the pulse divided by the pulse period. This ratio is used with the frequency value to determine the pulse width and the interval between pulses.

*PulseCount*

The number of pulses to generate. Setting the pulse count to zero will result in pulses being generated until the [cbPulseOutStop\(\)](#) function is called.

*InitialDelay*

The amount of time to delay before starting the timer output after enabling the output.

*IdleState*

The resting state of the output. Set it to one of the IdleState constants. Choices are:

0 = IDLE\_LOW  
1 = IDLE\_HIGH

*Options*

Reserved for future use.

### Returns

- [Error code](#) or 0 if no errors
- Frequency – the actual frequency set.

## cbPulseOutStop() function

Stops a timer output. Use [cbPulseOutStart\(\)](#) to start the output.

### Function Prototype

C/C++

```
int cbPulseOutStop (int BoardNum, int TimerNum);
```

Visual Basic

```
Function cbPulseOutStop(ByVal BoardNum&, ByVal TimerNum&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. The specified board must have a pulse-type counter. BoardNum may be 0 to 99.

*TimerNum*

The timer to stop. Valid values are zero (0) up to the number of timers on the board – 1.

### Returns

- [Error code](#) or 0 if no errors

## cbTimerOutStart() function

Starts a timer square wave output. Use [cbTimerOutStop\(\)](#) to stop the output.

### Function Prototype

C/C++

```
int cbTimerOutStart(int BoardNum, int TimerNum, double *Frequency)
```

Visual Basic

```
Function cbTimerOutStart(ByVal BoardNum&, ByVal TimerNum&, Frequency#) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. The specified board must have a timer-type counter. BoardNum may be 0 to 99.

*TimerNum*

The timer to output the square wave from. Valid values are zero (0) up to the number of timers – 1 on the board.

*Frequency*

The desired square wave frequency. The timers clock will be divided down by integer values to produce the frequency. The actual frequency output will be returned. Valid values are dependant on the timer's clock and the timer resolution.

### Returns

- [Error code](#) or 0 if no errors
- Frequency – the actual frequency set.



## cbTimerOutStop() function

Stops a timer square wave output. Use [cbTimerOutStart\(\)](#) to start the output.

### Function Prototype

C/C++

```
int cbTimerOutStop(int BoardNum, int TimerNum)
```

Visual Basic

```
Function cbTimerOutStop(ByVal BoardNum&, ByVal TimerNum&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with the InstaCal. The specified board must have a timer-type counter. BoardNum may be 0 to 99.

*TimerNum*

The timer to stop. Valid values are zero up to the number of timers on the board – 1.

### Returns

- [Error code](#) or 0 if no errors

## cbLogConvertFile() function

Converts a binary log file to a comma-separated values (.CSV) text file or another text file format that you specify.

### Function Prototype

C/C++

```
int cbLogConvertFile(char* srcFile, char* destFile, int startSample, int count, int delimiter)
```

Visual Basic

```
Function cbLogConvertFile(ByVal Filename$, ByVal DestFilename$, ByVal StartSample&, ByVal SampleCount&, ByVal Delimiter&) As Long
```

### Arguments

*srcFile*

The name and path of the binary file to read.

*destFile*

The name and destination path of the converted file. Use the file extension of the file type that you want to create.

*startSample*

The index number of the first sample to read.

*count*

The number of samples to read.

*delimiter*

Specifies the character used between fields in the converted file. Set to one of the Delimiter constants. Choices are:

0 = DELIMITER\_COMMA

1 = DELIMITER\_SEMICOLON

2 = DELIMITER\_SPACE

3 = DELIMITER\_TAB

### Returns

- [Error code](#) or 0 if no errors

### Notes

- Timestamp data is stored according to the timeZone and timeFormat arguments. Refer to [cbLogSetPreferences\(\)](#).
- Timestamps in the converted file may be in either 12-hour or 24-hour format based on the setting of the timeFormat argument. Timestamps can optionally be converted to local time based on the setting of the timeZone argument.
- AI temperature data is returned according to the Units preference. Refer to [cbLogSetPreferences\(\)](#).
- The Units preference is only applied to the AI data if the data was logged as temperature data. Refer to [cbLogGetAIInfo\(\)](#). This value is ignored if the AI data was logged as raw data.
- The units preference is always applied to CJC data, since it is always logged as temperature data.
- If the destFile argument ends with a .CSV extension, the delimiter argument must be set to DELIMITER\_COMMA. Otherwise, an [INVALIDDELIMITER](#) error is returned.
- You can open a comma-separated values text file (.CSV) directly in Microsoft Excel. Text files with extensions other than .CSV can only be imported into Excel.

## cbLogGetAIChannelCount() function

Returns the total number of analog input channels logged in a binary file.

### Function Prototype

C/C++

```
int cbLogGetAIChannelCount(char* Filename, int* AICount)
```

Visual Basic

```
Function cbLogGetAIChannelCount(ByVal Filename$, ByRef AICount&) As Long
```

### Arguments

*Filename*

The name of the file to retrieve the information from.

*AICount*

The number of analog input channels logged in the binary file.

### Returns

- [Error code](#) or 0 if no errors
- AICount – Returns the number of analog input channels logged in the binary file.

## cbLogGetAIInfo() function

Returns the channel number and unit value of each analog input channel logged in a binary file.

### Function Prototype

C/C++

```
int cbLogGetAIInfo(char* Filename, int* ChannelNumbers, int* Units)
```

Visual Basic

```
Function cbLogGetAIInfo(ByVal Filename$, ByRef ChannelNumbers&, ByRef Units&) As Long
```

### Arguments

*Filename*

The name of the file to retrieve the information.

*ChannelNumbers*

An array that contains the analog input channel numbers logged in the file.

*Units*

An array that contains the unit values set for the device in InstaCal for each analog input channel logged in the file.

### Returns

- [Error code](#) or 0 if no errors
- ChannelNumbers – Returns the analog input channel numbers logged in the binary file.
- Units – Returns the unit values set for the device in InstaCal for each analog input channel logged in the binary file. Returned values include:
  - 0 = Units\_Temperature
  - 1 = Units\_Raw

## cbLogGetCJCInfo() function

Returns the number of CJC temperature channels logged in a binary file.

### Function Prototype

C/C++

```
int cbLogGetCJCInfo(char* Filename, int* CJCCount)
```

Visual Basic

```
Function cbLogGetCJCInfo(ByVal Filename$, ByRef CJCCChannelCount&) As Long
```

### Arguments

*Filename*

The name of the file to retrieve the information from.

*CJCCount*

The number of CJC temperature channels logged in the file.

### Returns

- [Error code](#) or 0 if no errors
- CJCCount – Returns the number of CJC channels logged in the binary file.

## cbLogGetDIOInfo() function

Returns the number of digital I/O channels logged in a binary file.

### Function Prototype

C/C++

```
int cbLogGetDIOInfo(char* Filename, int* DIOCount)
```

Visual Basic

```
Function cbLogGetDIOInfo(ByVal Filename$, ByRef DIOChannelCount&) As Long
```

### Arguments

*Filename*

The name of the file to retrieve the information from.

*DIOCount*

The number of digital I/O channels logged in the binary file.

### Returns

- [Error code](#) or 0 if no errors
- DIOCount – Returns the number of digital I/O channels logged in the binary file.

## cbLogGetFileInfo() function

Returns the version level and byte size of a binary file.

### Function Prototype

C/C++

```
int cbLogGetFileInfo(char* Filename, int* Version, int* Size)
```

Visual Basic

```
Function cbLogGetFileInfo(ByVal Filename$, ByRef Version&, ByRef Size&) As Long
```

### Arguments

*Filename*

The name of the file to retrieve the information from.

*Version*

The version level of the binary file.

*Size*

The size in bytes of the binary file.

### Returns

- [Error code](#) or 0 if no errors
- Version – Returns the version level of the binary file.
- Size – Returns the size in bytes of the binary file.

## cbLogGetFileName() function

Returns the name of the  $n^{\text{th}}$  file in the directory containing binary log files.

### Function Prototype

C/C++

```
int cbLogGetFileName(int FileNumber, char* Path, char* Filename)
```

Visual Basic

```
Function cbLogGetFileName(ByVal FileNum&, ByVal Path$, ByVal Filename$) As Long
```

### Arguments

*FileNumber*

Index of the file whose name you want to return. Specify one of the following:

- The number ( $n$ ) that represents the location of the file in the directory (where  $n = 0, 1, 2$ , and so on)
- GETFIRST - get the first file in the directory
- GETNEXT - get the next file in the directory, based on the current index.

This parameter is the index of the file in the directory, and is not part of the filename.

*Path*

The full path to the directory containing the binary file. The path must be NULL terminated, and cannot be longer than 256 characters.

*Filename*

A NULL terminated string containing the full path to the file.

### Returns

- [Error code](#) or 0 if no errors
- Filename – Returns a NULL terminated string containing the full path to the file.

### Notes

- Set FileNumber to GETFIRST to access the first binary file in a directory. Subsequent calls with FileNumber = GETNEXT returns each successive file in the directory. When you call the function after accessing the last file in the directory, the function returns the error code [NOMOREFILES](#).



## cbLogGetPreferences() function

Returns API preference settings for time stamped data, analog temperature data, and CJC temperature data. Returns the default values unless changed using [cbLogSetPreferences\(\)](#).

### Function Prototype

C/C++

```
int cbLogGetPreferences(int* TimeFormat, int* TimeZone, int* Units)
```

Visual Basic

```
Function cbLogGetPreferences(ByRef TimeFormat&, ByRef TimeZone&, ByRef Units&) As Long
```

### Arguments

*TimeFormat*

Returns the time format used to display time stamp data. Set to one of the TimeFormat constants. Choices are:

0 = TIMEFORMAT\_12HOUR. For example 2:32:51PM.  
1 = TIMEFORMAT\_24HOUR. For example 14:32:51.

*TimeZone*

Returns the time zone to store time stamp data. Set to one of the TimeZone constants. Choices are:

0 = TIMEZONE\_LOCAL. Converts time stamped data to the local time zone on your computer.  
1 = TIMEZONE\_GMT. Leaves time stamped data in Greenwich Mean Time.

*Units*

Returns the unit to use for analog temperature data. This value is ignored if raw data values are logged. Set to one of the Units constants. Choices are:

0 = FAHRENHEIT  
1 = CELSIUS  
2 = KELVIN

### Returns

- [Error code](#) or 0 if no errors
- TimeFormat – Returns the format to apply to time stamped data from API functions that return time data.
- TimeZone – Returns the time zone to apply to time stamped data from API functions that return time data.
- Units – Returns the unit to use when converting temperature data from API functions that return temperature data.

## cbLogGetSampleInfo() function

Returns the sample interval, sample count, and the date and time of the first data point contained in a binary file.

### Function Prototype

#### Visual Basic

```
Function cbLogGetSampleInfo(ByVal Filename$, ByRef SampleInterval&, ByRef SampleCount&, ByRef  
StartDate&, ByRef StartTime&) As Long
```

#### C/C++

```
int cbLogGetSampleInfo(char* Filename, int* SampleInterval, int* SampleCount, int* StartDate, int*  
StartTime)
```

### Arguments

#### *Filename*

The name of the file to retrieve sample information from.

#### *SampleInterval*

The time interval, in seconds, between samples.

#### *SampleCount*

The number of samples contained in the file.

#### *StartDate*

The date when the first data point was logged in the file. Date values are packed in the following format:

Byte 0: day

Byte 1: month

Byte 2 - 3: year

#### *StartTime*

The time when the first data point was logged in the file. Time values are packed in the following format:

Byte 0: seconds

Byte 1: minutes

Byte 2: hours

Byte 3: 0xff = 24hour format; 0x0 = AM; 0x1 = PM

### Returns

- [Error code](#) or 0 if no errors
- SampleInterval – Returns the time interval, in seconds, between samples.
- SampleCount – Returns the number of samples in the file.
- StartDate – Returns the date when the first data point was logged in the file.
- StartTime – Returns the time when the first data point was logged in the file.

### Notes

- Time stamped data is returned according to the TimeZone and TimeFormat preferences. Refer to [cbLogSetPreferences\(\)](#).

# cbLogReadAIChannels() function

Reads analog input data from a binary file, and stores the values in an array.

## Function Prototype

C/C++

```
int cbLogReadAIChannels(char* Filename, int StartSample, int Count, float* AIChannels)
```

Visual Basic

```
Function cbLogReadAIChannels(ByVal Filename$, ByVal StartSample&, ByVal SampleCount&, ByRef AIChannelData!) As Long
```

## Arguments

*FileName*

The name of the file to retrieve the information from.

*StartSample*

The first sample to read from the binary file.

*Count*

The number of samples to read from the binary file.

*AIChannels*

Receives the analog input values.

## Returns

- [Error code](#) or 0 if no errors
- AIChannels – Returns the analog input values logged in the file.

## Notes

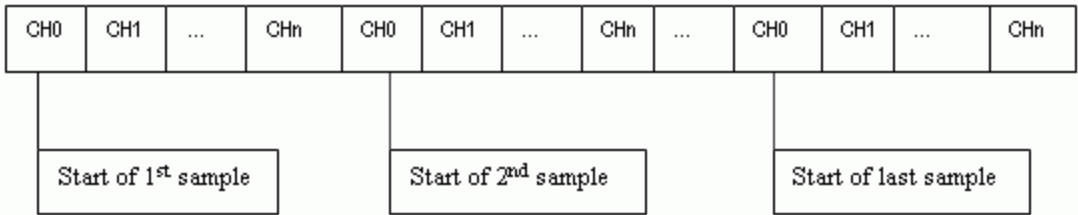
- The units of the analog input data that is returned is set by the value of the Units preference. Refer to [cbLogSetPreferences\(\)](#).
- The units preference is only applied if the logged data is temperature data. This value is ignored if the data logged is raw.

## Analog array

The user is responsible for allocating the size of the analog array, and ensuring that it is large enough to hold the data that will be returned. You can calculate the array allocation using the sampleCount value from [cbLogGetSampleInfo\(\)](#), and the aiCount value from [cbLogGetAIChannelCount\(\)](#):

```
float* AIChannels = new float[sampleCount * AICount];
```

The figure below shows the layout of the analog array, and how the elements should be indexed.



where *n* is (AICount – 1).

CH0 – CHn refer to the channels in the array, not the input channels of the device.

For example, assume that all of the even number input channels are logged. The analog array channels are mapped as shown here:

Array Channel	Device Input Channel
0	0
1	2
2	4
3	6

Use the following code fragment to access the elements of the analog array:

```
for (i=0; i<numberOfSamples; i++)
{
    for (j=0; j<numberOfAChannels; j++)
    {
        a = analogArray[(i * numberOfAChannels) + j];
    }
}
```

where

- the numberOfSamples is set by the SampleCount value from [cbLogGetSampleInfo\(\)](#).
- the numberOfAChannels is set by the AICount value from [cbLogGetAIChannelCount\(\)](#).

# cbLogReadCJCChannels() function

Reads CJC temperature data from a binary file, and stores the values in an array.

## Function Prototype

C/C++

```
int cbLogReadCJCChannels(char* Filename, int StartSample, int Count, float* CJCChannels)
```

Visual Basic

```
Function cbLogReadCJCChannels(ByVal Filename$, ByVal StartSample&, ByVal SampleCount&, ByRef CJCChannelData!) As Long
```

## Arguments

Filename

The name of the file to retrieve the information from.

StartSample

The first sample to read from the binary file.

Count

The number of samples to read from the binary file.

CJCChannels

Receives the CJC temperature values.

## Returns

- [Error code](#) or 0 if no errors
- CJCChannels – Returns the CJC temperature values logged in the file.

## Notes

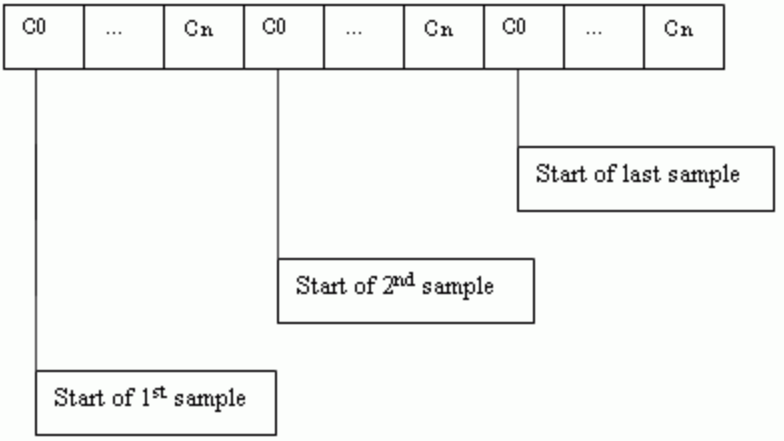
- The temperature scale of the CJC data that is returned is set by the value of the Units preference. Refer to [cbLogSetPreferences\(\)](#).

## CJC array

The user is responsible for allocating the size of the CJC array, and ensuring that it is large enough to hold the data that will be returned. You can calculate the array allocation using the sampleCount value from [cbLogGetSampleInfo\(\)](#), and the cjcCount value from [cbLogGetCJCInfo\(\)](#):

```
float* CJCChannels = new float[SampleCount * CJCCount];
```

The figure below shows the layout of the CJC array, and how the elements should be indexed.



where  $n$  is (CJCCount - 1).

Use the following code fragment to access the elements of the CJC array.

```
for (i=0; i<numberOfSamples; i++)
{
    for (j=0; j<numberOfCJCChannels; j++)
    {
        c = cjcArray[(i * numberOfCJCChannels) + j];
    }
}
```

where

the numberOfSamples is set by the sampleCount value from [cbLogGetSampleInfo\(\)](#).

the numberOfCJCChannels is set by the cjcCount value from [cbLogGetCJCInfo\(\)](#).

# cbLogReadDIOChannels() function

Reads digital I/O channel data from a binary file, and stores the values in an array.

## Function Prototype

C/C++

```
int cbLogReadDIOChannels(char* Filename, int StartSample, int Count, int* DIOChannels)
```

Visual Basic

```
Function cbLogReadDIOChannels(ByVal Filename$, ByVal StartSample&, ByVal SampleCount&, ByRef DIOChannelData&) As Long
```

## Arguments

*FileName*

The name of the file to retrieve the information from.

*StartSample*

The first sample to read from the binary file.

*Count*

The number of samples to read from the binary file.

*DIOChannels*

Receives the DIO input values.

## Returns

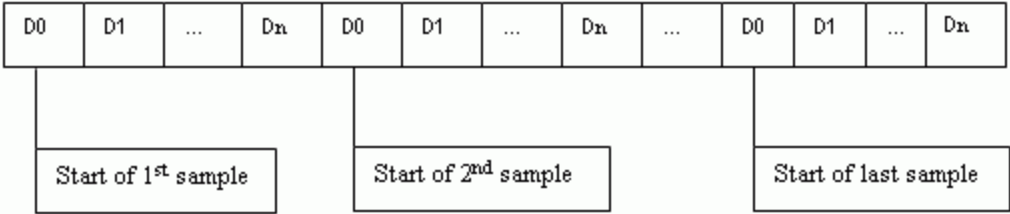
- [Error code](#) or 0 if no errors
- DIOChannels – Returns the DIO channel values logged in the file. Each element of the array contains the value of one bit from a digital channel.

## DIO array

The user is responsible for allocating the size of the DIO array, and ensuring that it is large enough to hold the data that will be returned. You can calculate the array allocation using the SampleCount value from [cbLogGetSampleInfo\(\)](#) and the DIOCount value from [cbLogGetDIOInfo\(\)](#):

```
int* DIOChannels = new int[SampleCount * DIOCount];
```

The figure below shows the layout of the DIO array, and how the elements should be indexed.



Where  $n$  is (DIOCount - 1)

Use the following code fragment to access the elements of the DIO array:

```
for (i=0; i<numberOfSamples; i++)
{
    for (j=0; j<numberOfDIOChannels; j++)
    {
        d = dioArray[(i * numberOfDIOChannels) + j];
    }
}
```

where

the numberOfSamples is set by the SampleCount value from [cbLogGetSampleInfo\(\)](#)

the numberOfDIOChannels is set by the DIOCount value from [cbLogGetDIOInfo\(\)](#)



## cbLogReadTimeTags() function

Reads the date and time values logged in a binary file. This function stores the date values in a dateTags array, and the time values in a timeTags array.

### Function Prototype

C/C++

```
int cbLogReadTimeTags(char* Filename, int StartSample, int Count, int* DateTags, int* TimeTags)
```

Visual Basic

```
Function cbLogReadTimeTags(ByVal Filename$, ByVal StartSample&, ByVal SampleCount&, ByRef Dates&, ByRef Times&) As Long
```

### Arguments

*Filename*

The name of the file to retrieve the information from.

*StartSample*

The first sample to read from the binary file.

*Count*

The number of samples to read from the binary file

*DateTags*

Receives the date value for each sample logged in the file. The dates are packed in the following format:

Byte 0: day

Byte 1: month

Byte 2-3: year

*TimeTags*

Receives the time value for each sample logged in the file. The times are packed in the following format:

Byte 0: seconds

Byte 1: minutes

Byte 2: hours

Byte 3: 0xff = 24hour format; 0x0 = AM; 0x1 = PM

### Returns

- [Error code](#) or 0 if no errors
- DateTags – Returns the date values for each sample logged in the file
- TimeTags – Returns the time values for each sample logged in the file.

### Notes

- Time stamped data is returned according to the timeZone preference value and the timeFormat preference value. Refer to [cbLogSetPreferences\(\)](#).
- Time stamped data are logged in the file if InstaCal is configured to do so. If time stamps are not logged, the TimeTags and DateTags arrays are filled with values calculated from the file header information.

### Array size

The user is responsible for allocating the size of the DateTags and TimeTags arrays, and ensuring that they are large enough to hold the data that is returned. You can calculate the array allocation using the SampleCount value from [cbGetSampleInfo\(\)](#).

```
int* dates = new int[SampleCount];  
int* times = new int[SampleCount];
```

**DateTags array**

The figure below shows the layout of the DateTags array, and how the elements should be indexed.

D0	D1	D2	...	Dn
----	----	----	-----	----

where:  $n$  is  $(\text{SampleCount} - 1)$

Each sample has only one date. Use the following code fragment to access the elements of the DateTags array:

```
for (i=0; i<numberOfSamples; i++)
{
    d = DateTagsArray[i];
}
```

**TimeTags array**

The figure below shows the layout of the TimeTags array, and how the elements should be indexed.

T0	T1	T2	...	<del>Tn</del>
----	----	----	-----	---------------

where:  $n$  is  $(\text{SampleCount} - 1)$

Each sample has only one time stamp. Use the following code fragment to access the elements of the TimeTags array:

```
for (i=0; i<numberOfSamples; i++)
{
    t = TimeTagsArray[i];
}
```

## cbLogSetPreferences() function

Sets preferences for returned time stamped data, analog temperature data, and CJC temperature data.

### Function Prototype

C/C++

```
int cbLogSetPreferences(int TimeFormat, int TimeZone, int Units)
```

Visual Basic

```
Function cbLogSetPreferences(ByVal TimeFormat&, ByVal TimeZone&, ByVal Units&) As Long
```

### Arguments

*TimeFormat*

Specifies the time format to apply when returning time stamped data (when using [cbLogReadTimeTags\(\)](#) for example). Set to one of the TimeFormat constants. Choices are:

0 = TIMEFORMAT\_12HOUR. For example 2:32:51PM (default).

1 = TIMEFORMAT\_24HOUR. For example 14:32:51.

*TimeZone*

Specifies whether to convert time stamped data that is returned (when using [cbLogReadTimeTags\(\)](#) for example) to the local time zone or to return the time stamps as they are stored in the file (in the GMT time zone). Set to one of the TimeZone constants. Choices are:

0 = TIMEZONE\_LOCAL. Converts timestamp data to the local time zone on your computer (default).

1 = TIMEZONE\_GMT. Leaves time stamped data in Greenwich Mean Time.

*Units*

Specifies whether to convert temperature data returned (when using [cbLogReadAICchannels\(\)](#) for example) to Fahrenheit or Kelvin, or return temperature data as they are stored in the file (in Celsius units).

Set to one of the Units constants. Choices are:

0 = FAHRENHEIT (default)

1 = CELSIUS

2 = KELVIN

This value is ignored if raw data is logged.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- The TimeFormat and TimeZone preferences are applied to all time data returned using API functions that return time data.
- The units preference specifies the temperature scale that the API applies when reading and converting analog temperature and CJC data.

## cbDBitIn() function

Reads the state of a single digital input bit.

This function treats all of the DIO ports of a particular type on a board as a single port. It lets you read the state of any individual bit within this port. Note that with some port types, such as 8255 ports, if the port is configured for DIGITALOUT, cbDBitIn provides readback of the last output value.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions.

## Function Prototype

C/C++

```
int cbDBitIn(int BoardNum, int PortType, int BitNum, unsigned short *BitValue)
```

Visual Basic

```
Function cbDBitIn Lib (ByVal BoardNum&, ByVal PortType&, ByVal BitNum&, BitValue%) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortType*

There are three general types of digital ports — ports that are programmable as input or output, ports that are fixed input or output, and ports for which each bit may be programmed as input or output. For the first of these types, set PortType to FIRSTPORTA. For the latter two types, set PortType to AUXPORT. Some boards have both types of digital ports (DAS1600). Set PortType to either FIRSTPORTA or AUXPORT, depending on which digital inputs you wish to read.

*BitNum*

Specifies the bit number within the single large port.

*BitValue*

Place holder for return value of bit. Value will be 0 or 1. A 0 indicates a logic low reading, a 1 indicates a logic high reading. Logic high does not necessarily mean 5 V. 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() function

Sets the state of a single digital output bit.

This function treats all of the DIO ports of a particular type on a board as a single large port. It lets you set the state of any individual bit within this large port.

Most configurable ports require configuration before writing. Check the board-specific information in the *Universal Library User's Guide* to determine if the port should be configured for your hardware. When configurable, use [cbDConfigPort\(\)](#) to configure a port for output, and [cbDConfigBit\(\)](#) to configure a bit for output.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions.

## Function Prototype

C/C++

```
int cbDBitOut(int BoardNum, int PortType, int BitNum, unsigned short BitValue)
```

Visual Basic

```
Function cbDBitOut(ByVal BoardNum&, ByVal PortType&, ByVal BitNum&, ByVal BitValue%) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortType*

There are three general types of digital ports — ports that are programmable as input or output, ports that are fixed input or output, and ports for which each bit may be programmed as input or output. For the first of these types, set PortType to FIRSTPORTA. For the latter two types, set PortType to AUXPORT. Some boards have both types of digital ports (DAS1600). Set PortType to either FIRSTPORTA or AUXPORT, depending on which digital port you wish to write to.

*BitNum*

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

*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 5 V. See the board manual for chip specifications.

## Returns

- [Error code](#) or 0 if no errors

## cbDConfigBit() function

Configures a specific digital bit as Input or Output. This function treats all DIO ports on a board as a single port (AUXPORT). This function is NOT supported by 8255 type DIO ports. Please refer to board-specific information for details.

### Function Prototype

C/C++

```
int cbDConfigBit(int BoardNum, int PortType, int BitNum, int Direction)
```

Visual Basic

```
Function cbDConfigBit(ByVal BoardNum&, ByVal PortType&, ByVal BitNum&, ByVal Direction&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortType*

The port (AUXPORT) whose bits are to be configured. The port specified must be bitwise configurable. Check the board-specific information in the *Universal Library User's Guide* for details.

*BitNum*

The bit number to configure as input or output. See board-specific information for details.

*Direction*

DIGITALOUT or DIGITALIN configures the specified bit for output or input, respectively.

### Returns

- [Error code](#) or 0 if no errors

## cbDConfigPort() function

Configures a digital port as input or output.

This function is for use with ports that may be programmed as input or output, such as those on the 82C55 chips and 8536 chips. Refer to the Zilog 8536 manual for details of chip operation. Also refer to the **82C55 data sheet** that is located in the accompanying [82C55A.pdf](#) file in the *Documents* subdirectory where the UL is installed (C:\Program files\Measurement Computing\DAQ by default).

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions.



## Function Prototype

C/C++

```
int cbDConfigPort(int BoardNum, int PortNum, int Direction)
```

Visual Basic

```
Function cbDConfigPort(ByVal BoardNum&, ByVal PortNum&, ByVal Direction&) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

The specified port must be configurable. For most boards, AUXPORT is not configurable. Check the board-specific information in the *Universal Library User's Guide* for details.

*Direction*

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

## Returns

- [Error code](#) or 0 if no errors

## Note

- When used on ports within an 8255 chip, this function will reset all ports on that chip configured for output to a zero state. This means that if you set an output value on FIRSTPORTA and then change the configuration on FIRSTPORTB from OUTPUT to INPUT, the output value at FIRSTPORTA will be all zeros. You can, however, set the configuration on SECONDPORTx without affecting the value at FIRSTPORTA. For this reason, this function is usually called at the beginning of the program for each port requiring configuration.

## cbDIn() function

Reads a digital input port.

Note that for some port types, such as 8255 ports, if the port is configured for DIGITALOUT, this function will provide readback of the last output value.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions.

## Function Prototype

C/C++

```
int cbDIn(int BoardNum, int PortNum, unsigned short *DataValue)
```

Visual Basic

```
Function cbDIn(ByVal BoardNum&, ByVal PortNum&, DataValue%) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

Specifies which digital I/O port to read. Some hardware does allow readback of the state of the output using this function. Check the board-specific information in the *Universal Library User's Guide*.

*DataValue*

Digital input value returned here.

## Returns

- [Error code](#) or 0 if no errors
- DataValue - Digital input value returned here

## Notes

- 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.

Refer to the example programs and the board-specific information contained in the *Universal Library User's Guide* for clarification of valid PortNum values.



## cbDInScan() function

Multiple reads of digital input port of a high speed digital port on a board with a pacer clock, such as the CIO-PDMA16.

### Function Prototype

C/C++

```
int cbDInScan(int BoardNum, int PortNum, long Count, long *Rate, int MemHandle, int Options)
```

Visual Basic

```
Function cbDInScan(ByVal BoardNum&, ByVal PortNum&, ByVal Count&, Rate&, ByVal MemHandle&, ByVal  
Options&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

Specifies which digital I/O port to read (usually FIRSTPORTA or FIRSTPORTB).

*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.

*MemHandle*

Handle for Windows buffer to store data. This buffer must have been previously allocated with the [cbWinBufAlloc\(\)](#) function.

*Options*

Bit fields that control various options. Refer to the constants in the "[Options argument values](#)" section below.

### Returns

- [Error code](#) or 0 if no errors.
- Rate – actual sampling rate returned.
- MemHandle – digital input value returned via the allocated Windows buffer.

## Options argument values

BACKGROUND	<p>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 MemHandle.</p> <p>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 MemHandle will continue in the background. Use <a href="#">cbGetStatus()</a> with DIFUNCTION to check on the status of the background operation. Use <a href="#">cbStopBackground()</a> with DIFUNCTION to terminate the background process before it has completed.</p>
CONTINUOUS	<p>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 <a href="#">cbStopBackground()</a> with DIFUNCTION. Normally this option should be used in combination with BACKGROUND so that your program will regain control.</p>
EXTCLOCK	<p>If this option is used then transfers will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each transfer will be triggered on the appropriate edge of the trigger input signal (refer to board-specific information in the <i>Universal Library User's Guide</i>). When this option is used the Rate argument is ignored. The transfer rate is dependent on the trigger signal.</p>
EXTTRIGGER	<p>If this option is used, then the scan will not begin until the signal on the trigger input line meets the trigger criteria.</p>
HIGHRESRATE	<p>Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>Rate</i> argument above).</p>
WORDXFER	<p>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 the buffer. When WORDXFER is used, it is generally required to set PortNum to FIRSTPORTA.</p>

## Note

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

## cbDOut() function

Writes a byte to a digital output port.

Most configurable ports require configuration before writing. Check the board-specific information in the *Universal Library User's Guide* to determine if the port should be configured for your hardware. When configurable, use [cbDConfigPort\(\)](#) to configure a port for output.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O functions.

## Function Prototype

C/C++

```
int cbDOut(int BoardNum, int PortNum, unsigned short DataValue)
```

Visual Basic

```
Function cbDOut(ByVal BoardNum&, ByVal PortNum&, ByVal DataValue%) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

There are three general types of digital ports — ports that are programmable as input or output, ports that are fixed input or output, and ports for which each bit may be programmed as input or output. For the first of these types, set PortNum to FIRSTPORTA. For the latter two types, set PortNum to AUXPORT. Some boards have both types of digital ports (for example the DAS1600 Series). Set PortNum to either FIRSTPORTA or AUXPORT depending on the digital port you want to set.

*DataValue*

Digital value to be written.

## Returns

- [Error code](#) or 0 if no errors.

## Notes

- The size of the ports vary. 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.

Refer to the example programs and the board-specific information in the *Universal Library User's Guide* for clarification of valid PortNum values.

## cbDOutScan() function

Writes a series of bytes or words to the digital output port on a board with a pacer clock.

### Function Prototype

C/C++

```
int cbDOutScan(int BoardNum, int PortNum, long Count, long *Rate, int MemHandle, int Options)
```

Visual Basic

```
Function cbDOutScan(ByVal BoardNum&, ByVal PortNum&, ByVal Count&, Rate&, ByVal MemHandle&, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

Specifies which digital I/O port to write (usually FIRSTPORTA or FIRSTPORTB). The specified port must be configured as an output.

*Count*

The number of times to write digital output.

*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.

*MemHandle*

Handle for Windows buffer to store data. This buffer must have been previously allocated with the [cbWinBufAlloc\(\)](#) function.

*Options*

Bit fields that control various options. Refer to the constants in the "[Options argument values](#)" section below.

### Returns

- Error code or 0 if no errors.
- Rate – actual sampling rate returned.

## Options argument values

ADCCLOCK	Paces the data output operation using the ADC clock.
ADCCLOCKTRIG	Triggers a data output operation when the ADC clock starts.
BACKGROUND	<p>If the BACKGROUND option is not used, then the <code>cbDOutScan()</code> function will not return control to your program until all of the requested data has been output.</p> <p>When the BACKGROUND option is used, control returns immediately to the next line in your program and the transfer to the digital output port from <code>MemHandle</code> will continue in the background. Use <code>cbGetStatus()</code> with <code>DOFUNCTION</code> to check on the status of the background operation. Use <code>cbStopBackground()</code> with <code>DOFUNCTION</code> to terminate the background process before it has completed.</p>
CONTINUOUS	This option puts the function in an endless loop. Once it transfers the required number of bytes, it resets to the start of the buffer and begins again. The only way to stop this operation is with <code>cbStopBackground()</code> with <code>DOFUNCTION</code> . Normally this option should be used in combination with BACKGROUND so that your program will regain control.
EXTCLOCK	<p>When this option is used, transfers are controlled by the signal on the external clock input rather than by the internal pacer clock. Each transfer will be triggered on the appropriate edge of the clock input signal (refer to board-specific information contained in the <i>UL Users Guide</i>).</p> <p>When this option is used, the Rate argument is used for reference only. The transfer rate is dependent on the clock signal. An approximation of the external clock rate is used to determine the size of the packets to transfer from the board. Set the Rate argument to an approximate maximum value.</p>
NONSTREAMEDIO	<p>When this option is used, you can output non-streamed data to a specific DAC output channel.</p> <p>To load the data output buffer into the device's internal output FIFO, the aggregate size of the data output buffer must be <math>\leq</math> the size of the internal data output FIFO in the device. Once the sample data are transferred or downloaded to the device, the device is responsible for outputting the data. You can't make any changes to the output buffer once the output begins.</p> <p>With NONSTREAMEDIO mode, you do not have to periodically feed output data through the program to the device for the data output to continue. However, the size of the buffer is limited.</p>
WORDXFER	Normally this function writes a single (byte) port. If WORDXFER is specified then it will write two adjacent ports as the low and high byte of a single array element in the buffer. When WORDXFER is used, it is generally required to set <code>PortNum</code> to <code>FIRSTPORTA</code> .

## Notes

- BYTEXFER is the default option. Make sure you are using an array when your data is arranged in bytes. Use the WORDXFER option for word array transfers.
- NONSTREAMEDIO can only be used with the number of samples (`Count`) set equal to the size of the FIFO or less.
- Transfer Method may not be specified. DMA is used.

## cbErrHandling() function

Sets the error handling for all subsequent function calls. Most functions return error codes after each call. In addition, other error handling features are 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.

### Function Prototype

C/C++

```
int cbErrHandling(int ErrReporting, int ErrHandling)
```

Visual Basic

```
Function cbErrHandling(ByVal ErrReporting&, ByVal ErrHandling&) As Long
```

### Arguments

*ErrReporting*

This argument controls when the library will print error messages on the screen. The default is DONTPRINT. Set it to one of the constants in the "[ErrReporting argument values](#)" section below.

*ErrHandling*

This argument specifies what class of error will cause the program to halt. The default is DONTSTOP. Set it to one of the constants in the "[ErrHandling argument values](#)" section below.

### Returns

- Always returns 0.

### ErrReporting argument values

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 argument values

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. If you are running in an Integrated Development Environment (IDE), when errors occur the environment may be shut down along with the program. If your IDE behaves this way, (QuickBasic and VisualBasic do), then set ErrHandling to DONTSTOP. Refer to the " <a href="#">Error Codes</a> " topic for a complete list of error codes and their associated messages.

### Notes

- 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".

- **STOPALL** is not intended for 32-bit C console programs: Do not use the STOPALL option in 32-bit C console applications. Instead, use other methods to end the program, such as checking the return value of the function.

## cbGetErrMsg() function

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

### Function Prototype

C/C++

```
int cbGetErrMsg(int ErrCode, char ErrMsg[ERRSTRLEN])
```

Visual Basic

```
Function cbGetErrMsg(ByVal ErrCode&, ByVal ErrMsg$) As Long
```

### Arguments

*ErrCode*

The error code that is returned by any function in library.

*ErrMsg*

The error message is 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.

### Returns

- [Error code](#) or 0 if no errors
- ErrMsg - error message string is returned here

### Note

- See also [cbErrHandling\(\)](#) for an alternate method of handling errors.

## cbMemRead() function

Reads data from a memory board into an array.

### Function Prototype

C/C++

```
int cbMemRead(int BoardNum, unsigned short DataBuffer[], long FirstPoint, long Count)
```

Visual Basic

```
Function cbMemRead(ByVal BoardNum&, DataBuffer%, ByVal FirstPoint&, ByVal Count&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*DataBuffer*

Pointer to the data array.

*FirstPoint*

Index of first point to read, or FROMHERE. Use FirstPoint to specify the first point to read. For example, to read data sample numbers 200 through 250, set FirstPoint = 200 and Count = 50.

*Count*

Number of data points (words) to read.

### Returns

- [Error code](#) or 0 if no errors
- DataBuffer – data read from the memory board.

### Notes

- If you are going to read a large amount of data from the board in small chunks, 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.



## cbMemReadPretrig() function

Reads pre-trigger data collected with the [cbAPretrig\(\)](#) function from a memory board, and re-arranges the data in the correct order (pre-trigger data first, then post-trigger data). This function can only be used to retrieve data that was collected with the [cbAPretrig\(\)](#) function with EXTMEMORY set in the options argument. After each [cbAPretrig\(\)](#) 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

### Function Prototype

C/C++

```
int cbMemReadPretrig(int BoardNum, unsigned short DataBuffer[], long FirstPoint, long Count)
```

Visual Basic

```
Function cbMemReadPretrig(ByVal BoardNum&, DataBuffer%, ByVal FirstPoint&, ByVal Count&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*DataBuffer*

The pointer to the data array.

*FirstPoint*

Index of first point to read or FROMHERE. Use FirstPoint to specify the first point to read. For example, to read data sample numbers 200 through 250, set FirstPoint = 200 and Count = 50.

*Count*

Number of data samples (words) to read.

### Returns

- [Error code](#) or 0 if no errors
- DataBuffer – data read from the memory board.

### Notes

- When reading a large amount of data from the board in small chunks, set FirstPoint to FROMHERE to read each successive chunk. Using FROMHERE speeds up the operation of [cbMemReadPretrig\(\)](#) when working with large amounts of data.

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

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

- **DT Connect Conflicts:** The [cbMemReadPretrig\(\)](#) 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 [cbMemReadPretrig\(\)](#) until the [cbAInScan\(\)](#) has completed. If you do you will get a DTACTIVE error.

## cbMemReset() function

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.

### Function Prototype

C/C++

```
int cbMemReset(int BoardNum)
```

Visual Basic

```
Function cbMemReset(ByVal BoardNum&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- This function is used to reset the counter back to the start of the memory. Between successive calls to [cbAInScan\(\)](#), you should 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().

## cbMemSetDTMode() function

Sets the DT-Connect Mode of a Memory Board

### Function Prototype

C/C++

```
int cbMemSetDTMode(int BoardNum, int Mode)
```

Visual Basic

```
Function cbMemSetDTMode(ByVal BoardNum&, ByVal Mode&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*Mode*

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

### Returns

- [Error code](#) or 0 if no errors

### Notes

- This command only controls the direction of data transfer between the memory board and its parent board that is connected to it via a DT-Connect cable.
- If you are using the EXTMEMORY option, do not use cbMemSetDTMode(), as the memory board mode is already set with EXTMEMORY. Only use cbMemSetDTMode() when the parent board is not supported by the Universal Library.

## cbMemWrite() function

Writes data from an array to the memory card.

### Function Prototype

C/C++

```
int cbMemWrite(int BoardNum, unsigned short DataBuffer[], long FirstPoint, long Count)
```

Visual Basic

```
Function cbMemWrite(ByVal BoardNum&, DataBuffer%, ByVal FirstPoint&, ByVal Count&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*DataBuffer*

Pointer to the data array.

*FirstPoint*

Index of first point to write or FROMHERE. Use FirstPoint to specify the first point to write data to. For example, to write to location numbers 200 through 250, set FirstPoint = 200 and Count = 50.

*Count*

Number of data points (words) to write.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- To write a large amount of data to the board in small chunks, set FirstPoint to FROMHERE to write each successive chunk. For example, to write 300,000 points in 100,000 point chunks:

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

- **DT-Connect Conflicts:** The cbMemWrite() function cannot 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 cannot call cbMemWrite() until the cbAInScan() is complete. Doing so will generate a [DTACTIVE](#) error.

## cbDeclareRevision() function

New R3.3 ID

Initializes the Universal Library with the revision number of the library used to write your program. Must be the first Universal Library function to be called by your program.

### Function Prototype

C/C++

```
int cbDeclareRevision(float* RevNum);
```

Visual Basic

```
Function cbDeclareRevision(RevNum!) As Long
```

### Arguments

*RevNum*

Revision number of the Universal Library used to interpret function arguments.

Default setting: Any program using the 32-bit library and not containing this line of code will be defaulted to revision 5.4 argument assignments.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- As new revisions of the library are released, bugs from previous revisions are fixed and occasionally new functions are added. It is Measurement Computing's goal to preserve existing programs you have written and therefore to never change the order or number of arguments in a function. Sometimes this is not possible, as in the changes from revision 3.2 to 3.3. In revision 3.3, we added support for multiple background tasks, a feature that users have requested.

Allowing multiple background tasks required adding the argument BoardNum to several functions. Doing so would have meant that programs written for version 3.2 would not run with 3.3 if they called those functions. If not for the cbDeclareRevision() function, the programs would have had to be rewritten in each line where the affected functions are used, and the program recompiled.

The revision control function initializes the DLL so that the functions are interpreted according to the format of the revision you wrote and used to compile your program. This function is new in revision 3.3. To take advantage of it, the function must be added to your program and the program recompiled.

The function works by interpreting the UL function call from your program and filling in any arguments needed to run with the new revision. For example, the function cbAConvertData() had the argument BoardNum added in Revision 3.3.

- The two revisions of the function look like this:

Rev 3.2:

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

Rev 3.3:

```
int cbAConvertData(int BoardNum, long NumPoints, unsigned ADData[], int ChanTags[])
```

If your program has declared you are running code written for revision 3.2, and you call this function, the argument BoardNum is ignored. If you want the benefits afforded by BoardNum, you must rewrite your program with the new argument and declare revision 3.3 (or higher) in cbDeclareRevision().

If a revision less than 3.2 is declared, revision 3.2 is assumed.

## cbGetRevision() function

Gets the revision level of Universal Library DLL and the VXD.

### Function Prototype

C/C++

```
int cbGetRevision(float* DLLRevNum, float* VXDRevNum);
```

Visual Basic

```
Function cbGetRevision(DLLRevNum!, VXDRevNum!) As Long
```

### Arguments

*DLLRevNum*

Place holder for the revision number of Library DLL.

*VXDRevNum*

Place holder for the revision number of Library VXD.

### Returns

- [Error code](#), if revision levels of VXD and DLL are incompatible.

## cbFileAInScan() function

Scans a range of A/D channels and stores the samples in a disk file. `cbFileAInScan()` reads the specified number of A/D samples at the specified sampling rate from the specified range of A/D channels from the board. If the A/D board has programmable gain, it sets the gain to the specified range.

The collected data is returned to a file in binary format. Use [cbFileRead\(\)](#) to load data from that file into an array. See board-specific information to determine if this function is supported on your board.

## Function Prototype

C/C++

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

Visual Basic

```
Function cbFileAInScan(ByVal BoardNum&, ByVal LowChan&, ByVal HighChan&, ByVal Count&, Rate&, ByVal Range&, ByVal FileName$, ByVal Options&) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. The specified board must have an A/D. BoardNum may be 0 to 99.

*LowChan*

First A/D channel of scan

*HighChan*

Last A/D channel of scan

The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured (for example, eight channels 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, the number of samples collected per channel is equal to  $\text{Count} / (\text{HighChan} - \text{LowChan} + 1)$ .

*Rate*

Sample rate in samples per second (Hz) per channel. The maximum sampling rate depends on the A/D board that is being used (see Rate explanation in [cbAInScan\(\)](#)).

*Range*

If the selected A/D board does not have a programmable range feature, this argument is ignored. Otherwise set the Range argument to any range that is supported by the selected A/D board. Refer to board-specific information for a list of the [supported A/D ranges](#) of each board.

*FileName*

The name of the file in which to store the data. If the file doesn't exist, it will be created.

*Options*

Bit fields that control various options. Refer to the constants in the [Options argument values](#) section below.

## Returns

- [Error code](#) or 0 if no errors.
- Rate – the actual sampling rate

## Options argument values

EXTCLOCK	If this option is used, conversions are controlled by the signal on the trigger input line rather than by the internal pacer clock. Each conversion is triggered on the appropriate edge of the trigger input signal (refer to board-specific information in the Universal Library User's Guide). Additionally, the Rate argument is ignored. The sampling rate is dependent on the trigger signal.
EXTTRIGGER	<p>If this option is specified, the sampling does not begin until the trigger condition is met.</p> <p>On many boards, this trigger condition is programmable (refer to the <a href="#">cbSetTrigger()</a> function and board-specific information for details) and can be programmed for rising or falling edge or an analog level.</p> <p>On other boards, only <i>polled gate</i> triggering is supported. Assuming active high operation, data acquisition commences immediately if the trigger input is high. If the trigger input is low, acquisition is held off until it goes high. Acquisition continues until NumPoints&amp; samples are taken, regardless of the state of the trigger input. For polled gate triggering, this option is most useful if the signal is a pulse with a very low duty cycle (trigger signal in TTL low state most of the time) to hold off triggering until the pulse occurs.</p>
DTCONNECT	Samples are sent to the DT-Connect port if the board is equipped with one.

## Notes

- [OVERRUN Error](#) - (Error code 29) 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 returned from [cbFileGetInfo\(\)](#) in TotalCount is the number of points that were successfully collected.

## Important!

In order to understand the functions, read the board-specific information contained in the *Universal Library User's Guide*. We also urge you to examine and run one or more of the example programs supplied prior to attempting any programming of your own. Following this advice may save you hours of frustration, and wasted time.

This note, which appears elsewhere, is especially applicable to this function. Now is the time to read the board-specific information for your board. We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.



## cbFileGetInfo() function

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. Refer to board-specific information in the *Universal Library User's Guide* to determine if your board supports [cbFileAInScan\(\)](#) and/or [cbFilePretrig\(\)](#).

### Function Prototype

C/C++

```
int cbFileGetInfo(char *FileName, short *LowChan, short *HighChan, long *PretrigCount, long *TotalCount, long *Rate, int *Range)
```

Visual Basic

```
Function cbFileGetInfo(ByVal FileName$, LowChan%, HighChan%, PretrigCount&, TotalCount&, Rate&, Range&) As Long
```

### Arguments

*FileName*

Name of the streamer file.

*LowChan*

Variable to return LowChan to.

*HighChan*

Variable to return HighChan to.

*PretrigCount*

Variable to return PretrigCount to.

*TotalCount*

Variable to return TotalCount 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 the scan.
- HighChan – high A/D channel of the 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.

## cbFilePretrig() function

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. See board specific info to determine if this function is supported by your board.

## Function Prototype

C/C++

```
int cbFilePretrig(int BoardNum, int LowChan, int HighChan, long *PretrigCount, long *TotalCount, long *Rate, int Range, char *FileName, unsigned Options)
```

Visual Basic

```
Function cbFilePretrig(ByVal BoardNum&, ByVal LowChan&, ByVal HighChan&, PretrigCount&, TotalCount&, Rate&, ByVal Range&, ByVal FileName$, ByVal Options&) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. The specified board must have an A/D and pretrigger capability. BoardNum may be 0 to 99.

*LowChan*

First A/D channel of the scan.

*HighChan*

Last A/D channel of the scan.

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 Refer to board-specific information for the maximum number of channels allowed in differential and single ended modes.

*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 than 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 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.

*Rate*

Sample rate in samples per second (Hz) per channel. The maximum sampling rate depends on the A/D board that is being used. 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 (10 kHz) will result in the A/D converter rate of 40 kHz: 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. This argument also returns the value of the actual set. This may be different from the requested rate because of pacer limitations.

*Range*

If the selected A/D board does not have a programmable range feature, this argument is ignored. Otherwise, set the Range argument to any range that is supported by the selected A/D board. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

*FileName*

The name of the file in which to store the data. If the file doesn't exist, it will be created.

*Options*

Bit fields that control various options. Refer to the constants in the [Options argument values](#) section below.

## Returns

- [Error code](#) or 0 if no errors
- PretrigCount – actual number of pre-trigger samples collected.

- TotalCount – actual number of samples collected.
- Rate – the actual sampling rate.

## Options argument values

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 appropriate edge of the trigger input signal (refer to board-specific information in the <i>Universal Library User's Guide</i> ). 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.

## Notes

- [OVERRUN Error](#) - (Error code 29): 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 TotalCount will be the number of points that were successfully collected.

## cbFileRead() function

Reads data from a streamer file. When [cbFileAInScan\(\)](#) or [cbFilePretrig\(\)](#) fills the streamer file, this function returns the content of that file. Refer to information on your board in the *Universal Library User's Guide* to determine if your board supports [cbFileAInScan\(\)](#) and/or [cbFilePretrig\(\)](#).

## Function Prototype

C/C++

```
int cbFileRead(char *FileName, long FirstPoint, long *NumPoints, int *DataBuffer)
```

Visual Basic

```
Function cbFileRead(ByVal FileName$, ByVal FirstPoint&, NumPoints&, DataBuffer%) As Long
```

## Arguments

*FileName*

Name of the streamer file.

*FirstPoint*

Index of the first point to read.

*NumPoints*

Number of points to read from the file.

*DataBuffer*

Pointer to the data buffer that data will be read into.

## Returns

- [Error code](#) or 0 if no errors
  - *DataBuffer* – data read from file.
  - *NumPoints* – number of points actually read.
- NumPoints* may be less than the requested number of points if an error occurs.

## Notes

- Data format – The data is returned as 16 bits. The 16 bits may represent 12 bits of analog, 12 bits of analog plus 4 bits of channel, or 16 bits of analog. Use [cbAConvertData\(\)](#) to correctly load the data into an array.
- Loading portions of files – The file may contain much more data than can fit in *DataBuffer*. In those cases, use *NumPoints* 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.

## cbDaqInScan() function

Scans analog, digital, counter, and temperature input channels synchronously, and stores the samples in an array. This function only works with boards that support synchronous input.

### Function Prototype

C/C++

```
int cbDaqInScan(int BoardNum, short ChanArray[], short ChanTypeArray[], short GainArray[], int ChanCount, long* Rate, long *PretrigCount, long *TotalCount, int MemHandle, int Options);
```

Visual Basic

```
Function cbDaqInScan(ByVal BoardNum&, ChanArray%, ChanTypeArray%, GainArray%, ByVal ChanCount&, CBRate&, PretrigCount&, CBCount&, ByVal MemHandle&, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal. The specified board must support synchronous input.

*ChanArray*

Array containing channel values. Valid channel values are analog input channels, digital ports, counter input channels, and temperature input channels of the device.

*ChanTypeArray*

Array containing channel types. Each element of this array defines the type of the corresponding element in the ChanArray. Set to one of the constants in the "[ChanTypeArray argument values](#)" section below.

*GainArray*

Array containing A/D range codes. If the corresponding element in the ChanArray is not an analog input channel, the range code for this channel is ignored.

*ChanCount*

Number of elements in each of the three arrays - ChanArray, ChanTypeArray, and GainArray.

*Rate*

The sample rate at which samples are acquired, in samples per second per channel.

Rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*PretrigCount*

Sets the number of pre-trigger samples to collect. Specifies the number of samples to collect before the trigger occurs. This function won't run in pre-trigger mode if PreTrigCount is set to zero. PreTrigCount is ignored if the EXTTRIGGER option is not specified.

PreTrigCount also returns the value of the actual pre-trigger count set, which may be different from the requested pre-trigger count, because pre-trigger count must be a multiple of ChanCount.

PreTrigCount must be evenly divisible by the number of channels being scanned (ChanCount). If it is not, this function adjusts the number (up) to the next valid value, and returns that value to the PreTrigCount argument.

*TotalCount*

Total number of samples to collect. Specifies the total number of samples to collect and store in the buffer. TotalCount must be greater than PreTrigCount.

TotalCount also returns the value of the actual total count set, which may be different from the requested total count because total count must be a multiple of ChanCount.

TotalCount must be evenly divisible by the number of channels being scanned (ChanCount). If it is not, this function adjusts the number (down) to the next valid value, and returns that value to the TotalCount argument.

*MemHandle*

Handle for the Windows buffer to store data. This buffer must have been previously allocated with the [cbWinBufAlloc\(\)](#) function.

*Options*

Bit fields that control various options. This field may contain any combination of non-contradictory choices in the "[Options argument values](#)" section below.

### Returns

- [Error code](#) or 0 if no errors
- Rate – Actual sampling rate used.
- PreTrigCount – Actual pre-trigger count used.
- TotalCount – Actual total count used.
- MemHandle – Collected data returned via the Windows buffer.

## ChanTypeArray argument values

ANALOG	Analog input channel.
DIGITAL8	8-bit digital input port.
DIGITAL16	16-bit digital input port. (FIRSTPORTA only)
CTR16	16-bit counter.
CTR32LOW	Lower 16-bits of a 32-bit counter.
CTR32HIGH	Upper 16-bits of a 32-bit counter.
CJC	CJC channel.
TC	Thermocouple channel.  The <a href="#">cbGetTCValues()</a> function can be used to convert raw thermocouple data to data on a temperature scale (CELSIUS, FAHRENHEIT, or KELVIN). <b>Note:</b> If at least one TC channel is listed in the channel array, and averaging is enabled for that channel, the averaging will be applied to all of the channels listed in the channel array.
SETPOINTSTATUS	The setpoint status register. This is a bit field indicating the state of each of the setpoints. A "1" indicates that the setpoint criteria has been met.

## ChanTypeArray flag values

SETPOINT_ENABLE	Enables a setpoint. When this option is specified, it must be OR'ed with the ChanTypeArray argument values.  You set the setpoint criteria with the <a href="#">cbDagSetSetpoints()</a> function. The number of channels set with the SETPOINT_ENABLE flag must match the number of setpoints set by the <a href="#">cbDagSetSetpoints()</a> function's SetpointCount argument.
-----------------	---

## Options argument values

BACKGROUND	When the BACKGROUND option is used, control returns immediately to the next line in your program, and the data collection into the buffer continues in the background. If the BACKGROUND option is not used, the <a href="#">cbDagInScan()</a> function does not return control to your program until all of the requested data has been collected and returned to the buffer.  Use <a href="#">cbGetStatus()</a> with DAQIFUNCTION to check on the status of the background operation. Use <a href="#">cbStopBackground()</a> with DAQIFUNCTION to terminate the background process before it has completed. Execute <a href="#">cbStopBackground()</a> after normal termination of all background functions, in order to clear variables and flags.
CONTINUOUS	This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of the buffer and begins again. The only way to stop this operation is by using <a href="#">cbStopBackground()</a> with DAQIFUNCTION. Normally this option should be used in combination with BACKGROUND so that your program will regain control.
EXTCLOCK	If this option is used, conversions will be controlled by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal. When this option is used the Rate argument is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to a transfer mode that will allow the maximum conversion rate to be attained unless otherwise specified.
EXTTRIGGER	If this option is specified, the sampling will not begin until the trigger condition is met (refer to the <a href="#">cbDagSetTrigger()</a> function).
HIGHRESRATE	Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>Rate</i> argument above).

# cbDaqOutScan() function

Outputs values synchronously to analog output channels and digital output ports. This function only works with boards that support synchronous output.

## Function Prototype

C/C++

```
int cbDaqOutScan(int BoardNum, short ChanArray[], short ChanTypeArray[], short GainArray[], int ChanCount, long* Rate, long Count, int MemHandle, int Options);
```

Visual Basic

```
Function cbDaqOutScan(ByVal BoardNum&, ChanArray%, ChanTypeArray%, GainArray%, ByVal ChanCount&, CBRate&, ByVal CBCount&, ByVal MemHandle&, ByVal Options&) As Long
```

## Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal. The specified board must support synchronous output.

*ChanArray*

Array containing channel values. Valid channel values are analog output channels and digital ports.

*ChanTypeArray*

Array containing channel types. Each element of this array defines the type of the corresponding element in the ChanArray. Set to one of the constants in the "[ChanTypeArray argument values](#)" section below.

*GainArray*

Array containing D/A range codes. If the corresponding element in the ChanArray is not an analog output channel, the range code for this channel is ignored. If the board does not have programmable gain, this parameter is ignored, and therefore can be set to null.

*ChanCount*

Number of elements in each of the three arrays - ChanArray, ChanTypeArray, and GainArray.

*Rate*

Sample rate in scans per second. Rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*Count*

Sets the total number of values to output. Count also returns the value of the actual count set, which may be different from the requested total count because count must be a multiple of the channel count.

*MemHandle*

Handle for the Windows buffer from which data will be output. This buffer must have been previously allocated with the [cbWinBufAlloc\(\)](#) function and data values loaded (for example using [cbWinArrayToBuf\(\)](#)).

*Options*

Bit fields that control various options. This field may contain any combination of non-contradictory choices in the "[Options argument values](#)" section below.

## Returns

- [Error code](#) or 0 if no errors
- Rate – Actual sampling rate used.

## ChanTypeArray argument values

ANALOG	Analog output channel.
DIGITAL16	16-bit digital output port. (FIRSTPORTA only)

## Options argument values

ADCCLOCK	When this option is used, the data output operation will be paced by the ADC clock.
ADCCLOCKTRIG	If this option is used, the data output operation will be triggered upon the start of the ADC clock.
BACKGROUND	When this option is used, the output operations will begin running in the background and control will immediately return to the next line of your program. Use <a href="#">cbGetStatus()</a> with the DAQOFUNCTION option to check the status of background operation. Use the <a href="#">cbStopBackground()</a> function with the DAQOFUNCTION option to terminate background operations before they are completed. Execute <a href="#">cbStopBackground()</a> with DAQOFUNCTION after normal termination of all background functions in order to clear variables and flags.
CONTINUOUS	This option puts the function in an endless loop. Once it outputs the specified number (Count) of output values, it resets to the start of the buffer and begins again. The only way to stop this operation is by calling <a href="#">cbStopBackground()</a> with DAQOFUNCTION. This option should only be used in combination with BACKGROUND so that your program can regain control.
EXTCLOCK	<p>If this option is used, conversions will be paced by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal.</p> <p>When this option is used, the Rate argument is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to transfer types that allow the maximum conversion rate to be attained unless otherwise specified.</p>
NONSTREAMEDIO	<p>This option allows non-streamed data output to be generated to a specified output channel.</p> <p>In this mode, the aggregate size of data output buffer must be less than or equal to the size of the internal data output FIFO on the Measurement Computing device. This allows the data output buffer to be loaded into the device's internal output FIFO.</p> <p>Once the sample updates are transferred (or downloaded) to the device, the device is responsible for outputting the data. While the size is limited, and the output buffer cannot be changed once the output is started, this mode has the advantage being able to continue data output without having to periodically feed output data through the program to the device.</p>



## cbDaqSetSetpoints() function

Configures up to 16 detection setpoints associated with the input channels within a scan group. This function only works with boards that support synchronous input.

### Function Prototype

C/C++

```
int cbDaqSetSetpoints(int BoardNum, float *LimitAArray, float *LimitBArray, float *reserved, int
*SetpointFlagsArray, int *SetpointOutputArray, float *Output1Array, float *Output2Array, float
*OutputMask1Array, float *OutputMask2Array, int SetpointCount);
```

Visual Basic

```
Function cbDaqSetSetpoints(ByVal BoardNum&, LimitAArray!, LimitBArray!, Reserved!, SetpointFlagsArray&,
SetpointOutputArray&, Output1Array!, Output2Array!, OutputMask1Array!, OutputMask2Array!, ByVal
SetpointCount&) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal. The specified board must support synchronous input.

*LimitAArray*

Array containing the limit A values for the input channels used for the setpoint. Limit A specifies a value used to determine if the setpoint criteria are met.

*LimitBArray*

Array containing the limit B values for the input channels used for the setpoint. Limit B specifies a value used to determine if the setpoint criteria are met.

*Reserved*

Reserved for future use.

*SetpointFlagsArray*

Array containing the setpoint flags. Set to one of the constants in the [SetpointFlagsArray argument values](#) section below.

*SetpointOutputArray*

Array containing output sources. Set to one of the constants in the [SetpointOutputArray argument values](#) section below.

*Output1Array*

Array containing the values for the output channels used for the setpoint.

*Output2Array*

Array containing the values for the output channels used for the setpoint.

*OutputMask1Array*

Array containing the output masks for output value 1 (for FIRSTPORTC only).

*OutputMask2Array*

Array containing the output masks for output value 2 (for FIRSTPORTC only).

*SetpointCount*

Number of setpoints to configure (0 to 16). Set to 0 to disable the setpoints.

### Returns

- [Error code](#) or 0 if no errors

**SetpointFlagsArray argument values**

Flag	Description
SF_EQUAL_LIMITA	Setpoint criteria: The input channel = limit A.
SF_LESSTHAN_LIMITA	Setpoint criteria: The input channel < limit A.
SF_GREATERTHAN_LIMITB	Setpoint criteria: The input channel > limit B.
SF_INSIDE_LIMITS	Setpoint criteria: The input channel > limit A and < limit B.
SF_OUTSIDE_LIMITS	Setpoint criteria: The input channel < limit A and > limit B.
SF_HYSTERESIS	Setpoint criteria: If the input channel > limit A then output value 1. If the input channel < limit B then output value 2.
SF_UPDATEON_TRUEONLY	If the criteria is met then output value 1.
SF_UPDATEON_TRUEANDFALSE	If the criteria is met then output value 1, else output value 2.

**SetpointOutputArray argument values**

Output source	Description
SO_NONE	Perform no outputs.
SO_FIRSTPORTC	Output to FIRSTPORTC when the criteria is met.
SO_DIGITALPORT	Output to digital port when the criteria is met.
SO_DAC0	Output to DAC0 when the criteria is met. You must have a device with DAC0.
SO_DAC1	Output to DAC1 when the criteria is met. You must have a device with DAC1.
SO_DAC2	Output to DAC2 when the criteria is met. You must have a device with DAC2.
SO_DAC3	Output to DAC3 when the criteria is met. You must have a device with DAC3.
SO_TMR0	Output to timer 0 when the criteria is met.
SO_TMR1	Output to timer 1 when the criteria is met.

## cbDaqSetTrigger() function

Selects the trigger source and sets up its parameters. This trigger is used to initiate or terminate an acquisition using the [cbDaqInScan\(\)](#) function if the EXTTRIGGER option is selected. This function only works with boards that support synchronous output.

### Function Prototype

C/C++

```
int cbDaqSetTrigger(int BoardNum, int TrigSource, int TrigSense, int TrigChan, int ChanType, int Gain, float Level, float Variance, int TrigEvent);
```

Visual Basic

```
Function cbDaqSetTrigger(ByVal BoardNum&, ByVal TrigSource&, ByVal TrigSense&, ByVal TrigChan&, ByVal ChanType&, ByVal Gain&, ByVal Level!, ByVal Variance!, ByVal TrigEvent&) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal. The board must support synchronous output.

*TrigSource*

Specifies the type of triggering based on the external trigger source. Set to one of the constants specified in [TrigSource argument values](#) section below.

*TrigSense*

Specifies the trigger sensitivity. The trigger sensitivity normally defines the way in which a trigger event is detected based upon the characteristics of the trigger input signal. However, it often defines the way in which the trigger input signal(s) should be compared to the trigger level parameter value. Set to one of the constants specified in [TrigSense argument values](#) section below.

*TrigChan*

Specifies the trigger channel. The trigger channel must be a configured channel in the channel array (refer to [cbDaqInScan\(\)](#)).

*ChanType*

Specifies the channel type and should match the channel type setting for the trigger channel configured using the [cbDaqInScan\(\)](#) function.

*Gain*

Specifies the trigger channel gain code. If the device has programmable gain, this argument should match the gain code setting when the channel is configured using the [cbDaqInScan\(\)](#) function. The Gain parameter is ignored if TrigChan is not an analog channel.

*Level*

A single precision floating point value which represents, in engineering units, the level at or around which the trigger event should be detected.

This option is used for trigger types that depend on an input channel comparison to detect the start trigger or stop trigger event.

The actual level at which the trigger event is detected depends upon trigger sensing and variability. Refer to the [Trigger Levels](#) section below for more information.

*Variance*

A single-precision floating point value which represents, in engineering units, the amount that the trigger event can vary from the Level parameter.

While the TrigSense parameter indicates the direction of the input signal relative to the Level parameter, the Variance parameter specifies the degree to which the input signal can vary relative to the Level parameter.

*TrigEvent*

Specifies the trigger event type. Valid values indicate either a start trigger event (START\_EVENT) or a stop trigger event (STOP\_EVENT).

**START\_EVENT:** The start trigger event defines the conditions under which post-trigger acquisition data collection should be initiated or triggered. The start trigger event can vary in complexity from starting immediately, to starting on complex channel value definitions.

**STOP\_EVENT:** The stop trigger event signals the current data acquisition process to terminate. The stop event can be as simple as that of a scan count, or as complex as involving a channel value level condition.

### Returns

- [Error code](#) or 0 if no errors

## TrigSource argument values

TRIG_IMMEDIATE	Start trigger event only. Acquisition begins immediately upon invocation the <a href="#">cbDagInScan()</a> function. No pre-trigger data acquisition is possible with this trigger type.
TRIG_EXTTTL	Start trigger event only. Acquisition begins on the selectable edge of an external TTL signal. No pre-trigger data acquisition is possible with this trigger type.
TRIG_ANALOGHW	Start trigger event only. Data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.) TrigChan must be defined as the first channel in the channel scan group. No pre-trigger data acquisition is possible with this trigger type.
TRIG_ANALOGSW	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)
TRIG_DIGPATTERN	Post-trigger data acquisition beings upon receiving a specified digital pattern on the specified digital port.
TRIG_COUNTER	Post-trigger data acquisition begins upon detection of specified counter criteria.
TRIG_SCANCOUNT	Stop trigger event only. Stops collecting post-trigger data when the specified number of post-trigger scans are completed.

## TrigSense argument values

RISING_EDGE	Triggers when the signal goes from low to high (TTL trigger) or rises through a specified level (hardware analog, software analog, and counter).
FALLING_EDGE	Triggers when the signal goes from high to low (TTL trigger) or falls through a specified level (hardware analog, software analog, and counter).
ABOVE_LEVEL	Triggers when the signal is above a specified level (hardware analog, software analog, counter, and digital pattern).
BELOW_LEVEL	Triggers when the signal is below a specified level (hardware analog, software analog, counter, and digital pattern).
EQ_LEVEL	Triggers when the signal equals a specified level (hardware analog, software analog, counter, and digital pattern).
NE_LEVEL	Triggers when the signal does not equal a specified level (hardware analog, software analog, counter, and digital pattern).

## Trigger levels

The actual level at which the trigger event is detected depends upon trigger sensing and variability. The various ranges of possible values for the Level parameter based on the trigger source are listed here:

- TRIG\_ANALOG\_HW: The voltage used to define the trigger level. Trigger detection is performed in hardware.
- TRIG\_ANALOG\_SW: The voltage used to define the trigger level. Trigger detection is performed in software.
- TRIG\_DIGPATTERN: Sets the bit pattern for the digital channel trigger. Choices are:  
0.0 (no bits set): 255.0 (all bits set) for 8-bit digital ports.  
0.0 (no bits set): 65,535.0 (all bits set) for 16-bit digital ports.
- TRIG\_COUNTER: Selects either Pulse or Totalize counter values (0.0 65,535).
- TRIG\_IMMEDIATE: Ignored
- TRIG\_SCANCOUNT: Ignored

## Trigger start and stop criteria

The table below lists the trigger start and stop criteria based on the selected trigger type and sensitivity.

Trigger Start/Stop Source (TrigSource)	Trigger Sensitivity (TrigSense)	Trigger Start/Stop Criteria
TRIG_ANALOGHW (Start trigger event only)	RISING_EDGE	Triggers when the signal value $< (\text{Level} - \text{Variance})$ . Then, the signal value $> \text{Level}$ .
	FALLING_EDGE	Triggers when the signal value $> (\text{Level} + \text{Variance})$ . Then, the signal value $< \text{Level}$ .
	ABOVE_LEVEL	Triggers when the signal value $> (\text{Level})$ .
	BELOW_LEVEL	Triggers when the signal value $< (\text{Level})$ .
TRIG_ANALOGSW	RISING_EDGE	Triggers/stops when the signal value $< (\text{Level} - \text{Variance})$ . Then, the signal value $> \text{Level}$ .
	FALLING_EDGE	Triggers/stops when the signal value $> (\text{Level} + \text{Variance})$ . Then, the signal value $< \text{Level}$ .
	ABOVE_LEVEL	Triggers/stops when the signal value $> (\text{Level})$ .
	BELOW_LEVEL	Triggers/stops when the signal value $< (\text{Level})$ .
	EQ_LEVEL	Triggers/stops when the $(\text{Level} - \text{Variance}) < \text{signal value} < (\text{Level} + \text{Variance})$ .
	NE_LEVEL	Triggers/stops when the signal value $< (\text{Level} - \text{Variance})$ OR when the signal value $> (\text{Level} + \text{Variance})$ .
TRIG_DIGPATTERN	ABOVE_LEVEL	Triggers/stops when the (digital port value AND (bitwise) Variance) $> (\text{Level AND (bitwise) Variance})$ .
	BELOW_LEVEL	Triggers/stops when the (digital port value AND (bitwise) Variance) $< (\text{Level AND (bitwise) Variance})$ .
	EQ_LEVEL	Triggers/stops when the (digital port value AND (bitwise) Variance) $= (\text{Level AND (bitwise) Variance})$ .
	NE_LEVEL	Triggers/stops when the (digital port value AND (bitwise) Variance) $\neq (\text{Level AND (bitwise) Variance})$ .
TRIG_COUNTER	RISING_EDGE	Triggers/stops when the counter channel $< (\text{Level Level} - \text{Variance})$ . Then, the counter channel $> \text{Level}$ .
	FALLING_EDGE	Triggers/stops when the counter channel $> (\text{Level} + \text{Variance})$ . Then, the counter channel $< \text{Level}$ .
	ABOVE_LEVEL	Triggers/stops when the counter channel $> (\text{Level} - \text{Variance})$ .
	BELOW_LEVEL	Triggers/stops when the counter channel $< (\text{Level} + \text{Variance})$ .
	EQ_LEVEL	Triggers/stops when $(\text{Level} - \text{Variance}) < \text{counter channel} < (\text{Level} + \text{Variance})$ .
	NE_LEVEL	Triggers/stops when the counter channel $< (\text{Level} - \text{Variance})$ OR when the counter channel $> (\text{Level} + \text{Variance})$ .

# cbTIn() function

## Changed R3.3 ID

Reads an analog input channel, linearizes it according to the selected temperature sensor type, if required, and returns the temperature in units determined by the Scale argument. The CJC channel, the gain, and sensor type, are read from the InstaCal configuration file, and should be set by running InstaCal.

## Function Prototype

```
C/C++
int cbTIn(int BoardNum, int Chan, int Scale, float *TempVal, int Options)

Visual Basic
Function cbTIn(ByVal BoardNum&, ByVal Chan&, ByVal Scale&, TempVal!, ByVal Options&) As Long
```

## Arguments

- BoardNum*  
The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.
- Chan*  
Input channel to read.
- Scale*  
Specifies the temperature scale that the input will be converted to. Choices are CELSIUS, FAHRENHEIT, KELVIN, VOLTS, and NOSCALE.
- TempVal*  
The temperature in units determined by the Scale argument is returned here.
- Options*  
Bit fields that control various options. Refer to the constants in the "[Options argument values](#)" section below.

## Returns

- [Error code](#) or 0 if no errors
- TempVal – Temperature returned here

## Options argument values

FILTER	When selected, a smoothing function is applied to temperature readings, very much like the electrical smoothing inherent in all hand held temperature sensor instruments. This is the default. When selected, 10 samples are read from the specified channel and averaged. The average is the reading returned. Averaging removes normally distributed signal line noise.
NOFILTER	If you use the NOFILTER option then the readings will not be smoothed and you will see a scattering of readings around a mean.

Refer to the board-specific information in the *Universal Library User's Guide* to determine if your hardware supports these options.

## Notes

### Scale options

- Specify the *NOSCALE* option to retrieve raw data from the device. When NOSCALE is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.
- Specify the *VOLTS* option to read the voltage input of a thermocouple.

Refer to board-specific information in the *Universal Library User's Guide* to determine if your hardware supports these options.

### Using CIO-EXP boards

For CIO-EXP boards, the channel number is calculated using the following formula, where:

ADChan is the A/D channel that is connected to the multiplexer.

MuxChan is a number ranging from 0 to 15 that specifies the channel number on a particular bank of the multiplexer board.

$$\text{Chan} = (\text{ADChan} \times 16) + (16 + \text{MuxChan})$$

For example, you have an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0. (Remember that DAS08 channels are numbered 0, 1, 2, 3, 4, 5, 6 and 7). If you connect a thermocouple to channel 5 of the EXP16, the value for Chan would be  $(0 \times 16) + (16 + 5) = 0 + 21 = 21$ .

## CJC channel

The CJC channel is set in the InstaCal installation and configuration program. If you have multiple EXP boards, the Universal Library will apply the CJC reading to the linearization formula in the following manner:

- 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.
- If you 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: You have four CIO-EXP16 boards connected to a CIO-DAS08 on channel 0, 1, 2 and 3. You choose CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7, AND, you leave the CIO-EXP16's 2, 3 and 4 CJC channels to NOT SET. Result: The CIO-EXP boards will all use the CJC reading from CIO-EXP16 #1, connected to channel 7 for linearization. *It is important to keep the CIO-EXP boards in the same case and out of any breezes to ensure a clean CJC reading.*

## Specific Errors

If an [OUTOFRANGE](#) or [OPENCONNECTION](#) error occurs, the value returned in TempVal is -9999.0. If a [NOTREADY](#) error occurs, the value returned in TempVal is -9000.

## Important!

For an EXP board connected to an A/D board that does not have programmable gain (DAS08, DAS16, DAS16F), the A/D board range is read from the configuration file (cb.cfg). In most cases, set hardware-selectable ranges to  $\pm 5$  V for thermocouples, and to 0 to 10 V for RTDs. Refer to the board-specific information in the *Universal Library User's Guide* or in the user manual for your board. If the board has programmable gains, the cbTIn() function sets the appropriate A/D range.

# cbTInScan() function

## Changed R3.3 ID

Reads a range of channels from an analog input board, linearizes them according to temperature sensor type, if required, and returns the temperatures to an array in units determined by the Scale argument. The CJC channel, the gain, and temperature sensor type are read from the configuration file. Use the InstaCal configuration program to change any of these options.

## Function Prototype

C/C++

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

Visual Basic

```
Function cbTInScan(ByVal BoardNum&, ByVal LowChan&, ByVal HighChan&, ByVal Scale&, DataBuffer!, ByVal Options&) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*LowChan*

Low channel of the scan.

*HighChan*

High channel of the scan.

*Scale*

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

*DataBuffer*

The temperature is returned in units determined by the Scale argument. 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.

*Options*

Bit fields that control various options. Refer to the constants in the [Options argument values](#) section below.

## Returns

- [Error code](#) or 0 if no errors
- DataBuffer[] - Temperature values in units determined by the Scale argument are returned here for each channel in the scan.

## Options argument values

FILTER	When selected, a smoothing function is applied to temperature readings, very much like the electrical smoothing inherent in all hand held temperature sensor instruments. This is the default. When selected, 10 samples are read from the specified channel and averaged. The average is the reading returned. Averaging removes normally distributed signal line noise.
NOFILTER	If you use the NOFILTER option then the readings will not be smoothed and you will see a scattering of readings around a mean.

Refer to the board-specific information in the *Universal Library User's Guide* to determine if your hardware supports these options.

## Notes

### Scale options

- Specify the *NoScale* option to retrieve raw data from the device. When NoScale is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.
- Specify the *Volts* option to read the voltage input of a thermocouple.

Refer to board-specific information in the *Universal Library User's Guide* to determine if your hardware supports these options.



## Using CIO-EXP boards

For CIO-EXP boards, the channel number is calculated using the following formula:

$$\text{Chan} = (\text{ADChan} \times 16) + (16 + \text{MuxChan})$$

where:

ADChan is the A/D channel that is connected to the multiplexer.

MuxChan is a number ranging from 0 to 15 that specifies the channel number on a particular bank of the multiplexer board.

For example, you have an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0. (Remember that DAS08 channels are numbered 0, 1, 2, 3, 4, 5, 6 and 7). If you connect a thermocouple to channel 5 of the EXP16, the value for Chan would be  $(0 \times 16) + (16 + 5) = 0 + 21 = 21$ .

## CJC channel

The CJC channel is set in the InstaCal installation and configuration program. If you have multiple EXP boards, the Universal Library will apply the CJC reading to the linearization formula in the following manner:

- 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.
- If you 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: You have four CIO-EXP16 boards connected to a CIO-DAS08 on channel 0, 1, 2 and 3. You choose CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7, AND, you leave the CIO-EXP16's 2, 3 and 4 CJC channels to NOT SET. Result: The 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.

## Specific Errors

For most boards, if an [OUTOFRANGE](#) or [OPENCONNECTION](#) error occurs, the value in the array element associated with the channel causing the error returned will be -9999.0.

## Important!

For an EXP board connected to an A/D board that does not have programmable gain (DAS08, DAS16, DAS16F), the A/D board range is read from the configuration file (cb.cfg). In most cases, set hardware-selectable ranges to  $\pm 5$  V for thermocouples, and to 0 to 10 V for RTDs. Refer to the board-specific information in the *Universal Library User's Guide* or in the user manual for your board. If the board has programmable gains, the cbTIn() function sets the appropriate A/D range.

## cbWinBufAlloc() function

Allocates a Windows global memory buffer which can be used with the scan functions, and returns a memory handle for it.

Most devices return data in a 16-bit format. For these devices, the buffer can be created using `cbWinBufAlloc()`. Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using [cbWinBufAlloc32\(\)](#) or [cbWinBufAlloc64\(\)](#). See hardware-specific information to determine the type of buffer needed. If not specifically mentioned, use `cbWinBufAlloc()`.

### Function Prototype

C/C++

```
HGLOBAL cbWinBufAlloc(long NumPoints)
```

Visual Basic

```
Function cbWinBufAlloc(ByVal NumPoints&) As Long
```

### Arguments

*NumPoints*

The size of the buffer to allocate. Specifies how many data points (16-bit integers, NOT bytes) can be stored in the buffer.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other functions in the library, this function does not return an error code. It returns a Windows global memory handle which can then be passed to the scan functions in the library. If an error occurs the handle will come back as 0 to indicate that the buffer was not allocated.

## cbWinBufAlloc32() function

Allocates a Windows global memory buffer for use with the scan functions, and returns a memory handle for the buffer.

Most devices return data in a 16-bit format. For these devices, the buffer can be created using `cbWinBufAlloc()`. Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using `cbWinBufAlloc32()` or [cbWinBufAlloc64\(\)](#). See hardware-specific information to determine the type of buffer needed. If not specifically mentioned, use [cbWinBufAlloc\(\)](#).

### Function Prototype

C/C++

```
HGLOBAL cbWinBufAlloc32(long NumPoints)
```

Visual Basic

```
Function cbWinBufAlloc32(ByVal NumPoints&) As Long
```

### Arguments

*NumPoints*

The size of the buffer to allocate. Specifies how many data points (32-bit integers, NOT bytes) can be stored in the buffer.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other functions in the library, this function does not return an error code. It returns a Windows global memory handle which can then be passed to the scan functions in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.

## cbWinBufAlloc64() function

Allocates a Windows global memory buffer large enough to hold double precision data values, and returns a memory handle for the buffer.

### Function Prototype

C/C++

```
HGLOBAL cbWinBufAlloc64(long NumPoints);
```

Visual Basic

```
Function cbWinBufAlloc64(ByVal NumPoints&) As Long
```

### Arguments

*NumPoints*

The size of the buffer to allocate. Specifies the number of double precision values (8-byte or 64-bit) that the buffer will hold.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other functions in the library, this function does not return an error code. It returns a Windows global memory handle which can then be passed to the scan functions in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.

## cbWinBufFree() function

Frees a Windows global memory buffer which was previously allocated with [cbWinBufAlloc\(\)](#), [cbWinBufAlloc32\(\)](#), or [cbWinBufAlloc64\(\)](#).

### Function Prototype

C/C++

```
int cbWinBufFree(int MemHandle)
```

Visual Basic

```
Function cbWinBufFree(ByVal MemHandle&) As Long
```

### Arguments

*MemHandle*

A Windows memory handle. This must be a memory handle that was returned by [cbWinBufAlloc\(\)](#), [cbWinBufAlloc32\(\)](#), or [cbWinBufAlloc64\(\)](#) when the buffer was allocated.

### Returns

- [Error code](#) or zero if no errors.

## cbWinArrayToBuf() function

Copies data from an array into a Windows memory buffer.

### Function Prototype

C/C++

```
int cbWinArrayToBuf(unsigned short *dataArray, int MemHandle, long FirstPoint, long Count)
```

Visual Basic

```
Function cbWinArrayToBuf(DataArray%, ByVal MemHandle&, ByVal FirstPoint&, ByVal Count&) As Long
```

### Arguments

*dataArray*

The array containing the data to be copied.

*MemHandle*

This must be a memory handle that was returned by [cbWinBufAlloc\(\)](#) when the buffer was allocated. The data will be copied into this buffer.

*FirstPoint*

Index of first point in memory buffer where data will be copied to.

*Count*

Number of data points to copy.

### Returns

- [Error code](#) or zero if no errors.

### Notes

- This function copies data from an array to a Windows global memory buffer. This would typically be used to initialize the buffer with data before doing an output scan. Using the FirstPoint and Count arguments it is possible to fill a portion of the buffer. This can be useful if you want to send new data to the buffer after a BACKGROUND+CONTINUOUS output scan has been started – for example, during circular buffering.

Although this function is available to Windows C, it is not necessary since it is possible to manipulate the memory buffer directly by casting the MemHandle returned from [cbWinBufAlloc\(\)](#) to the appropriate type. This method avoids having to copy the data from an array to a memory buffer.

Refer to the following example:

```
long Count= 1000;
unsigned short *dataArray=NULL;
int MemHandle = 0;

/*allocate the buffer and cast it to an unsigned short*/
MemHandle = cbWinBufAlloc(Count);
dataArray = (unsigned short*)MemHandle;

/*calculate and store the waveform*/
for(int i=0; i<Count; ++i)
    dataArray[i] = 2047*(1.0 + sin(6.2832*i/Count));

/*output the waveform*/
cbAOutScan(.....,MemHandle,...);

/*free the buffer and NULL the pointer*/
cbWinBufFree(MemHandle);
dataArray = NULL;
```

## cbWinBufToArray() function

Copies data from a Windows memory buffer into an array.

### Function Prototype

C/C++

```
int cbWinBufToArray(int MemHandle, unsigned short* DataArray, long FirstPoint, long Count)
```

Visual Basic

```
Function cbWinBufToArray(ByVal MemHandle&, DataArray%, ByVal FirstPoint&, ByVal Count&) As Long
```

### Arguments

*MemHandle*

This must be a memory handle that was returned by [cbWinBufAlloc\(\)](#) when the buffer was allocated. The buffer should contain the data that you want to copy.

*DataArray*

The array that the data is copied to.

*FirstPoint*

Index of the first point in the memory buffer that data is copied from.

*Count*

Number of data points to copy.

### Returns

- [Error code](#) or zero if no errors.

### Notes

- This function copies data from a Windows global memory buffer to an array. This would typically be used to retrieve data from the buffer after executing an input scan function.

Using the FirstPoint and Count argument it is possible to copy only a portion of the buffer to the array. This can be useful if you want foreground code to manipulate previously collected data while a BACKGROUND scan continues to collect new data.

Although this function is available to Windows C programs, it is not necessary, since it is possible to manipulate the memory buffer directly by casting the MemHandle returned from [cbWinBufAlloc\(\)](#) to the appropriate type. This method avoids having to copy the data from the memory buffer to an array.

Refer to the following example:

```
/*declare and initialize the variables*/
long Count=1000;
unsigned short *DataArray=NULL;
int MemHandle=0;

/*allocate the buffer and cast it to a pointer to an unsigned short*/
MemHandle = cbWinBufAlloc(Count);
DataArray = (unsigned short*)MemHandle;

/*Scan the waveform data*/
cbAInScan(.....,MemHandle,...);

/*print the results*/
for(int i=0; i<Count; ++i)
    printf("Data[%d]=%d\n", i, DataArray[i])

/*free the buffer and NULL the pointer*/
cbWinBufFree(MemHandle);
DataArray = NULL;
```

## cbWinBufToArray32() function

Copies 32-bit data from a Windows memory buffer into an array.

### Function Prototype

C/C++

```
int cbWinBufToArray32(int MemHandle, unsigned long* DataArray, long FirstPoint, long Count)
```

Visual Basic

```
Function cbWinBufToArray32(ByVal MemHandle&, DataArray&, ByVal FirstPoint&, ByVal Count&) As Long
```

### Arguments

*MemHandle*

The memory handle that was returned by [cbWinBufAlloc32\(\)](#) when the buffer was allocated. The buffer should contain the data that you want to copy.

*DataArray*

The array that the data is copied to.

*FirstPoint*

The index of the first point in the memory buffer that data is copied from.

*Count*

The number of data points to copy.

### Returns

- [Error code](#) or zero if no errors.

### Notes

- You can copy only a portion of the buffer to the array using the FirstPoint and Count argument. This is useful if you want foreground code to manipulate previously collected data while a BACKGROUND scan continues to collect new data.
- Although this function is available to Windows C programs, it is not necessary, since you can manipulate the memory buffer directly by casting the MemHandle returned from cbWinBufAlloc32() to the appropriate type. This method avoids having to copy the data from the memory buffer to an array.

Refer to the following example:

```
/*declare and initialize the variables*/
long Count = 1000;
unsigned short *DataArray = NULL;
int MemHandle = 0;

/*allocate the buffer and cast it to a pointer to an unsigned long*/
MemHandle = cbWinBufAlloc32(Count);
DataArray = (unsigned long*)MemHandle;

/*scan in the data*/
cbCInScan(.....,MemHandle,...);

/*print the results*/
for(int i=0; i<Count; ++i)
    printf("Data[%d]=%d\n", i, DataArray[i]);

/*free the buffer and NULL the pointer*/
cbWinBufFree(MemHandle);
DataArray = NULL;
```



## cbScaledWinArrayToBuf() function

Copies double precision values from an array into a Windows memory buffer.

### Function Prototype

C/C++

```
int cbScaledWinArrayToBuf(double *dataArray, HGLOBAL MemHandle, long StartPt, long Count);
```

Visual Basic

```
function cbScaledWinArrayToBuf(dataArray#, ByVal MemHandle&, ByVal FirstPoint&, ByVal CBCount&) As Long
```

### Arguments

*dataArray*

The array containing the data to be copied.

*MemHandle*

This must be a memory handle that was returned by [cbScaledWinBufAlloc\(\)](#) when the buffer was allocated. The data will be copied into this buffer.

*FirstPoint*

Index of the first point in the memory buffer where the data will be copied.

*Count*

Number of data points to copy.

### Returns

- [Error code](#) or zero if no errors.

### Notes

- This function is used in conjunction with the SCALEDATA scan option and [cbScaledWinBufAlloc\(\)](#).

## cbScaledWinBufAlloc() function

Allocates a Windows global memory buffer large enough to hold scaled data obtained from scan operations in which the SCALEDATA scan option is selected, and returns a memory handle for the buffer.

### Function Prototype

C/C++

```
int cbScaledWinBufAlloc(int NumPoints);
```

Visual Basic

```
Function cbScaledWinBufAlloc(ByVal NumPoints&) As Long
```

### Arguments

*NumPoints*

The size of the buffer to allocate. Specifies the number of double precision values (8-byte or 64-bit) that the buffer will hold.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- This function is used in conjunction with the SCALEDATA scan option and [cbScaledWinBufToArray\(\)](#) or [cbScaledWinArrayToBuf\(\)](#).
- Unlike most other functions in the library, this function does not return an error code. It returns a Windows global memory handle which can then be passed to the scan functions in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.

## cbScaledWinBufToArray() function

Copies double precision values from a Windows memory buffer into an array.

### Function Prototype

C/C++

```
int cbScaledWinBufToArray(int MemHandle, double* DataArray, long FirstPoint, long Count);
```

Visual Basic

```
Function cbScaledWinBufToArray(ByVal MemHandle&, DataArray#, ByVal FirstPoint&, ByVal Count&) As Long
```

### Arguments

*MemHandle*

The memory handle that was returned by [cbScaledWinBufAlloc\(\)](#) when the buffer was allocated. The buffer should contain the data that you want to copy.

*DataArray*

A pointer to the start of the destination array to which the data samples are copied.

*FirstPoint*

The buffer index of the first sample to copy from the buffer.

*Count*

The number of samples to copy into DataArray.

### Returns

- [Error code](#) or zero if no errors.

### Notes

- This function is used in conjunction with the SCALEDATA scan option and [cbScaledWinBufAlloc\(\)](#).

## cbDeviceLogin() function

Opens a device session with a shared device.

### Function Prototype

C/C++

```
int cbDeviceLogin(int BoardNum, char* UserName, char* Password);
```

Visual Basic

```
Function cbDeviceLogin(ByVal BoardNum&, UserName$, Password$) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*UserName*

A null-terminated string that identifies the user name used to log in to a device session.

*Password*

A null-terminated string that identifies the password used to log in to a device session.

### Returns

- [Error code](#) or 0 if no errors.

### Notes

- If the user name or password is invalid, the function returns INVALIDLOGIN.
- If the session is already opened by another user, the function returns SESSIONINUSE.

## cbDeviceLogout() function

Releases the device session with a shared device.

### Function Prototype

C/C++

```
int cbDeviceLogout(int BoardNum);
```

Visual Basic

```
Function cbDeviceLogout(ByVal BoardNum) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

### Returns

- [Error code](#) or 0 if no errors.

## cbDisableEvent function

Disables one or more event conditions and disconnects their user-defined handlers.

### Function Prototype

C/C++

```
int cbDisableEvent(int BoardNum, unsigned EventType)
```

Visual Basic

```
Function cbDisableEvent(ByVal BoardNum&, ByVal EventType&) as Long
```

### Arguments

*BoardNum*

The board number used to indicate which device's event handling will be disabled. BoardNum may be 0 to 99. Refers to the number associated with the board when it was installed with InstaCal.

*EventType*

Specifies one or more event conditions to disable. More than one event type can be specified by bitwise OR'ing the event types. Note that specifying an event that has not been enabled is benign and will not cause any errors. Refer to [cbEnableEvent\(\)](#) for valid EventType settings.

To disable all events in a single call, use ALL\_EVENT\_TYPES.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- For most event types, this function cannot be called while any background operations ([cbAInScan\(\)](#), [cbAPretrig\(\)](#), or [cbAOutScan\(\)](#)) are active. Perform a [cbStopBackground](#) before calling cbDisableEvent(). However, for ON\_EXTERNAL\_INTERRUPT events, you can call cbDisableEvent() while the board is actively generating events.

### Important!

In order to understand the functions, refer to the board-specific information in the *Universal Library User's Guide* and also in the Readme files installed with the Universal Library. We also urge you to examine and run one or more of the example programs supplied prior to attempting any programming of your own. Following this advice may save you hours of frustration, and wasted time.

This note, which appears elsewhere, is especially applicable to this function. Now is the time to read the board specific information for your board. We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.

## cbEnableEvent() function

Binds one or more event conditions to a user-defined callback function. Upon detection of an event condition, the user-defined function is invoked with board- and event-specific data. Detection of event conditions occurs in response to interrupts. Typically, this function is used in conjunction with interrupt driven processes such as [cbAInScan\(\)](#), [cbAPretrig\(\)](#), or [cbAOutScan\(\)](#).

### Function Prototype

#### C/C++

```
int cbEnableEvent(int BoardNum, unsigned EventType, unsigned EventParam, void* CallbackFunc, void* UserData)
```

#### Visual Basic

```
Function cbEnableEvent(ByVal BoardNum&, ByVal EventType&, ByVal EventParam&, ByVal CallbackFunc&, ByRef UserData as Any) as Long
```

### Arguments

#### *BoardNum*

The board number used to indicate which device will generate the event conditions. BoardNum may be 0 to 99. Refers to the number associated with the board when it was installed with InstaCal.

#### *EventType*

Specifies one or more event conditions that will be bound to the user-defined callback function. More than one event type can be specified by bitwise OR'ing the event types. Refer to the constants in the [EventType argument values](#) below.

#### *EventParam*

Additional data required to specify some event conditions, such as an ON\_DATA\_AVAILABLE event or ON\_EXTERNAL\_INTERRUPT event.

For ON\_DATA\_AVAILABLE events, EventParam is used to determine the minimum number of samples to acquire during an analog input scan before generating the event. For ON\_EXTERNAL\_INTERRUPT events, EventParam is used to latch digital bits on supported hardware by setting it to one of the constants in the [EventParam argument values](#) section below.

Most event conditions ignore this value.

#### *CallbackFunc*

The address of or pointer to the user-defined callback function to handle the above event type(s). This function must be defined using the standard call (\_\_stdcall) calling convention. Consequently, Visual Basic programs must define their callback functions in standard modules (.bas) and cannot be object methods. C++ programs can define this callback function as either a global function or as a static member function of a class (note that static members do NOT have access to instance specific data).

Refer to the "[User Callback function](#)" for proper function syntax.

#### *UserData*

The address of or pointer to user-defined data that will be passed to the user-defined callback function. This parameter is NOT dereferenced by the library or its drivers; as a consequence, a NULL pointer can be supplied.

### Returns

- [Error code](#) or 0 if no errors

## EventType argument values

ON_DATA_AVAILABLE	Generates an event whenever the number of samples acquired during an analog input scan increases by EventParam samples or more. Note that for BLOCKIO scans, events will be generated on packet transfers; for example, even if EventParam is set to 1, events will only be generated every packet-size worth of data (256 samples for the PCI-DAS1602) for aggregate rates greater than 1 kHz for the default cbInScan() mode.  For <a href="#">cbAPretrig()</a> , the first event is not generated until a minimum of EventParam samples after the pretrigger.
ON_END_OF_AI_SCAN	Generates an event upon completion or fatal error of <a href="#">cbInScan()</a> or <a href="#">cbAPretrig()</a> . Some devices, such as the USB-1208FS and USB-1408FS, will generate an end of scan event after <a href="#">cbStopBackground</a> is called, but most devices do not. Handle post-scan tasks directly after calling cbStopBackground.
ON_END_OF_AO_SCAN	Generates an event upon completion or fatal error of <a href="#">cbAOutScan()</a> . Some devices, such as the USB-1208FS and USB-1408FS, will generate an end of scan event after <a href="#">cbStopBackground</a> is called, but most devices do not. Handle post-scan tasks directly after calling cbStopBackground.
ON_EXTERNAL_INTERRUPT	For some digital and counter boards, generates an event upon detection of a pulse at the External Interrupt pin.
ON_PRETRIGGER	For <a href="#">cbAPretrig()</a> , generates an event upon detection of the first trigger.
ON_SCAN_ERROR	Generates an event upon detection of a driver error during BACKGROUND input and output scans. This includes OVERRUN, UNDERRUN, and TOOFEW errors.

## EventParam argument values

LATCH_DI	Returns the data that was latched in at the most recent interrupt edge.
LATCH_DO	Latches out the data most recently written to the hardware.

## Notes

- This function cannot be called while any background operations ([cbInScan\(\)](#), [cbAPretrig\(\)](#), or [cbAOutScan\(\)](#)) are active. If a background operation is in progress when cbEnableEvent() is called, the function returns an [ALREADYACTIVE](#) error. Perform a [cbStopBackground\(\)](#) call before calling cbEnableEvent().
- Events will not be generated faster than the user callback function can handle them. If an event type becomes multi-signaled before the event handler returns, events are merged. The event handler is called once per event type, and is supplied with the event data corresponding to the latest event. When more than one event type is generated, the event handler for each event type is called in the same order in which they are enabled.
- Events are generated while handling board-generated interrupts. Therefore, using cbStopBackground() to abort background operations *does not* generate ON\_END\_OF\_AI\_SCAN or ON\_END\_OF\_AO\_SCAN events. However, the event handlers can be called immediately after calling cbStopBackground().
- cbEnableEvent() is intended for use with Windows applications. Use with console applications can produce unpredictable results.



## cbFlashLED() function

Causes the LED on a USB device to flash.

### Function Prototype

C/C++

```
int cbFlashLED(int BoardNum);
```

Visual Basic

```
Function cbFlashLED(ByVal BoardNum&) as Long
```

### Argument

*BoardNum*

The board number of the USB device whose LED will flash.

### Note

After calling cbFlashLED(), wait a few seconds before calling additional functions, or execution of the next function may fail.

## cbFromEngUnits() function

Converts a single precision voltage (or current) value in engineering units to an integer count value. This function is typically used to obtain a data value from a voltage value for output to a D/A with functions such as cbAOut().

### Function Prototype

C/C++

```
int cbFromEngUnits(int BoardNum, int Range, float EngUnits, unsigned short *DataVal)
```

Visual Basic

```
Function cbFromEngUnits(ByVal BoardNum&, ByVal Range&, ByVal EngUnits!, DataVal%) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. This function uses the board number to determine the resolution and polarity values to use in the conversion. BoardNum may be 0 to 99.

*Range*

The voltage (or current) range to use for the conversion to counts. When using this function to obtain a value to send to a D/A board, keep in mind that some D/A boards have programmable voltage ranges, and others set the voltage range via switches on the board. In either case, the desired range must be passed to this function.

Refer to board-specific information in the *Universal Library User's Guide* for a list of the [supported A/D ranges](#) of each board.

*EngUnits*

The single precision voltage (or current) value to use for the conversion to counts. Set the value to be within the range specified by the Range argument.

*DataVal*

The function returns an integer count to this variable that is equivalent to the EngUnits argument using the resolution of the D/A on the board referenced by BoardNum (if any).

### Returns

- [Error code](#) or 0 if no errors.
- DataVal – the integer count equivalent to EngUnits is returned here.

### Notes

- This function is not supported for hardware with resolution greater than 16 bits.

The default resolution of this function is 12 bits. If the device referenced by BoardNum has neither analog input nor analog output, the result is a 12 bit conversion.

If the device referenced by BoardNum has both analog input and analog output, the resolution and transfer function of the D/A converter on the device is used.

## cbGetBoardName() function

Returns the board name of a specified board.

### Function Prototype

C/C++

```
int cbGetBoardName(int BoardNum, char *BoardName)
```

Visual Basic

```
Function cbGetBoardName(ByVal BoardNum&, ByVal BoardName$) As Long
```

### Arguments

*BoardNum*

Refers either to the number associated with a board when it was installed with InstaCal, GETFIRST, or GETNEXT. BoardNum may be 0 to 99, or GETFIRST or GETNEXT.

*BoardName*

A null-terminated string variable that the board name will be returned to. This string variable must be pre-allocated to be at least as large as BOARDNAMELEN. This size is guaranteed to be large enough to hold the longest board name string. Refer also to the board type codes in the "[Measurement Computing Device IDs](#)" section.

### Returns

- [Error code](#) or 0 if no errors.
- BoardName – return string containing the board name.

### Notes

There are two distinct ways of using this function:

- Pass a board number as the BoardNum argument. The string that is returned describes the board type of the installed board.
- Set BoardNum to GETFIRST or GETNEXT to get a list of all board types that are supported by the library. Set BoardNum to GETFIRST to get the first board type in the list of supported boards. Subsequent calls with Board=GETNEXT returns each of the other board types supported by the library. When you reach the end of the list, BoardName is set to an empty string. Refer to the **ulgt04** example program for more details.

# cbGetStatus() function

Returns the status about the background operation currently running.

## Function Prototype

C/C++

```
int cbGetStatus(int BoardNum, int *Status, long *CurCount, long *CurIndex, int FunctionType)
```

Visual Basic

```
Function cbGetStatus(ByVal BoardNum&, Status%, CurCount&, CurIndex&, FunctionType&) As Long
```

## Arguments

*BoardNum*

The number associated with the board when it was installed with the InstaCal. BoardNum may be 0 to 99.

*Status*

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

*CurCount*

The CurCount argument specifies how many points have been input or output since the Background process started. Use it to gauge how far along the operation is towards completion. Generally, CurCount returns the total number of samples transferred between the DAQ board and the Windows data buffer at the time cbGetStatus() was called.

When you set both the CONTINUOUS and BACKGROUND options, CurCount's behavior depends on the board model. Refer to the board-specific information in the *Universal Library User's Guide* for the behavior of your board.

With recent MCC DAQ designs, the CurCount argument continually increases in increments of the packet size as Windows' circular data buffer recycles, until it reaches 231. Since the Count argument is a signed integer, at 2,147,483,647 + 1, the Count rolls back to a negative number (-2,147,483,647). The Count argument resumes incrementing, eventually reaching 0 and increasing back up to 2,147,483,647.

The CurIndex argument is usually more useful than the CurCount argument in managing data collected when you set both the CONTINUOUS and BACKGROUND options.

*CurIndex*

The CurIndex argument is an index into the Windows data buffer. This index points to the start of the last completed channel scan that was transferred between the DAQ board and the Windows data buffer. If no points in the buffer have been transferred, CurIndex equals -1 in most cases.

For CONTINUOUS operations, CurIndex rolls over when the Windows data buffer is full. This rollover indicates that "new" data is now overwriting "old" data. Your goal is to process the old data before it gets overwritten. You can keep ahead of the data flow by copying the old data out of the buffer before new data overwrites it.

The CurIndex argument can help you access the most recently transferred data. Your application does not have to process the data exactly when it becomes available in the buffer – in fact, you should avoid doing so unless absolutely necessary. The CurIndex argument generally increments by the packet size, but in some cases the CurIndex increment can vary within the same scan. One instance of a variable increment is when the packet size is not evenly divisible by the number of channels.

You should determine the best size of the "chunks" of data that your application can most efficiently process, and then periodically check on the CurIndex argument value to determine when that amount of additional data has been transferred.

Refer to the *Universal Library User's Guide* for information on your board, particularly when using Pre-Trigger.

*FunctionType*

Specifies which scan to retrieve status information about. Refer to the [FunctionType argument values](#) section below.

## Returns

- [Error code](#) or 0 if no errors
- Status – Returns the status of the operation:
  - 0 – a background process is not currently executing.
  - 1 – a background process is currently executing.
- CurCount – Returns the current number of samples collected.
- CurIndex – Returns the Current sample index.

## FunctionType argument values

AIFUNCTION	Specifies analog input scans started with <a href="#">cbAInScan()</a> or <a href="#">cbAPretrig()</a> .
AOFUNCTION	Specifies analog output scans started with <a href="#">cbAOutScan()</a> .

DIFUNCTION	Specifies digital input scans started with <a href="#">cbDInScan()</a> .
DOFUNCTION	Specifies digital output scans started with <a href="#">cbDOutScan()</a> .
CTRFUNCTION	Specifies counter background operations started with <a href="#">cbCStoreOnInt()</a> or <a href="#">cbCInScan()</a> .
DAQIFUNCTION	Specifies a synchronous input scan started with <a href="#">cbDagInScan()</a> .
DAQOFUNCTION	Specifies a synchronous output scan started with <a href="#">cbDagOutScan()</a> .

## cbGetTCValues() function

Converts raw thermocouple data collected using the [cbDagInScan\(\)](#) function to data on a temperature scale (Celsius, Fahrenheit, or Kelvin).

### Function Prototype

C/C++

```
int cbGetTCValues(int BoardNum, short *ChanArray, short *ChanTypeArray, int ChanCount, int MemHandle,
int FirstPoint, long Count, int Scale, float *TempValArray)
```

Visual Basic

```
Function cbGetTCValues (ByVal BoardNum&, ChanArray%, ChanTypeArray%, ByVal ChanCount&, ByVal
MemHandle&, ByVal FirstPoint&, ByVal Count&, ByVal CbScale&, TempValArray!) As Long
```

### Arguments

*BoardNum*

The board number used to collect the data. BoardNum may be 0 to 99. Refers to the number associated with the board used to collect the data when it was installed with InstaCal. The specified board must support synchronous input.

*ChanArray*

Array containing channel values. Valid channel values are analog and temperature input channels and digital ports. ChanArray must match the channel array used with the [cbDagInScan\(\)](#) function.

*ChanTypeArray*

Array containing channel types. Each element of this array defines the type of the corresponding element in the ChanArray. ChanTypeArray must match the channel type settings used with the [cbDagInScan\(\)](#) function

*ChanCount*

Number of elements in ChanArray.

*MemHandle*

This must be a memory handle that was returned by [cbWinBufAlloc\(\)](#) when the buffer was allocated. The buffer should contain the data that you want to convert.

*FirstPoint*

The index into the raw data memory buffer that holds the first sample of the first channel to be converted. The index into the raw memory is (FirstPoint x ChanCount) so that converted data always starts with the first channel specified in the scan. For example, if FirstPoint is 14 and the number of channels is 8, the index of the first converted sample is 112.

*Count*

The number of samples per channel to convert to engineering units. Count should not exceed Windows buffer size / ChanCount – FirstPoint.

*Scale*

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

*TempArray*

The array to hold the converted data. This array must be allocated by the user, and must be large enough to hold count samples x the number of temperature channels.

### Returns

- [Error code](#) or 0 if no errors
- TempValArray – Converted data.

## cbInByte() function

Reads a byte from a hardware register on a board.

### Function Prototype

C/C++

```
int cbInByte(int BoardNum, int PortNum)
```

Visual Basic

```
Function cbInByte(ByVal BoardNum&, ByVal PortNum&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this argument to the offset for the desired register. This function takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

### Returns

- The current value of the specified register.

### Notes

- cbInByte() is used to read 8 bit ports. [cbInWord\(\)](#) is used to read 16-bit ports.

This function was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.

## cbInWord() function

Reads a word from a hardware register on a board.

### Function Prototype

C/C++

```
int cbInWord(int BoardNum, int PortNum)
```

Visual Basic

```
Function cbInWord(ByVal BoardNum&, ByVal PortNum&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this argument to the offset for the desired register. This function takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

### Returns

- The current value of the specified register

### Notes

- [cbInByte\(\)](#) is used to read 8 bit ports. cbInWord() is used to read 16-bit ports.

This function was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.



## cbOutByte() function

Writes a byte to a hardware register on a board.

### Function Prototype

C/C++

```
int cbOutByte(int BoardNum, int PortNum, int PortVal)
```

Visual Basic

```
Function cbOutByte(ByVal BoardNum&, ByVal PortNum&, ByVal PortVal%) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this argument to the offset for the desired register. This function takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

*PortVal*

Value that is written to the register.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- cbOutByte() is used to write to 8 bit ports. [cbOutWord\(\)](#) is used to write to 16-bit ports.

This function was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.

## cbOutWord() function

Writes a word to a hardware register on a board.

### Function Prototype

C/C++

```
int cbOutWord(int BoardNum, int PortNum, int PortVal)
```

Visual Basic

```
Function cbOutByte(ByVal BoardNum&, ByVal PortNum&, ByVal PortVal%) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*PortNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this argument to the offset for the desired register. This function takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

*PortVal*

Value that is written to the register.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- [cbOutByte\(\)](#) is used to write to 8 bit ports. cbOutWord() is used to write to 16-bit ports.

This function was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.

## cbRS485() function

Sets the direction of RS-485 communications port buffers.

### Function Prototype

C/C++

```
int cbRS485(int BoardNum, int Transmit, int Receive)
```

Visual Basic

```
Function cbRS485(ByVal BoardNum&, ByVal Transmit&, ByVal Receive&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*Transmit*

Set to ENABLED or DISABLED (CBENABLED or CBDISABLED in Visual Basic). The transmit RS-485 line driver is turned on. Data written to the RS-485 UART chip is transmitted to the cable connected to that port.

*Receive*

Set to ENABLED or DISABLED (CBENABLED or CBDISABLED in Visual Basic). The receive RS-485 buffer is turned on. Data present on the cable connected to the RS-485 port is received by the UART chip.

### Returns

- [Error code](#) or 0 if no errors

### Notes

- You can simultaneously enable or disable the transmit and receive buffers. If both are enabled, data written to the port is also received by the port. For a complete discussion of RS485 network construction and communication, refer to the CIO-COM485 or PCM-COM485 hardware manual.

## cbStopBackground() function

Stops one or more subsystem background operations that are in progress for the specified board. Use this function to stop any function that is running in the background. This includes any function that was started with the BACKGROUND option, as well as [cbCStoreOnInt\(\)](#) (which always runs in the background).

Execute `cbStopBackground()` after normal termination of all background functions to clear variables and flags.

### Function Prototype

C/C++

```
int cbStopBackground(int BoardNum, int FunctionType)
```

Visual Basic

```
Function cbStopBackground(ByVal BoardNum&, ByVal FunctionType&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*FunctionType*

Specifies which background operation to stop. Specifies which scan to retrieve status information about. Refer to the [FunctionType argument values](#) section below.

### Returns

- [Error code](#) or 0 if no errors

### FunctionType argument values

AIFUNCTION	Specifies analog input scans started with <a href="#">cbAInScan()</a> or <a href="#">cbAPretrig()</a> .
AOFUNCTION	Specifies analog output scans started with <a href="#">cbAOutScan()</a> .
DIFUNCTION	Specifies digital input scans started with <a href="#">cbDInScan()</a> .
DOFUNCTION	Specifies digital output scans started with <a href="#">cbDOutScan()</a> .
CTRFUNCTION	Specifies counter background operations started with <a href="#">cbCStoreOnInt()</a> or <a href="#">cbCInScan()</a> .
DAQIFUNCTION	Specifies a synchronous input scan started with <a href="#">cbDagInScan()</a> .
DAQOFUNCTION	Specifies a synchronous output scan started with <a href="#">cbDagOutScan()</a> .

## cbTEDSRead() function

Reads data from a TEDS sensor into an array.

### Function Prototype

C/C++

```
int cbTEDSRead(int BoardNum, int Chan, BYTE* DataBuffer, long *Count, int Options)
```

Visual Basic

```
Function cbTEDSRead(ByVal BoardNum&, ByVal Chan&, DataBuffer, CBCount&, ByVal Options&) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. BoardNum may be 0 to 99.

*Chan*

A/D channel number.

*DataBuffer*

Pointer to the data array.

*Count*

Number of data points to read.

*Options*

Reserved for future use.

### Returns

- [Error code](#) or 0 if no errors.
- Count - The actual number of data points read.

### Options argument values

*Default*

Reserved for future use.

## cbToEngUnits() function

Converts an integer count value to an equivalent single precision voltage (or current) value. This function is typically used to obtain a voltage value from data received from an A/D with functions such as [cbAIn\(\)](#).

### Function Prototype

C/C++

```
int cbToEngUnits(int BoardNum, int Range, unsigned short DataVal, float *EngUnits)
```

Visual Basic

```
Function cbToEngUnits(ByVal BoardNum&, ByVal Range&, ByVal DataVal%, EngUnits!) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. This function uses the board number to determine the resolution and polarity values to use for the conversion. BoardNum may be 0 to 99.

*Range*

Voltage (or current) range to use for the conversion to engineering units. When using this function to obtain engineering units from a value received from an A/D board, keep in mind that some A/D boards have programmable voltage ranges, and others set the voltage range via switches on the board. In either case, the desired range must be passed to this function.

Refer to board-specific information in the Universal Library User's Guide for a list of the [supported A/D ranges](#) of each board.

*DataVal*

An integer count value (typically, one returned from an A/D board).

*EngUnits*

The single precision voltage (or current) value that is equivalent to DataVal is returned to this variable. The value will be within the range specified by the Range argument using the resolution of the A/D on the board referenced by BoardNum (if any).

### Returns

- [Error code](#) or 0 if no errors.
- EngUnits – the engineering units value equivalent to DataVal is returned to this variable.

### Notes

- This function is not supported for hardware with resolution greater than 16 bits.
- The default resolution of this function is 12 bits, so if the device referenced by BoardNum has neither analog input nor analog output, the result will be a 12 bit conversion.
- If the device referenced by BoardNum has both analog input and analog output, the resolution and transfer function of the A/D converter on the device is used.

## cbToEngUnits32() function

Converts an integer count value to an equivalent double precision voltage (or current) value.

### Function Prototype

C/C++

```
int cbToEngUnits32(int BoardNum, int Gain, unsigned long DataValue, double *EngUnits);
```

Visual Basic

```
Function cbToEngUnits32(ByVal BoardNum&, ByVal Gain&, ByVal DataValue&, EngUnits#) As Long
```

### Arguments

*BoardNum*

The number associated with the board when it was installed with InstaCal. This function uses the board number to determine the resolution and polarity values to use for the conversion. BoardNum may be 0 to 99.

*Gain*

Voltage (or current) range to convert to engineering units. When using this function to obtain engineering units from a value received from an A/D board, keep in mind that some A/D boards have programmable voltage ranges, and others set the voltage range via switches on the board. In either case, the desired range must be passed to this function.

Refer to board-specific information in the *Universal Library User's Guide* for a list of the [supported A/D ranges](#) of each board.

*DataValue*

An integer count value (typically, one returned from an A/D board) to convert to engineering units.

*EngUnits*

The double precision voltage (or current) value that is equivalent to DataValue is returned to this variable. The value will be within the range specified by the Range argument using the resolution of the A/D on the board referenced by BoardNum (if any).

### Returns

- [Error code](#) or 0 if no errors.
- EngUnits – the engineering units value equivalent to DataValue is returned to this variable.

### Notes

- This function is typically used to obtain a voltage (or current) value from data received from an A/D with functions such as [cbAIn32\(\)](#).
- This function should be used for devices with a resolution of 20-bits or more.

The default resolution of this function is 32 bits, so if the device referenced by BoardNum has neither analog input nor analog output, the result will be a 32 bit conversion.

If the device referenced by BoardNum has both analog input and analog output, the resolution and transfer function of the A/D converter on the device is used.

# User Callback function

The User Callback function is called as an argument of the `cbEnableEvent()` function. You create the function using the prototype shown below. You call the function by passing either it's address or a pointer to the function to the `CallbackFunc` argument of the `cbEnableEvent()` function.

## Callback function prototype

C/C++

```
void __stdcall CallbackFunc(int BoardNum, unsigned EventType, unsigned EventData, void* UserData);
```

Visual Basic

```
Sub CallbackFunc(ByVal BoardNum&, ByVal EventType&, ByVal EventData&, ByRef UserData as UserDataTypes)
    where UserDataTypes is the data type of the UserData argument passed to cbEnableEvent\(\).
End Sub
```

## Arguments

- BoardNum*  
Indicates which board caused the event.
- EventType*  
Indicates which event occurred.
- EventData*  
Board-specific data associated with this event. Returns the value of the `EventType` as listed in the "[EventData argument values](#)" section below.
- UserData*  
The pointer or reference to data supplied by the `UserData` parameter in [cbEnableEvent\(\)](#). Note that before use, this parameter must be cast to the same data type as passed in to `cbEnableEvent()`.

## EventData argument values

EventType	Value of EventData
ON_DATA_AVAILABLE	The number of samples acquired since the start of the scan.
ON_END_OF_AI_SCAN	The total number of samples acquired upon the scan completion or end.
ON_END_OF_AO_SCAN	The total number of samples output upon the scan completion or end.
ON_EXTERNAL_INTERRUPT	The number of interrupts generated since enabling the <code>ON_EXTERNAL_INTERRUPT</code> event.
ON_PRETRIGGER	The number of pretrigger samples available at the time of pretrigger. This value is invalid for some boards when a <a href="#">TOOFEW</a> error occurs. Refer to board details.
ON_SCAN_ERROR	The <a href="#">error code</a> of the scan error.



## UL for .NET Class Library Overview

The Microsoft .NET platform provides a framework that allows for the development of Windows applications using a wide range of new programming languages. These languages include VB .NET, C#, managed C++, JScript, and any other language that is compliant with the .NET Common Language Runtime (CLR). The CLR is a multi-language execution environment.

The interface to the Universal Library consists of standard "C" functions. These functions are not CLR-compliant. Therefore, the Universal Library for .NET was developed. This library enables the various .NET programming languages to call into the Universal Library.

The Universal Library for .NET consists of a set of classes. For the most part, the methods within each class have a corresponding function in the standard UL. Each UL for .NET method has virtually the same parameter set as their UL counterparts.

## MccDaq namespace

The MccDaq namespace is a logical naming scheme for grouping related types. The .NET Framework uses a hierarchical naming scheme for grouping types into logical categories of related functionality. The namespace contains the classes and enumerated constants by which your UL for .NET applications can access the Universal Library data types and functions.

## MccDaq classes

The MccDaq namespace contains five main classes:

- [MccBoard class](#)
- [ErrorInfo class](#)
- [MccService class](#)
- [GlobalConfig class](#)
- [DataLogger class](#)

The MccDaq namespace also contains the following four secondary classes:

- [cBoardConfig](#): Contains all of the members for setting and getting board-level configuration.
- [cCtrConfig](#): Contains all of the members for setting and getting the counter-level configuration of a board.
- [cDioConfig](#): Contains all of the members for getting the digital configuration of a board.
- [cExpansionConfig](#): Contains all of the members for setting and getting expansion board configuration.

These secondary classes include methods that are accessible from properties of the MccBoard class.

## MccDaq enumerated constants

The MccDaq Namespace contains enumerated values which are used by many of the methods available from the MccDaq classes. Click [here](#) for a list of the enumerated values.

## DataLogger class

Contains all of the members for reading and converting binary log files.

The DataLogger class is a member of the MccDaq namespace. Refer to the "[UL for .NET Class Library Overview](#)" for an explanation of the MccDaq namespace.

## Property and methods

The DataLogger class provides one property to get a reference to the file name associated with the current instance of the DataLogger.

The DataLogger class includes 14 methods used to read and convert the data contained in a binary log file. These methods are equivalents of the function calls used in the standard Universal Library. The methods have virtually the same parameter set as their UL counterparts.

- [FileName property](#) - Returns the file name associated with an instance of the DataLogger class.
- [ConvertFile\(\)](#) - Converts a binary log file to a comma-separated values (.CSV) text file or another text file format that you specify.
- [GetAIChannelCount\(\)](#) - Retrieves the total number of analog channels that were logged in a binary file.
- [GetAIInfo\(\)](#) - Retrieves the channel number and unit value of each analog input channel logged in a binary file.
- [GetCJCInfo\(\)](#) - Retrieves the number of CJC temperature channels that were logged.
- [GetDIOInfo\(\)](#) - Retrieves the number of digital I/O channels logged in a binary file.
- [GetFileInfo\(\)](#) - Retrieves the version level and byte size of a binary file.
- [GetFileName\(\)](#) - Retrieves the name of the  $n^{\text{th}}$  file in the directory containing binary log files.
- [GetPreferences\(\)](#) - Retrieves API preference settings for time stamp data, analog temperature data, and CJC temperature data. Returns the default values unless changed using [SetPreferences\(\)](#).
- [GetSampleInfo\(\)](#) - Retrieves the sample interval, sample count, and the date and time of the first data point in a binary file.
- [ReadAIChannels\(\)](#) - Retrieves analog input data from a binary file, and stores the values in an array.
- [ReadCJCChannels\(\)](#) - Retrieves CJC temperature data from a binary file, and stores the values in an array.
- [ReadDIOChannels\(\)](#) - Retrieves digital I/O channel data from a binary file, and stores the values in an array.
- [ReadTimeTags\(\)](#) - Retrieves date and time values logged in a binary file. This method stores date values in the dateTags array, and time values in the timeTags array.
- [SetPreferences\(\)](#) - Sets preferences for returned time stamp data, analog temperature data, and CJC temperature data.

## ErrorInfo class

Contains all of the members for storing and reporting error codes and messages. This class also includes error code enumerated constants, which define the error number and associated message that can be returned when you call a method.

Most UL for .NET methods return ErrorInfo objects. Error information is stored internally on the return from calling the low-level UL function. The error is reported when the user calls the class library methods.

The ErrorInfo class is a member of the MccDaq namespace. Refer to the "[UL for .NET Class Library Overview](#)" for an explanation of the MccDaq namespace.

## Properties and methods

The ErrorInfo class includes the following properties that you can use to examine error information.

- [Message](#) - Gets the text of the error message associated with an error constant.
- [Value](#) - Gets the error constant associated with an ErrorInfo object.
- [LogToFile](#) - When set *true*, records time-stamped error codes to a file.

## Enumerated constants

### ErrorCode

Lists the named constants for all error codes.

The error number and associated error constant are listed below. Click on the name of an error constant to display an explanation of the error message. Use the vertical scroll bar to view all of the error constants.

#### Error 0-99

0	<a href="#">NoErrors</a>
1	<a href="#">BadBoard</a>
2	<a href="#">DeadDigitalDev</a>
3	<a href="#">DeadCounterDev</a>
4	<a href="#">DeadDaDev</a>
5	<a href="#">DeadAdDev</a>
6	<a href="#">NotDigitalConf</a>
7	<a href="#">NotCounterConf</a>
8	<a href="#">NotDaConf</a>
9	<a href="#">NotAdConf</a>
10	<a href="#">NotMuxConf</a>
11	<a href="#">BadPortNum</a>
12	<a href="#">BadCounterDevNum</a>
13	<a href="#">BadDaDevNum</a>
14	<a href="#">BadSampleMode</a>
15	<a href="#">BadInt</a>
16	<a href="#">BadAdChan</a>
17	<a href="#">BadCount</a>
18	<a href="#">BadCntrConfig</a>
19	<a href="#">BadDaVal</a>
20	<a href="#">BadDaChan</a>
22	<a href="#">AlreadyActive</a>
23	<a href="#">PageOverrun</a>
24	<a href="#">BadRate</a>
25	<a href="#">CompatMode</a>
26	<a href="#">TrigState</a>
27	<a href="#">AdStatusHung</a>
28	<a href="#">TooFew</a>
29	<a href="#">OverRun</a>

#### Error 100-203

100	<a href="#">NotDosFunc</a>
101	<a href="#">RangeMismatch</a>
102	<a href="#">ClockTooSlow</a>
103	<a href="#">BadCalFactors</a>
104	<a href="#">BadConfigType</a>
105	<a href="#">BadConfigItem</a>
106	<a href="#">NoPcmciaBoard</a>
107	<a href="#">NoBackground</a>
108	<a href="#">StringTooShort</a>
109	<a href="#">ConvertExtMem</a>
110	<a href="#">BadEuAdd</a>
111	<a href="#">Das16JrRateWarning</a>
112	<a href="#">Das08TooLowRate</a>
114	<a href="#">AmbigSensorOnGp</a>
115	<a href="#">NoSensorTypeOnGp</a>
116	<a href="#">NoConversionNeeded</a>
117	<a href="#">NoExtContinuous</a>
118	<a href="#">InvalidPretrigConvert</a>
119	<a href="#">BadCtrReq</a>
120	<a href="#">BadTrigThreshold</a>
121	<a href="#">BadPcmSlotRef</a>
122	<a href="#">AmbigPcmSlotRef</a>
123	<a href="#">BadSensorType</a>
126	<a href="#">CfgFileNotFound</a>
127	<a href="#">NoVddInstalled</a>
128	<a href="#">NoWindowsMemory</a>
129	<a href="#">OutOfDosMemory</a>
130	<a href="#">ObsoleteOption</a>
131	<a href="#">NoPcmReqKey</a>

#### Error 300-1008

300	<a href="#">InternalErr32</a>
304	<a href="#">CfgFileReadFailure</a>
305	<a href="#">CfgFileWriteFailure</a>
308	<a href="#">CfgFileCantOpen</a>
325	<a href="#">BadRtdConversion</a>
326	<a href="#">NoPciBios</a>
327	<a href="#">BadPciIndex</a>
328	<a href="#">NoPciBoard</a>
334	<a href="#">CantInstallInt</a>
339	<a href="#">CantMapPCMCis</a>
344	<a href="#">NoMoreFiles</a>
345	<a href="#">BadFileName</a>
347	<a href="#">LossOfData</a>
348	<a href="#">InvalidBinaryFile</a>
400-499	<a href="#">PcmciaErrs</a>
500-599	<a href="#">Internal DOS error</a>
600-699	<a href="#">Internal Windows error</a>
603	<a href="#">WinCannotEnableInt</a>
605	<a href="#">WinCannotDisableInt</a>
606	<a href="#">WinCantPageLockBuffer</a>
630	<a href="#">NoPCMCARD</a>
801	<a href="#">InvalidGainArrayLength</a>
802	<a href="#">InvalidDimensionOLength</a>
1000	<a href="#">NotTEDSSensor</a>
1001	<a href="#">InvalidTEDSSensor</a>
1002	<a href="#">CalibrationFailed</a>
1003	<a href="#">BitUsedForTerminalCountStatus</a>
1004	<a href="#">PortUsedForTerminalCountStatus</a>



## GlobalConfig class

Contains all of the members for getting global board configuration information.

The GlobalConfig class is a member of the MccDaq namespace. Refer to the "[UL for .NET Class Library Overview](#)" for an explanation of the MccDaq namespace.

## Properties and methods

The GlobalConfig class includes three properties that you can use to examine global board configuration information.

- [NumBoards](#) - Returns the maximum number of boards you can install at one time.
- [NumExpBoards](#) - Returns the maximum number of expansions boards allowed to be installed on the board.
- [Version](#) - This information is used by the library to determine compatibility.

Each of these properties is typed as an integer.

## MccBoard class

The MccBoard class provides access to all of the methods for data acquisition and properties providing board information and configuration for a particular board.

The MccBoard class is a member of the MccDaq namespace. Refer to the "[UL for .NET Class Library Overview](#)" for an explanation of the MccDaq namespace.

## Class constructors

The MccBoard class provides two constructors; one which accepts a board number parameter and one with no arguments.

The following code examples demonstrate how to create a new instance of the MccBoard class using the latter version with a default board number of 0.

### VB .NET

```
Private DaqBoard As MccDaq.MccBoard
DaqBoard = New MccDaq.MccBoard()
```

### C# .NET

```
private MccDaq.MccBoard DaqBoard;
DaqBoard = new MccDaq.MccBoard();
```

The following code examples create a new instance of the MccBoard class with the board number passed to it.

### VB .NET

```
Private DaqBoard As MccDaq.MccBoard
DaqBoard = New MccDaq.MccBoard(BoardNumber)
```

### C# .NET

```
private MccDaq.MccBoard DaqBoard;
DaqBoard = new MccDaq.MccBoard(BoardNumber);
```

## Class Properties

The MccBoard class includes six properties that you can use to examine or change the configuration of your board. The configuration information for all boards is stored in the CB.CFG file, and is loaded from CB.CFG by all programs that use the library.

- [BoardName](#)
- [BoardNum](#)
- [BoardConfig](#)
- [CtrConfig](#)
- [DioConfig](#)
- [ExpansionConfig](#)

## Class Methods

The MccBoard class includes over 100 methods for data acquisition. The MccBoard class methods are equivalents of the function calls used in the standard Universal Library. The MccBoard class methods have virtually the same parameter set as their UL counterparts.

- [Analog I/O Methods](#)
- [Configuration Methods and Properties](#)
- [Counter Methods](#)
- [DataLogger Methods and Property](#)
- [Digital I/O Methods](#)
- [Error Handling Methods and Properties](#)
- [Memory Board Methods](#)
- [Revision Control Methods](#)
- [Streamer File Methods](#)
- [Synchronous I/O Methods](#)

- [Temperature Input Methods](#)
- [Windows Memory Management Methods](#)
- [Miscellaneous Methods](#)

## MccService class

Contains all of the members for calling utility UL functions.

The MccService class is a member of the MccDaq namespace. Refer to the "[UL for .NET Class Library Overview](#)" for an explanation of the MccDaq namespace.

## Methods

The MccService class contains ten static methods. You do not need to create an instance of the MccService class to call these methods.

- [DeclareRevision\(\)](#) - Declares the revision number of the Universal Library for .NET with which the program was written.
- [ErrHandling\(\)](#) - Sets the method of reporting and handling errors for all function calls.
- [FileGetInfo\(\)](#) - Reads streamer file information on how much data is in the file, and the conditions under which it was collected (sampling rate, channels, etc.).
- [FileRead\(\)](#) - Reads a selected number of data points from a streamer file into an array.
- [GetBoardName\(\)](#) - Returns the board name of a specified board.
- [GetRevision\(\)](#) - Returns the revision number of the Universal Library DLL and SSVXD.
- [WinArrayToBuf\(\)](#) - Copies data from an array to a Windows buffer.
- [WinBufAlloc\(\)](#) - Allocates a Windows memory buffer.
- [WinBufAlloc32\(\)](#) - Allocates a Windows memory buffer for use with 32-bit scan functions.
- [WinBufAlloc64\(\)](#) - Allocates a Windows memory buffer large enough to hold double precision data values.
- [WinBufFree\(\)](#) - Free a Windows buffer.
- [WinBufToArray\(\)](#) - Copies data from a Windows buffer to an array.
- [WinBufToArray32\(\)](#) - Copies 32-bit data from a Windows buffer to an array.
- [ScaledWinBufAlloc\(\)](#) - Allocates a Windows global memory buffer large enough to hold scaled data obtained from scan operations in which the ScaleData scan option is selected.
- [ScaledWinBufToArray\(\)](#) - Copies double-precision values from a Windows buffer to an array.



## ACalibrateData() method

Calibrates the raw data collected by [AInScan\(\)](#) from boards with real time software calibration when the real time calibration has been turned off. The AInScan() method can return either raw A/D data or calibrated data, depending on whether or not the NoCalibrateData option was used.

Member of the [MccBoard class](#).

## Function Prototype

### VB .NET

```
Public Function ACalibrateData(ByVal numPoints As Integer, ByVal range As MccDaq.Range, ByVal adData As Short()) As MccDaq.ErrorInfo

Public Function ACalibrateData(ByVal numPoints As Integer, ByVal range As MccDaq.Range, ByVal adData As System.UInt16()) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo ACalibrateData(int numPoints, MccDaq.Range range, short[] adData)

public MccDaq.ErrorInfo ACalibrateData(int numPoints, MccDaq.Range range, ushort[] adData)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

### VB .NET

```
Public Function ACalibrateData(ByVal numPoints As Integer, ByVal range As MccDaq.Range, ByRef adData As Short) As MccDaq.ErrorInfo

Public Function ACalibrateData(ByVal numPoints As Integer, ByVal range As MccDaq.Range, ByRef adData As System.UInt16) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo ACalibrateData(int numPoints, MccDaq.Range, ref ushort adData)

public MccDaq.ErrorInfo ACalibrateData(int numPoints, MccDaq.Range range, ref short adData)
```

## Parameters

### *numPoints*

The number of samples to convert.

### *range*

The programmable gain/range used when the data was collected. Refer to board specific information for a list of the [supported A/D ranges](#) of each board.

### *adData*

Reference to data array.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- adData - Converted data

## Notes

- When collecting data using [AInScan\(\)](#) with the NoCalibrateData option, use this method to calibrate the data after it is collected.

The name of the array must match that used in [AInScan\(\)](#) or [WinBufToArray\(\)](#).

Applying software calibration factors in real time on a per-sample basis eats up machine cycles. If your CPU is slow, or if processing time is at a premium, withhold calibration until after the acquisition run is complete. Turning off real-time software calibration saves CPU time during a high-speed acquisition run.

- Processor speed is a factor for DMA transfers and for real-time software calibration. Processors of less than 150 MHz Pentium class may impose speed limits below the capability of the board (refer to board-specific information.) If your processor is less than a 150 MHz Pentium, and you need an acquisition speed in excess of 200 kHz, use the NoCalibrateData option to turn off real-time software calibration and save CPU time. After the acquisition is run, calibrate the data with ACalibrateData().

## AConvertData() method

Converts the raw data collected by [AInScan\(\)](#) into 12-bit A/D values. The AInScan() method can return either raw A/D data or converted data, depending on whether or not the ConvertData option is used. For many 12-bit A/D boards, the raw data is a 16-bit value that contains a 12-bit A/D value and a 4-bit channel tag (refer to board-specific information in the Universal Library User's Guide). The data returned to adData consists of just the 12-bit A/D value. The data returned to chanTags consists of just the channel numbers.

Member of the [MccBoard class](#).

## Function Prototype

### VB .NET

```
Public Function AConvertData(ByVal numPoints As Integer, ByVal adData As Short(), ByVal chanTags As Short()) As MccDaq.ErrorInfo

Public Function AConvertData(ByVal numPoints As Integer, ByVal adData As System.UInt16(), ByVal chanTags As System.UInt16()) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo AConvertData(int numPoints, short[] adData, short[] chanTags)

public MccDaq.ErrorInfo AConvertData(int numPoints, ushort[] adData, ushort[] chanTags)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

### VB .NET

```
Public Function AConvertData(ByVal numPoints As Integer, ByRef adData As Short, ByRef chanTags As Short) As MccDaq.ErrorInfo

Public Function AConvertData(ByVal numPoints As Integer, ByRef adData As System.UInt16, ByRef chanTags As System.UInt16) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo AConvertData(int numPoints, ref ushort adData, out ushort chanTags)

public MccDaq.ErrorInfo AConvertData(int numPoints, ref short adData, out short chanTags)
```

## Parameters

*numPoints*

Number of samples to convert

*adData*

Reference to start of data array

*chanTags*

Reference to start of channel tag array

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- adData - converted data
- chanTags - channel tags, if available

## Notes

- When collecting data using [AInScan\(\)](#) without the ConvertData option, use this method to convert the data after it has been collected. There are cases where the ConvertData option is not allowed. For example - if you are using both the DmaIo and Background option with AInScan() on some devices, the ConvertData option is not allowed. In those cases, this method should be used to convert the data after the data collection is complete.
- For some boards, each raw data point consists of a 12-bit A/D value with a 4-bit channel number. This method pulls each data point apart and puts the A/D value into the adData array and the channel number into the chanTags array.
- 12-bit A/D boards: The name of the array must match that used in AInScan(). Upon returning from AConvertData(), the adData array contains only 12-bit A/D data.

## AConvertPreTrigData() method

For products with pretrigger implemented in hardware (most products), this function converts the raw data collected by [APretrig\(\)](#). The APretrig() method can return either raw A/D data or converted data, depending on whether or not the ConvertData option was used. The raw data is not in the correct order as it is collected. After the data collection is completed, it must be rearranged into the correct order. This method also orders the data, starting with the first pretrigger data point and ending with the last post-trigger point.

Member of the [MccBoard](#) class.

### Function Prototype

#### VB .NET

```
Public Function AConvertPretrigData(ByVal preTrigCount As Integer, ByVal totalCount As Integer, ByVal adData As Short(), ByVal chanTags As Short()) As MccDaq.ErrorInfo

Public Function AConvertPretrigData(ByVal preTrigCount As Integer, ByVal totalCount As Integer, ByVal adData As System.UInt16(), ByVal chanTags As System.UInt16()) As MccDaq.ErrorInfo
```

#### C# .NET

```
public MccDaq.ErrorInfo AConvertPretrigData(int preTrigCount, int totalCount, short[] adData, short[] chanTags)

public MccDaq.ErrorInfo AConvertPretrigData(int preTrigCount, int totalCount, ushort[] adData, ushort[] chanTags)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

#### VB .NET

```
Public Function AConvertPretrigData(ByVal preTrigCount As Integer, ByVal totalCount As Integer, ByRef adData As Short, ByRef chanTags As Short) As MccDaq.ErrorInfo

Public Function AConvertPretrigData(ByVal preTrigCount As Integer, ByVal totalCount As Integer, ByRef adData As System.UInt16, ByRef chanTags As System.UInt16) As MccDaq.ErrorInfo
```

#### C# .NET

```
public MccDaq.ErrorInfo AConvertPretrigData(int preTrigCount, int totalCount, ref ushort adData, out ushort chanTags)

public AConvertPretrigData(int preTrigCount, int totalCount, ref short adData, out short chanTags)
```

### Parameters

#### *preTrigCount*

Number of pre-trigger samples (this value must match the value returned by the PretrigCount parameter in the APretrig() method).

#### *totalCount*

Total number of samples that were collected.

#### *adData*

Reference to data array (must match array name used in APretrig() method).

#### *chanTags*

A pointer to the start of the channel tag array (if available). Returns NULL if using a 16-bit board or if channel tags are not available. Refer to the note regarding [16-bit A/D boards](#) below.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- adData - converted data

### Notes

- When you collect data with APretrig() and you don't use the ConvertData option, you must use this method 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 APretrig() on some devices, the ConvertData option is not allowed. In those cases this method should be used to convert the data after the data collection is complete.

- 12-Bit A/D boards: On some 12-bit boards, each raw data point consists of a 12-bit A/D value with a 4-bit channel number. This method pulls each data point apart and puts the A/D value into the adData and the channel number into the chanTags array.

Upon returning from AConvertPretrigData(), adData array contains only 12-bit A/D data.

- 16-Bit A/D boards: This method is for use with 16-bit A/D boards only insofar as ordering the data. No channel tags are returned.

The name of the ADData array must match that used in [AInScan\(\)](#) or [WinBufToArray\(\)](#).

- VB .Net Programmers: After the data is collected with [APretrig\(\)](#), it must be copied to a BASIC array with [WinBufToArray\(\)](#).

**Important:** The entire array must be copied, which includes the extra 512 samples needed by APretrig(). Example code is provided here:

```
SampleCount% = 10000  
  
Dim A_D_Data%(SampleCount% + 512)  
Dim Chan_Tags%(SampleCount% + 512)  
  
APretrig%(LowChan, HighChan, PretrigCount%, SampleCount%...)  
  
WinBufToArray%(MemHandle%, A_D_Data%, SampleCount% + 512)  
AConvertPretrigData%(Pretrig_Count%, SampleCount%, A_D_Data%, Chan_Tags%)
```

## AIn() method

Reads an A/D input channel, and returns a 16-bit integer value. This method 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.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function AIn(ByVal channel As Integer, ByVal range As MccDaq.Range, ByRef dataValue As Short) As MccDaq.ErrorInfo

Public Function AIn(ByVal channel As Integer, ByVal range As MccDaq.Range, ByRef dataValue As System.UInt16) As MccDaq.ErrorInfo
```

C# .NET

```
MccDaq.ErrorInfo AIn(int channel, MccDaq.Range range, out ushort dataValue)

public MccDaq.ErrorInfo AIn(int channel, MccDaq.Range range, out short dataValue)
```

## Parameters

*channel*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured. For example, a USB-1608GX device has 8 differential or 16 single-ended analog input channels. Expansion boards also support this method, so this parameter can contain values up to 272. See board specific information for EXP boards if you are using an expansion board.

*range*

A/D range code. If the selected A/D board does not have a programmable gain feature, this parameter is ignored. If the A/D board does have programmable gain, set the range parameter to the desired A/D range. Refer to board specific information for a list of the supported A/D ranges of each board.

*dataValue*

Reference to data value.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataValue - The value of the A/D sample.

## AIn32() method

Reads an A/D input channel from the specified board, and returns a 32-bit integer value. 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. In general, this function should be used with devices with a resolution higher than 16-bits.

Member of the [MccBoard](#) class.

## Function Prototype

### VB .NET

```
Public Function AIn32(ByVal channel As Integer, ByVal range As MccDaq.Range, ByRef dataValue As Integer, ByVal options As Integer) As MccDaq.ErrorInfo

Public Function AIn32(ByVal channel As Integer, ByVal range As MccDaq.Range, ByRef dataValue As UInteger, ByVal options As Integer) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo AIn32(int channel, MccDaq.Range range, out uint dataValue, int options)

public MccDaq.ErrorInfo AIn32(int channel, MccDaq.Range range, out int dataValue, int options)
```

## Parameters

### *channel*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured. For example, a USB-2416 device has 16 differential or 32 single-ended analog input channels. Expansion boards also support this method, so this parameter can contain values up to 272. See board specific information for EXP boards if you are using an expansion board.

### *range*

A/D range code. If the selected A/D board does not have a programmable gain feature, this parameter is ignored. If the A/D board does have programmable gain, set the range parameter to the desired A/D range. Refer to board specific information for a list of the supported A/D ranges of each board.

### *dataValue*

Pointer or reference to data value.

### *options*

Reserved for future use.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataValue - The value of the A/D sample.

## AInScan() method

Scans a range of A/D channels and stores the samples in an array. AInScan() 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 the data array.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function AInScan(ByVal lowChan As Integer, ByVal highChan As Integer, ByVal numPoints As Integer, ByRef rate As Integer, ByVal range As MccDaq.Range, ByVal memHandle As IntPtr, ByVal options As MccDaq.ScanOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo AInScan(int lowChan, int highChan, int numPoints, ref int rate, MccDaq.Range range, IntPtr memHandle, MccDaq.ScanOptions options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function AInScan(ByVal lowChan As Integer, ByVal highChan As Integer, ByVal numPoints As Integer, ByRef rate As Integer, ByVal range As MccDaq.Range, ByVal memHandle As Integer, ByVal options As MccDaq.ScanOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo AInScan(int lowChan, int highChan, int numPoints, ref int rate, MccDaq.Range range, int memHandle, MccDaq.ScanOptions options)
```

## Parameters

*lowChan*

First A/D channel of the scan. When [ALoadQueue\(\)](#) is used, the channel count is determined by the total number of entries in the channel gain queue. lowChan is ignored.

*highChan*

Last A/D channel of the scan. When [ALoadQueue\(\)](#) is used, the channel count is determined by the total number of entries in the channel gain queue. highChan is ignored.

low / high 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 USB-1208FS has four channels for differential, eight for single-ended.

*numPoints*

Number of A/D samples to collect. Specifies the total number of A/D samples to collect. If more than one channel is being sampled, then the number of samples collected per channel is equal to count ÷ (highChan - lowChan + 1).

*rate*

The rate at which samples are acquired, in samples per second per channel.

For example, sampling four channels, 0-3, at a rate of 10,000 scans per second (10 kilohertz (kHz)) results in an A/D converter rate of 40 kHz: four channels at 10,000 samples per channel per second. With other software, 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.

The channel count is determined by the lowChan and highChan parameters. Channel Count = (highChan - lowChan + 1).

When [ALoadQueue\(\)](#) is used, the channel count is determined by the total number of entries in the channel gain queue. lowChan and highChan are ignored.

rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*range*

A/D range code. If the selected A/D board does not have a programmable range feature, this parameter is ignored. Otherwise, set the range parameter to any range that is supported by the selected A/D board. Refer to board-specific information for a list of the [supported A/D ranges](#) of each board.

*memHandle*

Handle for Windows buffer to store data. This buffer must have been previously allocated with [WinBufAllocEx\(\)](#), [WinBufAlloc32Ex\(\)](#), or [ScaledWinBufAllocEx\(\)](#).

## options

Bit fields that control various options. Refer to the constants in the "[options parameter values](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- rate - actual sampling rate used.
- memHandle - collected A/D data returned via the Windows buffer.

## options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ScanOptions enumeration (variable = MccDaq.ScanOptions.SingleIo, variable = MccDaq.ScanOptions.DmaIo, etc.).

Transfer method options	<p>The following four options determine how data is transferred from the board to PC memory. If none of these options are specified (recommended), the optimum sampling mode is automatically chosen based on board type and sampling speed. Use the default method unless you have a reason to select a specific transfer method.</p> <ul style="list-style-type: none"><li>■ SingleIo A/D transfers to memory are initiated by an interrupt. One interrupt per conversion. Rates attainable using SingleIo are PC-dependent and generally less than 10 kHz.</li><li>■ DmaIo A/D transfers are initiated by a DMA request.</li><li>■ BlockIo A/D transfers are handled in blocks (by REP-INSW for example). <b>BlockIo is not recommended for slow acquisition rates.</b> If the rate of acquisition is very slow, (for example less than 200 Hz), BlockIo is probably not the best choice for transfer mode. The reason for this is that status for the operation is not available until one packet of data has been collected (typically 512 samples). The implication is that if acquiring 100 samples at 100 Hz using BlockIo, the operation will not complete until 5.12 seconds has elapsed.</li><li>■ BurstIo Allows higher sampling rates for sample counts up to full FIFO. Data is collected into the local FIFO. Data transfers to the PC are held off until after the scan is complete. For Background scans, the count and index returned by <a href="#">GetStatus()</a> remain 0 and the status equals Running until the scan finishes. When the scan is complete and the data is retrieved, the count and index are updated and the status equals Idle. BurstIo is the default mode for non-Continuous fast scans (aggregate sample rates above 1000 Hz) with sample counts up to full-FIFO. To avoid the BurstIo default, specify BlockIo. BurstIo is not a valid option for most boards. It is used mainly for USB products. BURSTIO is not a valid option for most boards. It is used mainly for USB products.</li></ul>
Background	<p>If the Background option is not used, the <a href="#">AInScan()</a> method will not return control to your program until all of the requested data has been collected and returned to the buffer. 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 the buffer will continue in the background. Use <a href="#">GetStatus()</a> with AiFunction to check on the status of the background operation. Alternatively, some boards support <a href="#">EnableEvent()</a> for event notification of changes in status of Background scans. Use <a href="#">StopBackground()</a> with AiFunction to stop the background process before it has completed. StopBackground() should be executed after normal termination of all background methods in order to clear variables and flags.</p>
BurstMode	<p>Enables burst mode sampling. Scans from lowChan to highChan are 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 the <i>rate</i> parameter.</p> <p>BurstMode is not recommended for use with the SingleIo option. If this combination is used, the count value should be set as low as possible, preferably to the number of channels in the scan. Otherwise, overruns may occur.</p>
ConvertData	<p>This option is used to align data, either within each byte (in the case of some 12-bit devices) or within the buffer (see the <a href="#">APreTrig()</a> method). Only the former case applies for AInScan. The data stored on some 12-bit devices is offset in the devices data register. For these devices, the ConvertData option converts the data to 12-bit A/D values by shifting the data to the first 12 bits within the byte. For devices that store the data without an offset and for all 16-bit devices, this option is ignored.</p> <p>Use of ConvertData is recommended unless one of the following two conditions exist: 1) On some devices, ConvertData may not be specified if you are using the Background option</p>



	and DMA transfers. In this case, if data conversion is required, use <a href="#">AConvertData()</a> to realign the data. 2) Some 12-bit boards store the data as a 12-bit A/D value and a 4-bit channel number. Using ConvertData will strip out the channel number from the data. If you prefer to store the channel number as well as the data, call AConvertData() to retrieve the data and the channel number from the buffer after the data acquisition to the buffer is complete.
Continuous	<p>This option puts the method in an endless loop. Once it collects the required number of samples, it resets to the start of the buffer and begins again. The only way to stop this operation is with <a href="#">StopBackground()</a>. Normally this option should be used in combination with Background so that your program will regain control.</p> <p><b>numPoints parameter settings in Continuous mode:</b> For some DAQ hardware, numPoints must be an integer multiple of the packet size. Packet size is the amount of data that a DAQ device transmits back to the PC's memory buffer during each data transfer. Packet size can differ among DAQ hardware, and can even differ on the same DAQ product depending on the transfer method.</p> <p>In some cases, the minimum value for the numPoints parameter may change when the Continuous option is used. This can occur for several reasons; the most common is that in order to trigger an interrupt on boards with FIFOs, the circular buffer must occupy at least half the FIFO. Typical half-FIFO sizes are 256, 512 and 1,024.</p> <p>Another reason for a minimum numPoints value is that the buffer in memory must be periodically transferred to the user buffer. If the buffer is too small, data is overwritten during the transfer resulting in garbled data.</p> <p>Refer to board-specific information in the <i>Universal Library User's Guide</i> for packet size information for your particular DAQ hardware.</p>
DTConnect	All A/D values will be sent to the A/D board's DT-Connect port. This option is incorporated into the ExtMemory option. Use DTConnect only if the external board is not supported by the Universal Library.
ExtClock	<p>If this option is used then conversions will be controlled by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal (refer to the board-specific information contained in the <i>Universal Library User's Guide</i>). In most cases, when this option is used the rate parameter is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to a transfer mode that will allow the maximum conversion rate to be attained unless otherwise specified.</p> <p>In some cases, such as with the PCI-DAS4020/12, an approximation of the rate is used to determine the size of the packets to transfer from the board. Set the rate parameter to an approximate maximum value.</p> <p><b>SingleIo is recommended for slow external clock rates:</b> If the rate of the external clock is very slow (for example less than 200 Hz) and the board you are using supports BlockIo, you may want to include the SingleIo option. This is because the status for the operation is not available until one packet of data has been collected (typically 512 samples). The implication is that, if acquiring 100 samples at 100 Hz using BlockIo (the default for boards that support it if ExtClock is used), the operation will not complete until 5.12 seconds has elapsed.</p>
ExtMemory	Causes the command to send the data to a connected memory board via the DT-Connect interface rather than returning the data to the buffer. Data for each call to this method will be appended unless <a href="#">MemReset()</a> is called. The data should be unloaded with the <a href="#">MemRead()</a> method before collecting new data. When ExtMemory option is used, the reference to the buffer (memHandle) may be set to null or 0. Continuous option cannot be used with ExtMemory. Do not use ExtMemory and DtConnect together. The transfer modes DmaIo, SingleIo and BlockIo have no meaning when used with this option.
ExtTrigger	<p>If this option is specified, the sampling will not begin until the trigger condition is met. On many boards, this trigger condition is programmable (refer to <a href="#">SetTrigger()</a> and to board-specific info for details). On other boards, only 'polled gate' triggering is supported. Assuming active high operation, data acquisition will commence immediately if the trigger input is high. If the trigger input is low, acquisition will be held off until it goes high, and then continue until numPoints samples are taken, regardless of the state of the trigger input.</p> <p>This option is most useful if the signal is a pulse with a very low duty cycle (trigger signal in TTL low state most of the time) so that triggering will be held off until the occurrence of the pulse.</p>
HighResRate	Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>rate</i> parameter above).
NoCalibrateData	<p>Turns off real-time software calibration for boards which are software calibrated, by applying calibration factors to the data on a sample by sample basis as it is acquired. Examples are the PCM-DAS16/330 and PCM-DAS16x/12.</p> <p>Turning off software calibration saves CPU time during a high speed acquisition run. This may be required if your processor is less than a 150 MHz Pentium and you desire an acquisition speed in excess of 200 kHz. These numbers may not apply to your system. Only</p>

	trial will tell for sure. DO NOT use this option if you do not have to. If this option is used, the data must be calibrated after the acquisition run with the <code>ACalibrateData()</code> method.
NoToDints	<p>Disables the system's time-of-day interrupts for the duration of the scan. These interrupts are used to update the systems real time clock and are also used by various other programs.</p> <p>These interrupts can limit the maximum sampling speed of some boards - particularly the PCM-DAS08. If the interrupts are turned off using this option, the real-time clock will fall behind by the length of time that the scan takes.</p>
RetrigMode	<p>Re-arms the trigger after a trigger event is performed. With this mode, the scan begins when a trigger event occurs. When the scan completes, the trigger is re-armed to acquire the next the batch of data. You can specify the number of samples in the scan for each trigger event (described below). The RetrigMode option can be used with the Continuous option to continue arming the trigger until <code>StopBackground()</code> is called.</p> <p>You can specify the number of samples to acquire with each trigger event. This is the trigger count (<code>retrigCount</code>). Use <a href="#">SetAdRetrigCount()</a> to set the trigger count. If you specify a trigger count that is either zero or greater than the value of the <code>AInScan()</code> <code>numPoints</code> argument, the trigger count is set to the value of <code>numPoints</code>.</p> <p>Specify the Continuous option with the trigger count set to zero to fill the buffer with <code>numPoints</code> samples, re-arm the trigger, and refill the buffer upon the next trigger.</p>
ScaleData	<p>Converts raw scan data — to voltage, temperature, and so on, depending upon the selected channel sensor category — during the analog input scan, and puts the scaled data directly into the user buffer. The user buffer should have been allocated with <a href="#">ScaledWinBufAllocEx()</a>.</p>

### 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 `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  $8 \times 20,000 = 160,000$ , but the CIO-DAS16/JR is capable of converting only 120,000 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.

### Important!

In order to understand the functions, you must read the board-specific information contained in the in the *Universal Library User's Guide*. Examine and run the example programs before attempting your own program. Following this advice will save you hours of frustration, and possibly time wasted holding for technical support.

This note, which appears elsewhere, is especially applicable to this function. Refer to the board-specific information for your hardware in the *Universal Library User's Guide*. We suggest that you make a copy of that page for reference as you read this manual and examine the example programs.

## ALoadQueue() method

Loads the A/D board's channel/gain queue. This method only works with A/D boards that have channel/gain queue hardware.

Some products do not support channel/gain queue, and some that do support it are limited on the order of elements, number of elements, and gain values that can be included, etc. Please refer to the device-specific information in the *Universal Library User's Guide* for details of your particular product.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function ALoadQueue(ByVal chanArray As Short(), ByVal gainArray As MccDaq.Range(), ByVal count As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo ALoadQueue(short[] chanArray, MccDaq.Range[] gainArray, int count)
```

## Parameters

*chanArray*

Array containing channel values. This array should contain all of the channels that will be loaded into the channel gain queue.

*gainArray*

Array containing A/D range values. This array should contain each of the A/D ranges that are loaded into the channel gain queue.

*count*

Number of elements in *chanArray* and *gainArray* or 0 to disable channel/gain queue. Specifies the total number of channel/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 channel/gain queue. The maximum value is specific to the queue size of the A/D boards channel gain queue.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- Normally the [AInScan\(\)](#) method 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 method then all subsequent calls to [AInScan\(\)](#) will cycle through the channel/gain pairs that you have loaded into the queue.

## AOut() method

Sets the value of a D/A output.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function AOut(ByVal channel As Integer, ByVal range As MccDaq.Range, ByVal dataValue As Short) As MccDaq.ErrorInfo
```

```
Public Function AOut(ByVal channel As Integer, ByVal range As MccDaq.Range, ByVal dataValue As System.UInt16) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo AOut(int channel, MccDaq.Range range, ushort dataValue)
```

```
public MccDaq.ErrorInfo AOut(int channel, MccDaq.Range range, short dataValue)
```

## Parameters

*channel*

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

*range*

D/A range code. The output range of the D/A channel can be set to any of those supported by the board. If the D/A board does not have programmable ranges then this parameter is ignored.

*dataValue*

Value to set D/A to. Must be in the range 0 - N where N is the value  $2^{\text{Resolution}} - 1$  of the converter.

Exception: using 16-bit boards with Basic range is -32,768 to 32,767. Refer to the discussion on [16-bit values using a signed integer data type](#) for more information.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- Simultaneous Update Boards: If you set the simultaneous update jumper for simultaneous operation, use [AOutScan\(\)](#) for simultaneous update of multiple channels. AOut() always writes the D/A data then reads the D/A, which causes the D/A output to be updated.

## AOutScan() method

Outputs values to a range of D/A channels. This method can be used for paced analog output on hardware that supports paced output. It can also be used to update all analog outputs at the same time when the Simultaneous option is used.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function AOutScan(ByVal lowChan As Integer, ByVal highChan As Integer, ByVal numPoints As Integer, ByRef rate As Integer, ByVal range As MccDag.Range, ByVal memHandle As IntPtr, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo AOutScan(int lowChan, int highChan, int numPoints, ref int rate, MccDag.Range range, IntPtr memHandle, MccDag.ScanOptions options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function AOutScan(ByVal lowChan As Integer, ByVal highChan As Integer, ByVal numPoints As Integer, ByRef rate As Integer, ByVal range As MccDag.Range, ByVal memHandle As Integer, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo AOutScan(int lowChan, int highChan, int numPoints, ref int rate, MccDag.Range range, int memHandle, MccDag.ScanOptions options)
```

## Parameters

*lowChan*

First D/A channel of scan.

*highChan*

Last D/A channel of scan.

lowChan/highChan: The maximum allowable channel depends on which type of D/A board is being used.

*numPoints*

Number of D/A values to output. Specifies the total number of D/A values that will be output. Most D/A boards do not support timed outputs. For these boards, set the count to the number of channels in the scan.

*rate*

Sample rate in scans per second. For many 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 CIO-DAC04/12-HS, rate should be set to the D/A output rate (in scans/sec). This parameter also returns the value of the actual rate set. This value may be different from the user specified rate because of pacer limitations.

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 (10 kHz) will result in the D/A converter rates of 10 kHz: (one D/A per channel). The data transfer rate is 40,000 words per second; 4 channels \* 10,000 updates per scan.

The maximum update rate depends on the D/A board that is being used, and the sampling mode options.

*range*

D/A range code. The output range of the D/A channel can be set to any of those supported by the board. If the D/A board does not have a programmable gain, then this parameter is ignored.

*memHandle*

Handle for Windows buffer from which data is output. This buffer must have been previously allocated with the [WinBufAlloc\(\)](#) method and data values loaded (perhaps using [WinArrayToBuf\(\)](#)).

*options*

Bit fields that control various options. Refer to the constants in the [options Parameter Values](#) section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- Rate - actual sampling rate used.

## options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ScanOptions enumeration (variable = MccDaq.ScanOptions.Continuous, variable = MccDaq.ScanOptions.Background, etc.).

ADCClock	Paces the data output operation using the ADC clock.
ADCClockTrig	Triggers a data output operation when the ADC clock starts.
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 <a href="#">GetStatus()</a> with AoFunction to check the status of background operation. Alternatively, some boards support <a href="#">EnableEvent()</a> for event notification of changes in status of Background scans. Use <a href="#">StopBackground()</a> with AoFunction to terminate background operations before they are completed. StopBackground() should be executed after normal termination of all background methods in order to clear variables and flags.
Continuous	This option may only be used with boards which support interrupt, DMA or REP INSW transfer methods. This option puts the method in an endless loop. Once it outputs the specified number (numPoints) of D/A values, it resets to the start of the buffer and begins again. The only way to stop this operation is by calling <a href="#">StopBackground()</a> with AoFunction. This option should only be used in combination with Background so that your program can regain control.
ExtClock	If this option is used then conversions will be paced by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal (refer to board-specific information contained in the <i>Universal Library Users Guide</i> ). When this option is used the Rate parameter is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to transfer types that allow the maximum conversion rate to be attained unless otherwise specified.
ExtTrigger	If this option is specified the sampling will not begin until the trigger condition is met. On many boards, this trigger condition is programmable (see <a href="#">SetTrigger()</a> method and board-specific information in the <i>UL Users Guide</i> for details).
NonStreamedIO	<p>When this option is used, you can output non-streamed data to a specific DAC output channel. The aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. This allows the data output buffer to be loaded into the device's internal output FIFO. Once the sample updates are transferred or downloaded to the device, the device is responsible for outputting the data. You can't make any changes to the output buffer once the output begins.</p> <p>With NonStreamedIO mode, you do not have to periodically feed output data through the program to the device for the data output to continue. However, the size of the buffer is limited.</p> <p>NonStreamedIO can only be used with the number of samples (numPoints) set equal to the size of the FIFO or less.</p>
RetrigMode	<p>Re-arms the trigger after a trigger event is performed. With this mode, the scan begins when a trigger event occurs. When the scan completes, the trigger is re-armed to generate the next the batch of data. You can specify the number of samples to generate for each trigger event (described below). The RetrigMode option can be used with the Continuous option to continue arming the trigger until StopBackground() is called.</p> <p>You can specify the number of samples to generate with each trigger event. This is the trigger count (retrigCount). Use <a href="#">SetDACRetrigCount()</a> to set the trigger count. If you specify a trigger count that is either zero or greater than the value of the numPoints argument, the trigger count will be set to the value of numPoints.</p>
ScaleData	Gets scaled data, such as voltage, temperature, and so on, from the user buffer, and converts it to raw data. The user buffer should have been allocated with <a href="#">ScaledWinBufAlloc()</a> .
Simultaneous	When this option is used (if the board supports it 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 updated. 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.

## 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 lowChan = 0, highChan = 3 (four 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.

## APreTrig() method

Waits for a trigger to occur and then returns a specified number of analog samples before and after the trigger occurred. If only 'polled gate' triggering is supported, the trigger input line (refer to the User's Guide for the hardware) must be at TTL low before this method is called, or a [TrigState](#) error will occur. The trigger occurs when the trigger condition is met. Refer to the [SetTrigger\(\)](#) method for more details.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function APretrig(ByVal lowChan As Integer, ByVal highChan As Integer, ByRef pretrigCount As Integer, ByRef totalCount As Integer, ByRef rate As Integer, ByVal range As MccDag.Range, ByVal memHandle As IntPtr, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo APretrig(int lowChan, int highChan, ref int pretrigCount, ref int totalCount, ref int rate, MccDag.Range range, IntPtr memHandle, MccDag.ScanOptions options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function APretrig(ByVal lowChan As Integer, ByVal highChan As Integer, ByRef pretrigCount As Integer, ByRef totalCount As Integer, ByRef rate As Integer, ByVal range As MccDag.Range, ByVal memHandle As Integer, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo APretrig(int lowChan, int highChan, ref int pretrigCount, ref int totalCount, ref int rate, MccDag.Range range, int memHandle, MccDag.ScanOptions options)
```

## Parameters

*lowChan*

First A/D channel of scan.

*highChan*

Last A/D channel of scan.

lowChan/highChan: The maximum allowable channel depends on which type of A/D board is being used. For boards with both single-ended and differential inputs, the maximum allowable channel number also depends on how the board is configured (for example, eight channels for differential inputs, 16 for single-ended inputs).

*pretrigCount*

Number of pre-trigger A/D samples to collect. Specifies the number of samples to collect before the trigger occurs.

For products using a hardware implementation of pretrigger (most products), pretrigCount must be less than the (totalCount - 512). For these devices, if the trigger occurs too early, fewer than the requested number of pre-trigger samples are collected, and a [TOOFEW](#) error occurs. The pretrigCount is set to indicate how many samples were actually collected. The post trigger samples will still be collected.

For software implementations of pretrigger, pretrigCount must be less than totalCount. For these devices, triggers that occur before the requested number of pre-trigger samples are collected are ignored. See board-specific information.

*totalCount*

Total number of A/D samples to collect. Specifies the total number of samples that will be collected and stored in the buffer.

For products using a hardware implementation of pretrigger (most products), totalCount must be greater than or equal to the pretrigCount + 512. If the trigger occurs too early, fewer than the requested number of samples will be collected, and a TooFew error will occur. The totalCount will be set to indicate how many samples were actually collected.

For software implementations of pretrigger, totalCount must be greater than pretrigCount. For these devices, triggers that occur before the requested number of pre-trigger samples are collected are ignored. See board-specific information.

totalCount must be evenly divisible by the number of channels being scanned. If it is not, this method will adjust the number (down) to the next valid value and return that value to the totalCount parameter.

pretrigCount must also be evenly divisible by the number of channels being scanned. If it is not, this function will adjust the number (up) to the next valid value and return that value to the pretrigCount parameter.

*rate*

Sample rate in scans per second.

*range*

A/D Range code. If the selected A/D board does not have a programmable gain feature, this parameter is ignored. Otherwise, set to any range that is supported by the selected A/D board. Refer to board specific information for a list of the supported A/D ranges of each board.

#### memHandle

Handle for Windows buffer to store data. This buffer must have been previously allocated with the [WinBufAlloc\(\)](#) method.

For hardware trigger types, the buffer referenced by memHandle must be big enough to hold at least totalCount + 512 integers.

#### options

Bit fields that control various options. Refer to the constants in the [options Parameter Values](#) section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- pretrigCount - Number of pre-trigger samples.
- totalCount - Total number of samples collected.
- rate - actual sampling rate.
- memHandle - Collected A/D data returned via the Windows buffer.

## options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ScanOptions enumeration (variable = MccDaq.ScanOptions.DtConnect, variable = MccDaq.ScanOptions.ExtMemory, etc.).

Background	<p>If the Background option is not used, the APretrig() method will not return to your program until all of the requested data has been collected and returned to the buffer. When the Background option is used, control returns immediately to the next line in your program, and the data collection from the A/D into the buffer will continue in the background. Use <a href="#">GetStatus()</a> with AiFunction to check on the status of the background operation. Alternatively, some boards support <a href="#">EnableEvent()</a> for event notification of changes in status of BACKGROUND scans. Use <a href="#">StopBackground()</a> with AiFunction to terminate the background process before it has completed.</p> <p>Call StopBackground() after normal termination of all background methods to clear variables and flags.</p> <p>For hardware trigger types, you cannot use the ConvertData option in combination with the Background option for this method. To correctly order and parse the data, use <a href="#">AConvertPretrigData()</a> after the function completes.</p>
ConvertData	<p>For hardware trigger types, the data is collected into a "circular" buffer. The ConvertData option is used to align data within the buffer when the data acquisition is complete. This option is ignored for all 16-bit devices, and for 12-bit devices that store the data without an offset (refer to <a href="#">AInScan()</a>). Note that you can also call <a href="#">AConvertPretrigData()</a> to align data within the buffer when the data acquisition is complete.</p> <p>&gt;Use of ConvertData is recommended unless one of the following two conditions exist: 1) On some devices, ConvertData may not be specified if you are using the Background option and DMA transfers. In this case, if data conversion is required, use <a href="#">AConvertData()</a> to re-align the data. 2) Some 12-bit boards store the data as a 12-bit A/D value and a 4-bit channel number. Using ConvertData will strip out the channel number from the data. If you prefer to store the channel number as well as the data, call AConvertData() to retrieve the data and the channel number from the buffer after the data acquisition to the buffer is complete.</p> <p>The ConvertData option is not required for software triggered types.</p>
ExtClock	<p>This option is available only for boards that have separate inputs for external pacer and external trigger. See your hardware manual or refer to the board-specific information in the <i>UL Users Guide</i>.</p>
ExtMemory	<p>Causes this method to send the data to a connected memory board via the DT-Connect interface rather than returning the data to the buffer. If you use this option to send the data to a MEGA-FIFO memory board, then you must use <a href="#">MemReadPretrig()</a> to later read the pre-trigger data from the memory board. If you use <a href="#">MemRead()</a>, the data will NOT be in the correct order.</p> <p>Every time this option is used, it overwrites any data already stored in the memory board. All data should be read from the board (with <a href="#">MemReadPretrig()</a>) before collecting any new data. When this option is used, the memHandle parameter is ignored. The MEGA-FIFO memory must be fully populated in order to use the APretrig() method with the ExtMemory option.</p>
DtConnect	<p>When the DtConnect option is used with this method the data from ALL A/D conversions is sent out the DT-Connect interface. While this method is waiting for a trigger to occur, it will</p>



	send data out the DT-Connect interface continuously. If you have a Measurement Computing memory board plugged into the DT-Connect interface, then you should use the ExtMemory option rather than this option.
--	--

**Important!**

For hardware trigger types, the buffer referenced by memHandle must be big enough to hold at least totalCount + 512 integers.

## ATrig() method

Waits for a specified analog input channel to go above or below a specified value. ATrig() continuously reads the specified channel and compares its value to trigValue. Depending on whether trigType is set to TrigAbove or TrigBelow, it waits for the first A/D sample that is above or below trigValue. The first sample that meets the trigger criteria is returned to dataValue.

Member of the [MccBoard](#) class.

## Function Prototype

### VB .NET

```
Public Function ATrig(ByVal chan As Integer, ByVal trigType As MccDaq.TriggerType, ByVal trigValue As Short, ByVal range As MccDaq.Range, ByRef dataValue As Short) As MccDaq.ErrorInfo
```

```
Public Function ATrig(ByVal chan As Integer, ByVal trigType As MccDaq.TriggerType, ByVal trigValue As System.UInt16, ByVal range As MccDaq.Range, ByRef dataValue As System.UInt16) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo ATrig(int chan, MccDaq.TriggerType trigType, short trigValue, MccDaq.Range range, out short dataValue)
```

```
public MccDaq.ErrorInfo ATrig(int chan, MccDaq.TriggerType trigType, ushort trigValue, MccDaq.Range range, out ushort dataValue)
```

## Parameters

### *chan*

A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with 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 eight channels for differential inputs and 16 channels for single-ended inputs.

### *trigType*

MccDaq.TriggerType.TrigAbove or MccDaq.TriggerType.TrigBelow. Specifies whether to wait 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. Must be in the range 0 – 4,095 for 12-bit A/D boards, or 0-65,535 for 16-bit A/D boards. Refer to your BASIC manual for information on signed BASIC integer data types.

### *range*

Gain code. If the selected A/D board does not have a programmable gain feature, this parameter is ignored. Otherwise, set to any range that is supported by the selected A/D board. Refer to board specific information for a list of the supported A/D ranges of each board.

### *dataValue*

Returns the value of the first A/D sample to meet the trigger criteria.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataValue - value of the first A/D sample to match the trigger criteria.

## Notes

- Ctrl-C will not terminate the wait for an analog trigger that meets the specified condition. There are only two ways to terminate this call: satisfy the trigger condition, or to reset the computer.

## Caution!

Use caution when using this method in Windows programs. All active windows will lock on the screen until the trigger condition is satisfied. All keyboard and mouse activity will also lock until the trigger condition is satisfied.

# VIn() method

Reads an A/D input channel, and returns a single precision voltage value. If the specified A/D board has programmable gain, then this function sets the gain to the specified range. The voltage value is returned to dataValue.

Member of the [MccBoard](#) class.

## Function Prototype

```
VB .NET
Public Function VIn(ByVal channel As Integer, ByVal range As MccDag.Range, ByRef dataValue As Single,
    ByVal options as MccDag.VInOptions) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo VIn(System.Int32 channel, MccDag.Range range, System.Single dataValue,
MccDag.VInOptions options)
```

## Parameters

- channel*  
A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured.
- range*  
A/D range code. If the board has a programmable gain, it will be set according to this parameter value.  
Keep in mind that some A/D boards have a programmable gain feature, and others set the gain via switches on the board. In either case, the range that the board is configured for must be passed to this method. Refer to board-specific information in the *Universal Library User's Guide* for a list of the [supported A/D ranges](#) of each board.
- dataValue*  
A reference to the data value.
- options*  
Reserved for future use.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataValue - The value in volts of the A/D sample.

### options parameter values

Default	Reserved for future use.
---------	--------------------------

# VIn32() method

Reads an A/D input channel, and returns a double precision voltage value. If the specified A/D board has programmable gain, then this function sets the gain to the specified range. The voltage value is returned to dataValue.

Member of the [MccBoard](#) class.

## Function Prototype

```
VB .NET
Public Function VIn32(ByVal channel As Integer, ByVal range As MccDag.Range, ByRef dataValue As Double,
    ByVal options As MccDag.VInOptions) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo VIn32(int channel, MccDag.Range range, out double dataValue, MccDag.VInOptions
    options)
```

## Parameters

- channel*  
A/D channel number. The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured.
- range*  
A/D range code. If the board has a programmable gain, it will be set according to this parameter value.  
Keep in mind that some A/D boards have a programmable gain feature, and others set the gain via switches on the board. In either case, the range that the board is configured for must be passed to this method. Refer to board-specific information in the *Universal Library User's Guide* for a list of the supported A/D ranges of each board.
- dataValue*  
A reference to the data value.
- options*  
Reserved for future use.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataValue - The value in volts of the A/D sample.

### options parameter values

Default	Reserved for future use.
---------	--------------------------

# VOut() method

Sets the value of a D/A channel. This method cannot be used for current output.

Member of the [MccBoard](#) class.

## Function Prototype

```
VB .NET
Public Function VOut(ByVal channel As Integer, ByVal range As MccDag.Range, ByVal dataValue As Single,
ByVal options As MccDag.VOutOptions) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo VOut(System.Int32 channel, MccDag.Range range, System.Single dataValue,
MccDag.VOutOptions options)
```

## Parameters

- channel*  
The D/A channel number. The maximum allowable channel depends on which type of D/A board is being used.
- range*  
The D/A range code. If the device has a programmable gain, it is set according to this parameter value. If the range specified isn't supported, the function return a BADRANGE error.  
  
If the gain is fixed or manually selectable, make sure that this parameter matches the gain configured for the device. If it doesn't, the output voltage will not match the voltage specified in the dataValue parameter.
- dataValue*  
The voltage value to be written.
- options*  
Reserved for future use.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### options parameter values

Default	Reserved for future use.
---------	--------------------------

## BoardConfig property

Represents an instance of the [cBoardConfig](#) class. Use this property to call the board configuration methods.

Member of the [MccBoard](#) class.

## Property prototype

VB .NET

```
Public ReadOnly Property BoardConfig As MccDaq.cBoardConfig
```

C# .NET

```
public MccDaq.cBoardConfig BoardConfig [get]
```

## Methods

Over 20 UL for .NET configuration methods are accessible only from the BoardConfig property. Before you call any of these methods, you need to create an instance of an MccBoard object.

```
Dim MyBoard As MccDaq.MccBoard
```

```
MyBoard = New MccDaq.MccBoard(MyBoardNum)
```

To call a method from the BoardConfig property, use the notation shown in the example below.

```
MyErrorInfo = MyBoard.BoardConfig.GetBoardType(MyBoardType)
```

## BoardConfig.DACUpdate() method

Updates the voltage values on analog output channels. This method is usually called after a [SetDACUpdateMode\(\)](#) method call with its configVal parameter set to 1 (*on command*).

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function prototype

VB .NET

```
Public Function DACUpdate() As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo DACUpdate()
```

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetAdRetrigCount() method

Gets the number of samples to acquire during each trigger event when ScanOptions.RetrigMode is enabled.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetAdRetrigCount(ByRef retrigCount As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetAdRetrigCount(System.Int32 retrigCount)
```

### Parameters

retrigCount

Specifies the number of samples to acquire for each trigger event when RetrigMode is set.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## BoardConfig.GetBaseAdr() method

Gets the base address used by the Universal Library to communicate with a board. This is recommended for use only with ISA bus boards.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetBaseAdr(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetBaseAdr(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the base address to return (PCI boards may have several address ranges).

*configVal*

The board's base address.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetBoardType() method

Gets the unique number (device ID) assigned to the board (between 0 and 8000h) indicating the type of board installed.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetBoardType(ByRef configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetBoardType(out int configVal)
```

### Parameters

*configVal*

Returns a number indicating the board type.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetCiNumDevs() method

Gets the number of counter devices on the board.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetCiNumDevs(ByRef configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetCiNumDevs(out int configVal)
```

### Parameters

*configVal*

Returns the number of counter devices.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetClock() method

Gets the counter's clock frequency in MHz (40, 10, 8, 6, 5, 4, 3, 2, 1), or 0 for not supported.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetClock(ByRef configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetClock(out int configVal)
```

### Parameters

*configVal*

Clock frequency in MHz.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetDACRetrigCount() method

Gets the number of samples to generate during each trigger event when ScanOptions.RetrigMode is enabled.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDACRetrigCount(ByRef retrigCount As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDACRetrigCount(System.Int32 retrigCount)
```

### Parameters

retrigCount

Specifies the number of samples to generate for each trigger event when RetrigMode is set.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetDACStartup() method

Returns the board's configuration register STARTUP bit setting. Refer to SetDACStartup() [Notes](#) section for more information.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDACStartup(ByVal devNum As Integer, ByRef configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetDACStartup(int devNum, out int configVal)
```

### Parameters

*devNum*

The number of the DAC channel whose startup bit setting you want to get.

*configVal*

Returns the setting of the startup bit (0 or 1).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- configVal - Returns 0 if the startup bit is disabled, or 1 if the startup bit is enabled.

## BoardConfig.GetDACUpdateMode() method

Returns the update mode for a digital-to-analog converter (DAC).

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDACUpdateMode(ByVal configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetDACUpdateMode(out int configVal)
```

### Parameters

*configVal*

Returns a number indicating the DAC update mode (0 = immediate, 1 = on command).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- configVal - If configVal returns 0, the DAC update mode is immediate. Values written with [AOut\(\)](#) or [AOutScan\(\)](#) are automatically output by the DAC channels.

If configVal returns 1, the DAC update mode is set to on command. Values written with AOut() or AOutScan() are not output by the DAC channels until a [DACUpdate\(\)](#) method call is made.

## BoardConfig.GetDeviceId() method

Returns the name that identifies the instance of a device.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDeviceId(ByRef configVal As String, ByRef maxLen As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDeviceId(System.String configVal, System.Int32 maxLen)
```

### Parameters

configVal

Returns a string containing the name that identifies the device.

maxLen

Specifies the maximum number of bytes to read, and returns the number of bytes that were actually read.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## BoardConfig.GetDeviceNotes() method

Returns the device notes that are stored in the device's memory.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function prototype

VB .NET

```
Public Function GetDeviceNotes(ByVal start As Integer, ByRef configVal As String, ByRef maxLen As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDeviceNotes(System.Int32 start , System.String configVal, System.Int32 maxLen)
```

### Parameters

*start*

The start address of the device's memory to begin reading.

*maxLen*

Specifies the maximum number of bytes to read. Returns the number of bytes that were actually read.

*configVal*

Returns a string containing the name that identifies the device.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetDiNumDevs() method

Gets the number of digital devices on the board.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDiNumDevs(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDiNumDevs(out int configVal)
```

### Parameters

configVal

Returns the number of digital devices.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetDmaChan() method

Gets the DMA channel (0, 1, or 3) set for the board.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDmaChan(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDmaChan(out int configVal)
```

### Parameters

configVal

Returns DMA channel. 0, 1, or 3.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetDtBoard() method

Gets the number of the board with the DT-Connect interface used to connect to external memory boards.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDtBoard(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDtBoard(out int configVal)
```

### Parameters

*configVal*

Returns the board number of the board that the external memory board is connected to.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetIntLevel() method

Gets the interrupt level set for the board (0 for none, or 1 to 15).

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

## Function Prototype

VB .NET

```
Public Function GetIntLevel(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetIntLevel(out int configVal)
```

## Parameters

*configVal*

Returns the interrupt level (0 for none, or 1 – 15).

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetNumAdChans() method

Gets the number of A/D channels.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetNumAdChans(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetNumAdChans(out int configVal)
```

### Parameters

*configVal*

Returns the number of A/D channels.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetNumDaChans() method

Gets the number of D/A channels.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetNumDaChans(ByRef configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetNumDaChans(out int configVal)
```

### Parameters

*configVal*

Returns the number of D/A channels.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetNumExps() method

Gets the number of expansion boards.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetNumExps(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetNumExps(out int configVal)
```

### Parameters

*configVal*

Returns the number of expansion boards attached to the board.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## BoardConfig.GetNumIoPorts() method

Gets the number of I/O ports used by the board.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetNumIoPorts(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetNumIoPorts(out int configVal)
```

### Parameters

*configVal*

Returns the number of I/O ports used by the board.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetPanID() method

Returns the Personal Area Network (PAN) identifier for wireless communication.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetPANID(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetPANID(System.Int32 configVal)
```

### Parameters

*configVal*

Returns a number from 0 to 65,534 that identifies the Personal Area Network used for wireless communication.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetRange() method

Gets the selected voltage range. For switch-selectable gains only.

If the selected A/D board does not have a programmable gain feature, this method returns the range as defined by the installed InstaCal settings. If InstaCal and the board are installed correctly, the range returned corresponds to the input range set by switches on the board. Refer to board-specific information for a list of the A/D ranges supported by each board.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetRange(ByRef configVal As MccDag.Range) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetRange(out MccDag.Range configVal)
```

### Parameters

*configVal*

Returns the selected voltage range. Refer to board-specific information for a list of [valid settings](#).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetRFChannel() method

Returns the RF channel number that a wireless device uses to communicate.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetRFChannel(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetRFChannel(System.Int32 configVal)
```

### Parameters

*configVal*

Returns the number (from 12 to 23) of the RF channel selected for wireless communication.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetRSS() method

Returns the signal strength in dBm of a signal received by a remote device.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetRSS(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetRSS(System.Int32 configVal)
```

### Parameters

*configVal*

Returns the received signal strength in dBm of the remote device. In general, values are negative.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetUsesExps() method

Gets the True/False value indicating support of expansion boards.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetUsesExps(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetUsesExps(out int configVal)
```

### Parameters

*configVal*

Returns *True* if the board supports expansion boards, or *False* if the board does not support expansion boards.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.GetWaitState() method

Gets the value of the wait state jumper (1-enabled, 0-disabled).

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetWaitState(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetWaitState(out int configVal)
```

### Parameters

*configVal*

Returns the wait state of the board.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetAdRetrigCount() method

Sets the number of samples to acquire during each trigger event when ScanOptions.RetrigMode is enabled.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetAdRetrigCount(ByRef retrigCount As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetAdRetrigCount(System.Int32 retrigCount)
```

### Parameters

*retrigCount*

Specifies the number of samples to acquire per trigger event when RetrigMode is set. Set to zero to use the value of the numPoints argument of the scan function.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## BoardConfig.SetBaseAdr() method

Sets the base address used by the Universal Library to communicate with a board. This is recommended for use only with ISA bus boards.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetBaseAdr(ByVal devNum As Integer, ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetBaseAdr(int deveNum, int configVal)
```

### Parameters

*devNum*

Number of the base address to configure (should always be 0 – can't configure PCI base addresses).

*configVal*

Sets the base address of the board.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetClock() method

Sets the counter's clock source by the frequency (40, 10, 8, 6, 5, 4, 3, 2, 1), or 0 for not supported.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetClock(ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetClock(int configVal)
```

### Parameters

*configVal*

Sets the clock frequency in MHz.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetDACStartup() method

Sets the board's configuration register STARTUP bit to 0 or 1 to enable/disable the storing of digital-to-analog converter (DAC) startup values. Each time the DAC board is powered up, the stored values are written to the DACs. New DAC start-up values are stored in memory by calling [AOut\(\)](#) or [AOutScan\(\)](#) after calling SetDacStartup() with the argument set to 1. Refer to the "[Notes](#)" section below for more information.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetDACStartup(ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetDACStartup(int configVal)
```

### Parameters

*configVal*

Set to 0 to disable, or 1 to enable the storing of startup values for the channel.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

- Use the SetDACStartup() method to store the DAC values you would like each DAC channel to be set to each time the board is powered up.
- To store the current DAC values as start-up values, call SetDACStartup() with a configVal value of 1. Then, each time you call [AOut\(\)](#) or [AOutScan\(\)](#), the value written for each channel is stored in NV RAM. The last value written to a particular channel while SetDACStartup() is set to 1 is the value that channel will be set to at power up. Call SetDACStartup() again with a configVal value of 0 to stop storing values in NV RAM.

### Example

```
DacBoard.BoardConfig.SetDACStartup(1);  
    for (int i =1; i <8; i++)  
    {  
        DacBoard.AOut(i, BIP5VOLTS, DACValue[i]);  
    }  
DacBoard.BoardConfig.SetDACStartup(chanNum, 0);
```

## BoardConfig.SetDACRetrigCount() method

Sets the number of samples to generate during each trigger event when ScanOptions.RetrigMode is enabled.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetDACRetrigCount(ByRef retrigCount As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetDACRetrigCount(System.Int32 retrigCount)
```

### Parameters

*retrigCount*

Specifies the number of samples to generate per trigger event when RetrigMode is set. Set to zero to use the value of the numPoints argument of the scan function.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetDACUpdateMode() method

Sets the update mode for a digital-to-analog converter (DAC).

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetDACUpdateMode(ByVal devNum as Integer, ByVal configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo SetDACUpdateMode(int devNum, int configVal)
```

### Parameters

*devNum*

Number of the channel whose update mode you want set.

*configVal*

When set to 0, the DAC update mode is *immediate*. Values written with [AOut\(\)](#) or [AOutScan\(\)](#) are automatically output by the DAC channels.

When set to 1, the DAC update mode is *on command*. Values written with [AOut\(\)](#) or [AOutScan\(\)](#) are not output by the DAC channel(s) until a [DACUpdate\(\)](#) method call is made.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetDeviceId() method

Sets the name that identifies the instance of a device.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetDeviceId(ByVal configVal As String, ByRef maxLen As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetDeviceId(System.String configVal, System.Int32 maxLen)
```

### Parameters

*configVal*

Sets the string that contains the name identifying a device.

*maxLen*

Specifies the maximum number of bytes to write, and returns the number of bytes that were actually written. For WLS Series devices, the string can contain up to 20 characters.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetDeviceNotes() method

Sets the device notes to store in the device's memory.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetDeviceNotes(ByVal start As Integer, ByVal configVal As String, ByRef maxLen As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetDeviceNotes(System.Int32 start, System.String configVal, System.Int32 maxLen)
```

### Parameters

*start*

The start address of the device's memory to begin writing.

*maxLen*

Specifies the maximum number of bytes to write, and returns the number of bytes that were actually written. For WLS Series devices, the string can contain up to 20 characters.

*configVal*

The text to store in the device's memory.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetDmaChan() method

Sets the DMA channel (0, 1 or 3).

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetDmaChan(ByVal configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo SetDmaChan(int configVal)
```

### Parameters

*configVal*

Sets the DMA channel to 0, 1 or 3.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## BoardConfig.SetIntLevel() method

Sets the interrupt level: 0 for none, or 1 to 15. Recommended for use only with ISA bus boards.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetIntLevel(ByVal configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo SetIntLevel(int configVal)
```

### Parameters

*configVal*

Sets the interrupt level. Valid settings are 0 for none, or 1 – 15.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetNumAdChans() method

Sets the number of A/D channels available on the board.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetNumAdChans(ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetNumAdChans(int configVal)
```

### Parameters

*configVal*

Sets the number of A/D channels on the board. Check board-specific info for valid numbers. Note that this setting affects the single-ended/differential input mode of boards for which this setting is programmable.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetPanID() method

Sets the Personal Area Network (PAN) identifier used for wireless communication.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetPANID(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetPANID(System.Int32 configVal)
```

### Parameters

*configVal*

Sets the number (from 0 to 65,534) that identifies the Personal Area Network used for wireless communication.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetRange() method

Sets the selected voltage range. For use with boards for which the range is manually selected.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetRange(ByVal configVal As MccDag.Range) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetRange(MccDag.Range configVal)
```

### Parameters

*configVal*

Range code.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetRFChannel() method

Sets the RF channel number used for wireless communications.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetRFChannel(ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetRFChannel(System.Int32 configVal)
```

### Parameters

*configVal*

Sets the number of the RF channel to use for wireless communications. Valid channel numbers are 12 to 23.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardConfig.SetWaitState() method

Sets the value of the Wait State jumper.

Member of the [cBoardConfig](#) class. Accessible from the [MccBoard.BoardConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetWaitState(ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetWaitState(int configVal)
```

### Parameters

*configVal*

Sets the wait state on the board (1 = enabled, 0 = disabled).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## BoardNum property

Number of the board associated with an instance of the MccBoard class.

Member of the [MccBoard class](#).

## Property prototype

VB .NET

```
Public ReadOnly Property BoardNum As Integer
```

C# .NET

```
public int BoardNum {get}
```

## CtrConfig property

Represents an instance of the [cCtrConfig](#) class. Use this property to call counter chip configuration methods.

Member of the [MccBoard](#) class.

## Property prototype

VB .NET

```
Public ReadOnly Property CtrConfig As MccDaq.cCtrConfig
```

C# .NET

```
public MccDaq.cCtrConfig CtrConfig {get}
```

## Methods

The GetCtrType() configuration method is accessible only from the CtrConfig property. Before you call this method, you need to create an instance of an MccBoard object.

```
Dim MyBoard As MccDaq.MccBoard  
MyBoard = New MccDaq.MccBoard(MyBoardNum)
```

To call this method from the CtrConfig property, use the notation shown in the example below.

```
MyErrorInfo = MyBoard.CtrConfig.GetCtrType(MyCtrNum, MyCtrType)
```



## CtrConfig.GetCtrType() method

Gets the value that indicates the counter type.

Member of the [cCtrConfig](#) class. Accessible from the [MccBoard.CtrConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetCtrType(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetCtrType(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the counter device.

*configVal*

Returns the counter type, where: 1 = 8254, 2 = 9513, 3 = 8536, 4 = 7266, 5 = event counter, 6 = scan counter, 7 = timer counter, 8 = quadrature counter, and 9 = pulse counter.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## DioConfig property

Represents an instance of the cDioConfig class. Use this property to call various digital I/O configuration methods.

Member of the [MccBoard](#) class.

## Property prototype

VB .NET

```
Public ReadOnly Property DioConfig As MccDaq.cDioConfig
```

C# .NET

```
public MccDaq.cDioConfig DioConfig {get}
```

## Methods

Six configuration methods are accessible only from the DioConfig property. Before you call any of these methods, you need to create an instance of an MccBoard object.

```
Dim MyBoard As MccDaq.MccBoard  
MyBoard = New MccDaq.MccBoard(MyBoardNum)
```

To call these methods from the DioConfig property, use the notation shown in the example below.

```
MyErrorInfo = MyBoard.DioConfig.GetNumBits(MyDevNum, MyNumBits)
```

## DioConfig.GetConfig() method

Gets the configuration of a digital device (digital input or digital output).

Member of the [cDioConfig](#) class. Accessible from the [MccBoard.DioConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetConfig(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetConfig(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the digital device.

*configVal*

Current configuration (1 = DigitalOut, 2 = DigitalIn).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## DioConfig.GetCurVal method

Gets the current value of digital outputs.

Member of the [cDioConfig](#) class. Accessible from the [MccBoard.DioConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetCurVal(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetCurVal(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the digital device.

*configVal*

Current value of the digital output.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## DioConfig.GetDevType() method

Gets the device type of the digital port (AUXPORT, FIRSTPORTA, etc.).

Member of the [cDioConfig](#) class. Accessible from the [MccBoard.DioConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDevType(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDevType(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the digital device.

*configVal*

Constant that indicates the type of device (AUXPORT, FIRSTPORTA, etc.).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## DioConfig.GetDInMask() method

Determines the bits on a specified port that are configured for input.

Member of the [cDioConfig](#) class. Accessible from the [MccBoard.DioConfig](#) property.

### Function prototype

VB .NET:

```
Public Function GetDInMask(ByVal devNum As Integer, ByRef configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET:

```
public MccDaq.ErrorInfo GetDInMask(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the port whose input bit configuration you want to determine.

*configVal*

Returns the bit configuration of the specified port. Any of the lower eight bits that return a value of 1 are configured for input.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

Use GetDInMask() with the [GetDOutMask\(\)](#) method to determine if an AuxPort is configurable. If you apply both methods to the same port, and both configVal parameters returned have input and output bits that overlap, the port is not configurable. You can determine overlapping bits by ANDing both parameters.

For example, the PCI-DAS08 has seven bits of digital I/O (four outputs and three inputs). For this board, the configVal parameter returned by GetDInMask() is always 7 (0000 0111), while the configVal parameter returned by GetDOutMask() is always 15 (0000 1111). When you *And* both configVal parameters together, you get a non-zero number (7). Any non-zero number indicates that input and output bits overlap for the specified port, and that port is a non-configurable AuxPort.

## DioConfig.GetDOutMask() method

Determines the bits on a specified port that are configured for output.

Member of the [cDioConfig](#) class. Accessible from the [MccBoard.DioConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetDOutMask(ByVal devNum As Integer, ByRef configVal As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetDOutMask(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the port whose output bit configuration you want to determine.

*configVal*

Returns the bit configuration of the specified port. Any of the lower eight bits that return a value of 1 are configured for output.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

Use GetDOutMask() with the [GetDInMask\(\)](#) method to determine if an AuxPort is configurable. If you apply both methods to the same port, and both configVal parameters returned have input and output bits that overlap, the port is not configurable. You can determine overlapping bits by *ANDing* both parameters.

For example, the PCI-DAS08 has seven bits of digital I/O (four outputs and three inputs). For this board, the configVal parameter returned by GetDInMask() is always 7 (0000 0111), while the configVal parameter returned by GetDOutMask() is always 15 (0000 1111). When you *And* both configVal parameters together, you get a non-zero number (7). Any non-zero number indicates that input and output bits overlap for the specified port, and that port is a non-configurable AuxPort.

## DioConfig.GetNumBits() method

Gets the number of bits in the digital port.

Member of the [cDioConfig](#) class. Accessible from the [MccBoard.DioConfig](#) property.

### Function Prototype

VB NET

```
Public Function GetNumBits(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# ..NET

```
public MccDag.ErrorInfo GetNumBits(int devNum, out int configVal)
```

### Parameters

devNum

Number of the digital device.

configVal

Number of bits in the digital port.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## ExpansionConfig property

Represents an instance of the cExpansionConfig class. Use this property to call various expansion board configuration methods.

Member of the [MccBoard](#) class.

## Property prototype

VB .NET

```
Public ReadOnly Property ExpansionConfig As MccDaq.cExpansionConfig
```

C# .NET

```
public MccDaq.cExpansionConfig ExpansionConfig {get}
```

## Methods

Over a dozen configuration methods are accessible only from the ExpansionConfig property. Before you call any of these methods, you need to create an instance of an MccBoard object.

```
Dim MyBoard As MccDaq.MccBoard  
MyBoard = New MccDaq.MccBoard(MyBoardNum)
```

To call these methods from the ExpansionConfig property, use the notation shown in the example below.

```
MyErrorInfo = MyBoard.ExpansionConfig.GetBoardType(MyExpNum, MyExpType)
```

## ExpansionConfig.GetBoardType()

Gets the expansion board type.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetBoardType(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetBoardType(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Returns a number indicating the expansion board type. Refer to "[Measurement Computing Device IDs](#)" for more information.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.GetCjcChan() method

Gets the channel that the CJC is connected to.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetCjcChan(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetCjcChan(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number indicating the channel on the A/D board that the CJC is connected to.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.GetMuxAdChan1() method

Gets the first A/D channel that the EXP board is connected to.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetMuxAdChan1(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetMuxAdChan1(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number indicating the first A/D channel that the EXP board is connected to.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.GetMuxAdChan2() method

Gets the second A/D channel that the EXP board is connected to.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetMuxAdChan2(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetMuxAdChan2(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number indicating the second A/D channel that the EXP board is connected to.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.GetNumExpChans() method

Gets the number of expansion board channels.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetNumExpChans(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetNumExpChans(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number of channels on the expansion board.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.GetRange1() method

Gets the range/gain of the low 16 channels.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetRange1(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetRange1(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Returns the range (gain) of the low 16 channels.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.GetRange2() method

Gets the range/gain of the high 16 channels.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetRange2(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetRange2(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Returns the range (gain) of the high 16 channels.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## ExpansionConfig.GetThermType() method

Gets the type of thermocouple configuration for the board (J, K, E, T, R, S, and B types).

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function GetThermType(ByVal devNum As Integer, ByRef configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetThermType(int devNum, out int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number indicating the type of thermocouple configured for the board. (J = 1, K = 2, T = 3, E = 4, R = 5, S = 6, B = 7, Platinum .00392 = 257, Platinum .00391 = 258, Platinum .00385 = 259, Copper .00427 = 260, Nickel/Iron .00581 = 261, Nickel/Iron .00527 = 262).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.SetCjcChan() method

Sets the channel that the CJC is connected to.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetCjcChan(ByVal devNum As Integer, ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetCjcChan(int devNum, int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number that sets the A/D channel to connect to the CJC.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.SetMuxAdChan1() method

Sets the first A/D channel that the EXP board is connected to.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetMuxAdChan1(ByVal devNum As Integer, ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetMuxAdChan1(int devNum, int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number that sets the first A/D channel that the EXP board is connected to.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.SetMuxAdChan2() method

Sets the second A/D channel that the EXP board is connected to.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetMuxAdChan2(ByVal devNum As Integer, ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetMuxAdChan2(int devNum, int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number that sets the second A/D channel that the EXP board is connected to.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.SetRange1() method

Sets the range/gain of the low 16 channels.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetRange1(ByVal devNum As Integer, ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetRange1(int devNum, int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number that sets the range (gain) of the low 16 channels.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.SetRange2() method

Sets the range/gain of the high 16 channels.

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetRange2(ByVal devNum As Integer, ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetRange2(int devNum, int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number that sets the range (gain) of the high 16 channels.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ExpansionConfig.SetThermType() method

Sets the type of thermocouple configuration for the board (J, K, E, T, R, S, and B types).

Member of the [cExpansionConfig](#) class. Accessible from the [MccBoard.ExpansionConfig](#) property.

### Function Prototype

VB .NET

```
Public Function SetThermType(ByVal devNum As Integer, ByVal configVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo SetThermType(int devNum, int configVal)
```

### Parameters

*devNum*

Number of the expansion board.

*configVal*

Number that sets the type of thermocouple configured for the board. (J = 1, K = 2, T = 3, E = 4, R = 5, S = 6, B = 7, Platinum .00392 = 257, Platinum .00391 = 258, Platinum .00385 = 259, Copper .00427 = 260, Nickel/Iron .00581 = 261, Nickel/Iron .00527 = 262).

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## GetSignal() method

Retrieves the configured Auxiliary or DAQ Sync connection and polarity for the specified timing and control signal.

This method is intended for advanced users. Except for the SYNC\_CLK input, you can easily view the settings for the timing and control signals using InstaCal.

Member of the [MccBoard class](#).

Note: This method is not supported by all board types. Refer to the board-specific information contained in the *Universal Library User's Guide*.

## Function Prototype

VB .NET

```
Public Function GetSignal(ByVal direction As MccDaq.SignalDirection, ByVal signalType As MccDaq.SignalType,  
ByVal index As Integer, ByRef connectionPin As MccDaq.ConnectionPin, ByRef  
signalPolarity As MccDaq.SignalPolarity) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetSignal(MccDaq.SignalDirection direction, MccDaq.SignalType signalType, int  
index, out MccDaq.ConnectionPin connectionPin, out MccDaq.SignalPolarity signalPolarity)
```

## Parameters

*direction*

Specifies whether to retrieve the source ([MccDaq.SignalDirection.SignalIn](#)) or destination ([MccDaq.SignalDirection.SignalOut](#)) of the specified signal.

*signalType*

Signal type whose connection is to be retrieved. See [SelectSignal](#) for valid signal types.

*index*

Indicates which connection to reference when there is more than one connection associated with the output signal type. When querying output signals, increment this value until [BadIndex](#) is returned or 0 is returned via the connection parameter to determine all the output connectionPins for the specified output Signal. The first connectionPin is indexed by 0.

For input signals (direction= [MccDaq.SignalDirection.SignalIn](#)), always set index to 0.

*connectionPin*

The specified connection is returned through this variable. Note that this is set to 0 if no connection is associated with the signalType, or if the index is set to an invalid value. Refer to the [SelectSignal\(\)](#) method's "direction, connectionPin, and polarity parameter values" section for expected return values.

*signalPolarity*

Holds the polarity for the associated signalType and connectionPin.

For output signals assigned an AuxOut connectionPin, the return value is either [MccDaq.SignalPolarity.Inverted](#) or [MccDaq.SignalPolarity.NonInverted](#).

For [AdcConvert](#), [DacUpdate](#), [AdcTbSrc](#) and [DacTbSrc](#) input signals, the return value is either [MccDaq.SignalPolarity.PositiveEdge](#) or [MccDaq.SignalPolarity.NegativeEdge](#).

All other signals return 0.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- The above timing and control configuration information can also be viewed and edited inside InstaCal: Open InstaCal, click on the board, and press the "Configure..." button or menu item. If the board supports DAQ Sync and Auxiliary Input/Output signal connections, a button labeled "Advanced Timing & Control Configuration" displays. Press this button to open a display for viewing and modifying the above timing and control signals.



## NumBoards property

Returns the maximum number of boards you can install at one time.

Member of the [GlobalConfig class](#).

## Property prototype

VB .NET

```
Public Shared ReadOnly Property NumBoards As Integer
```

C# .NET

```
public int NumBoards {get}
```

## NumExpBoards property

Returns the maximum number of expansion boards you can install on a board.

Member of the [GlobalConfig class](#).

## Property prototype

VB .NET

```
Public Shared ReadOnly Property NumExpBoards As Integer
```

C# .NET

```
public static int NumExpBoards [get]
```

## SelectSignal() method

Configures timing and control signals to use specific Auxiliary or DAQ Sync connections as a source or destination.

This method is intended for advanced users. Except for the SyncClk input, you can easily configure all the timing and control signals using InstaCal.

Member of the [MccBoard class](#).

**Note:** SelectSignal() is not supported by all board types. Refer to the board-specific information contained in the *Universal Library User's Guide* for details.

## Function Prototype

VB .NET

```
Public Function SelectSignal(ByVal direction As MccDaq.SignalDirection, ByVal signalType As MccDaq.SignalType,  
ByVal connectionPin As MccDaq.ConnectionPin, ByVal polarity As MccDaq.SignalPolarity) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo SelectSignal(MccDaq.SignalDirection direction, MccDaq.SignalType signal,  
MccDaq.ConnectionPin connectionPin, MccDaq.SignalPolarity polarity)
```

## Parameters

*direction*

Direction of the specified signal type to be assigned a connector pin. For most signal types, this should be either SignalIn or SignalOut.

For the SyncClk, AdcTbSrc and DacTbSrc signals, the external source can also be disabled by specifying Disabled(=0), such that it is neither input nor output.

Set it in conjunction with the signalType, connectionPin, and polarity parameters. Refer to the "[direction, connectionPin, and polarity parameter values](#)" section below.

*signalType*

Signal type to be associated with a connector pin. Set it to one of the constants in the "[signalType parameter values](#)" section below.

*connectionPin*

Designates the connector pin to associate the signal type and direction. Since individual pin selection is not allowed for the DAQ-Sync connectors, all DAQ-Sync pin connections are referred to as [DsConnector](#). The MccDaq.ConnectionPin.AuxIn and MccDaq.ConnectionPin.AuxOut settings match their corresponding hardware pin names.

*polarity*

AdcTbSrc and DacTbSrc input signals (direction = MccDaq.SignalDirection.SignalIn) can be set for either rising edge (MccDaq.SignalPolarity.PositiveEdge) or falling edge (MccDaq.SignalPolarity.NegativeEdge) signals. The AuxOut connections can be set to MccDaq.SignalPolarity.Inverted or MccDaq.SignalPolarity.NonInverted from their internal polarity.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## signalType parameter values

All of the signalType settings are MccDaq.SignalType enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the SignalType enumeration (variable = MccDaq.SignalType.AdcConvert, variable = MccDaq.SignalType.AdcGate, etc.).

Signal	Connection
AdcConvert	A/D conversion pulse or clock.
AdcGate	External gate for A/D conversions.
AdcScanClk	A/D channel scan signal.
AdcScanStop	A/D scan completion signal.
ADC_SSH	A/D simultaneous sample and hold signal.
AdcStartScan	Start of A/D channel-scan sequence signal.
AdcStartTrig	A/D scan start trigger.
AdcStopTrig	A/D stop- or pre- trigger.
AdcTbSrc	A/D pacer timebase source.
Ctr1Clk	CTR1 clock source.
Ctr2Clk	CTR2 clock source.
DacStartTrig	D/A start trigger.
DacTbSrc	D/A pacer timebase source.
DacUpdate	D/A update signal.
DGnd	Digital ground.
SyncClk	STC timebase signal.

## direction, connectionPin, and polarity parameter values

- All of the direction settings are [MccDaq.SignalDirection](#) enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the SignalDirection enumeration (variable = MccDaq.SignalDirection.SignalIn, variable = MccDaq.SignalDirection.SignalOut, etc.).
- All of the connectionPin settings are [MccDaq.ConnectionPin](#) enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ConnectionPin enumeration (variable = MccDaq.ConnectionPin.AuxIn0, variable = MccDaq.ConnectionPin.DsConnector, etc.).
- All of the polarity settings are [MccDaq.SignalPolarity](#) enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the SignalPolarity enumeration (variable = MccDaq.SignalPolarity.PositiveEdge, variable = MccDaq.ConnectionPin.Negative, etc.).

Valid input settings (direction = MccDaq.SignalDirection.SignalIn)		
signalType	connectionPin	polarity
AdcConvert	AuxIn0..AuxIn5 DsConnector	PositiveEdge or NegativeEdge
AdcGate	AuxIn0..AuxIn5	See <a href="#">SetTrigger()</a> .
AdcStartTrig	AuxIn0..AuxIn5 DsConnector	
AdcStopTrig	AuxIn0..AuxIn5 DsConnector	
AdcTbSrc	AuxIn0..AuxIn5	PositiveEdge or NegativeEdge
DacStartTrig	AuxIn0..AuxIn5 DsConnector	Not assigned here.
DscTbSrc	AuxIn0..AuxIn5	PositiveEdge or NegativeEdge
DacUpdate	AuxIn0..AuxIn5 DsConnector	PositiveEdge or NegativeEdge
SyncClk	DsConnector	Not assigned here.

Valid output settings (direction = MccDaq.SignalDirection.SignalOut)		
signalType	connectionPin	polarity
AdcConvert	AuxOut0..AuxOut2 DsConnector	Inverted* or NonInverted
AdcScanClk	AuxOut0..AuxOut2	
AdcScanStop	AuxOut0..AuxOut2	
AdcSsh	AuxOut0..AuxOut2	
AdcStartScan	AuxOut0..AuxOut2	
AdcStartTrig	AuxOut0..AuxOut2 DsConnector	
AdcStopTrig	AuxOut0..AuxOut2 DsConnector	
Ctr1Clk	AuxOut0..AuxOut2	
Ctr2Clk	AuxOut0..AuxOut2	
DacStartTrig	AuxOut0..AuxOut2 DsConnector	
DacUpdate	AuxOut0..AuxOut2 DsConnector	
DGND	AuxOut0..AuxOut2	Not assigned here.
SyncClk	DsConnector	Not assigned here.

\* Inverted is only valid for Auxiliary Output (AuxOut) connections.

Valid disabled settings (direction = MccDaq.SignalDirection.Disabled)		
signalType	connectionPin	polarity
<a href="#">AdcTbSrc</a>	Not assigned here.	Not assigned here.
<a href="#">DacTbSrc</a>		
SyncClk		

## Notes

- You can view and edit the above timing and control configuration information from InstaCal. Open InstaCal, click on the board, and press the **Configure...** button or menu item. If the board supports DAQ Sync and Auxiliary Input/Output signal connections, a **Advanced Timing & Control Configuration** button appears. Press that button to open a display for viewing and modifying the above timing and control signals.
- Except for the AdcTbSrc, DacTbSrc and SyncClk signals, selecting an input signal connection does not necessarily activate it. Alternately, assigning an output signal to a connection does activate the signal upon performing the respective operation. For instance, when running an ExtClock AInScan(), AdcConvert SignalIn selects the connection to use as an external clock to pace the A/D conversions; if AInScan() is run without setting the **ExtClock** option, however, the selected connection is not activated and the signal at that connection is ignored. In both cases, the AdcConvert signal is output the connection(s) selected for the AdcConvert SignalOut. Since there are no scan options for enabling the Timebase Source and the SyncClk, selecting an input for the A/D or D/A Timebase Source, or SyncClk does activate the input source for the next respective operations.
- Multiple input signals can be mapped to the same AuxIn connection by successive calls to SelectSignal(); however, only one connection can be mapped to each input signal. If another connection had already been assigned to an input signal, the former selection is de-assigned and the new connection is assigned.
- Only one output signal can be mapped to the same AuxOut $n$  connection; however, multiple connections can be mapped to the same output signal by successive calls to SelectSignal(). If an output signal had already been assigned to a connection, then the former output signal is de-assigned and the new output signal is assigned to the connection.
- When selecting **DsConnector** for a signal, only one direction per signal type can be defined at a given time. *Attempting to assign both directions of a signal to the DsConnector results in only the latest selection being applied.*

If the signal type had formerly been assigned an input direction from the DsConnector, assigning the output direction for that signal type results in the input signal being reassigned to its default connection.

Default input signal connections	Input signal	Default connection
	AdcConvert	AuxIn0
	AdcGate	AuxIn5
	AdcStartTrig	AuxIn1
	AdcStopTrig	AuxIn2
	DacUpdate	AuxIn3
	DacStartTrig	AuxIn3

- AdcTbSrc and DacTbSrc are intended to synchronize the timebase of the analog input and output pacers across two or more

boards. Internal calculations of sampling and update rates assume that the external timebase has the same frequency as its internal clock. Adjust sample rates to compensate for differences in clock frequencies.

For instance, if the external timebase has a frequency of 10 MHz on a board that has a internal clock frequency of 40 MHz, the scan function samples or updates at a rate of about 1/4 the rate entered. However, while compensating for differences in external timebase and internal clock frequency, if the rate entered results in an invalid pacer count, the method returns a BADRATE error.

## SetTrigger() method

Selects the trigger source and sets up its parameters. This trigger is used to initiate analog to digital conversions using the following Universal Library for .NET methods:

- [AInScan\(\)](#), if the ExtTrigger option is selected.
- [DInScan\(\)](#), if the ExtTrigger option is selected.
- [CInScan\(\)](#), if the ExtTrigger option is selected.
- [APretrig\(\)](#)
- [FilePretrig\(\)](#)

Member of the [MccBoard class](#).

## Function Prototype

### VB .NET

```
Public Function SetTrigger(ByVal trigType As MccDaq.TriggerType, ByVal lowThreshold As Short, ByVal highThreshold As Short) As MccDaq.ErrorInfo

Public Function SetTrigger(ByVal trigType As MccDaq.TriggerType, ByVal lowThreshold As System.UInt16, ByVal highThreshold As System.UInt16) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo SetTrigger(MccDaq.TriggerType trigType, short lowThreshold, short highThreshold)

public MccDaq.ErrorInfo SetTrigger(MccDaq.TriggerType trigType, ushort lowThreshold, ushort highThreshold)
```

## Parameters

### *trigType*

Specifies the type of triggering based on the external trigger source. Set it to one of the constants specified in the Type column listed in the "[trigType parameter values](#)" section below.

### *LowThreshold*

Selects the low threshold used when the trigger input is analog. The range depends upon the resolution of the trigger circuitry. Must be 0 to 255 for 8-bit trigger circuits, 0 to 4,095 for 12-bit trigger circuits, and 0 to 65,535 for 16-bit trigger circuits. Refer to the "[Notes](#)" section below.

### *HighThreshold*

Selects the high threshold used when the trigger input is analog. The range depends upon the resolution of the trigger circuitry. Must be 0 to 255 for 8-bit trigger circuits, 0 to 4,095 for 12-bit trigger circuits, and 0 to 65,535 for 16-bit trigger circuits. Refer to the "[Notes](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## trigType parameter values

All of the trigType settings are MccDaq.TriggerType enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the TriggerType enumeration (variable = MccDaq.TriggerType.GateNegHys, variable = MccDaq.TriggerType.GatePosHys, etc.).

Trigger Source	Type	Explanation
Analog	GateNegHys	AD conversions are enabled when the external analog trigger input is more positive than highThreshold. AD conversions are disabled when the external analog trigger input is more negative than lowThreshold. Hysteresis is the level between lowThreshold and highThreshold.
	GatePosHys	AD conversions are enabled when the external analog trigger input is more negative than lowThreshold. AD conversions are disabled when the external analog trigger input is more positive than highThreshold. Hysteresis is the level between lowThreshold and highThreshold.
	GateAbove	AD conversions are enabled as long as the external analog trigger input is more positive than highThreshold
	GateBelow	AD conversions are enabled as long as the external analog trigger input is more negative than lowThreshold.
	GateInWindow	AD conversions are enabled as long as the external analog trigger is inside the region defined by lowThreshold and highThreshold.
	GateOutWindow	AD conversions are enabled as long as the external analog trigger is outside the region defined by lowThreshold and HighThreshold.
	TrigAbove	AD conversions are enabled when the external analog trigger input transitions from below highThreshold to above. Once conversions are enabled, the external trigger is ignored.
	TrigBelow	AD conversions are enabled when the external analog trigger input transitions from above lowThreshold to below. Once conversions are enabled, the external trigger is ignored.
Digital	GateHigh	AD conversions are enabled as long as the external digital trigger input is 5V (logic HIGH or '1').
	GateLow	AD conversions are enabled as long as the external digital trigger input is 0V (logic LOW or '0').
	TrigHigh	AD conversions are enabled when the external digital trigger is 5V (logic HIGH or '1'). Once conversions are enabled, the external trigger is ignored.
	TrigLow	AD conversions are enabled when the external digital trigger is 0V (logic LOW or '0'). Once conversions are enabled, the external trigger is ignored.
	TrigPosEdge	AD conversions are enabled when the external digital trigger makes a transition from 0V to 5V (logic LOW to HIGH). Once conversions are enabled, the external trigger is ignored.
	TrigNegEdge	AD conversions are enabled when the external digital trigger makes a transition from 5V to 0V (logic HIGH to LOW). Once conversions are enabled, the external trigger is ignored.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- The value of the threshold must be within the range of the analog trigger circuit associated with the board. Refer to the board-specific information in the *Universal Library User's Guide*. For example, on the PCI-DAS1602/16 the analog trigger circuit handles  $\pm 10$  V. A value of 0 corresponds to  $-10$  V, whereas a value of 65,535 corresponds to  $+10$  V.

If you are using signed integer types, the thresholds range from  $-32,768$  to  $32,767$  for 16-bit boards, instead of from 0 to 65,535. In this case, the unsigned value of 65,535 corresponds to a value of  $-1$ , 65,534 corresponds to  $-2$ , ...,  $32,768$  corresponds to  $-32,768$ .

- For most boards that support analog triggering, you can pass the required trigger voltage level and the appropriate range to [FromEngUnits\(\)](#) to calculate the highThreshold and lowThreshold values.
- For some boards, you must **manually calculate the threshold** by first calculating the least significant bit (LSB) for a particular range for the trigger resolution of your hardware. You then use the LSB to find the threshold in counts based on an analog voltage trigger threshold. Refer to the following procedure for details. For board-specific information, refer to your hardware in the "Analog Input Boards" section of the *Universal Library User's Guide*.



## Manually calculating the threshold

To calculate the threshold, do the following:

1. Calculate the least significant bit (LSB) by dividing the full scale range (FSR) by  $2^{\text{resolution}}$ .  
FSR is the entire span from  $-FS$  to  $+FS$  of your hardware for a particular range. For example, the full scale range of  $\pm 0$  V is 20 V.
2. Calculate how many times you need to add the LSB calculate in step 1 to the negative full scale ( $-FS$ ) to reach the trigger threshold value.

The maximum threshold value is  $2^{\text{resolution}} - 1$ . The formula is shown here:

$$\text{Abs}(-FS - \text{threshold in volts}) \div (\text{LSB}) = \text{threshold in counts}$$

Here are two examples that use this formula — one for 8-bit trigger resolution, and one for 12-bit trigger resolution.

- 8-bit example using the  $\pm 10$  volt range with a  $-5$  volt threshold:

Calculate the LSB:  $\text{LSB} = 20 \div 2^8 = 20 \div 256 = 0.078125$

Calculate the threshold:  $\text{Abs}(-10 - (-5)) \div 0.078125 = 5 \div 0.078125 = 64$  (round this result if it is not an integer). A count of 64 translates to a voltage threshold of  $-5.0$  volts.

- 12-bit example using the  $\pm 10$  volt range with a  $+1$  volt threshold:

Calculate the LSB:  $\text{LSB} = 20 \div 2^{12} = 20 \div 4096 = 0.00488$

Calculate the threshold:  $\text{Abs}(-10 - 1) \div 0.00488 = 11 \div 0.00488 = 2254$  (rounded from 2254.1). A count of 2254 translates to a voltage threshold of  $0.99952$  volts.

## C7266Config() method

Configures a 7266 counter for desired operation. This method can only be used with boards that contain a 7266 counter chip (Quadrature Encoder boards). For more information, refer to the LS7266R1 data sheet ([ls7266r1.pdf](#)) located in the "Documents" subdirectory of the installation.



Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function C7266Config(ByVal counterNum As Integer, ByVal quadrature As MccDag.Quadrature, ByVal countingMode As MccDag.CountingMode, ByVal dataEncoding As MccDag.DataEncoding, ByVal indexMode As MccDag.IndexMode, ByVal invertIndex As MccDag.OptionState, ByVal flagPins As MccDag.FlagPins, ByVal gateState As MccDag.OptionState) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo C7266Config(int counterNum, MccDag.Quadrature quadrature, MccDag.CountingMode countingMode, MccDag.DataEncoding dataEncoding, MccDag.IndexMode indexMode, MccDag.OptionState invertIndex, MccDag.FlagPins flagPins, MccDag.OptionState gateState)
```

### Parameters

*counterNum*

Number (1 to *n*) of the counter to configure, where *n* is the number of counters on the board.

*quadrature*

Selects the resolution multiplier for quadrature input (X1Quad, X2Quad, or X4Quad), or disables quadrature input (NoQuad) so that the counters can be used as standard TTL counters.

*countingMode*

Selects the operating mode for the counter. Refer to [CountingModes parameter values](#) below.

*dataEncoding*

Selects the format of the data that is returned by the counter - either Binary or BCD format. Options are BinaryCount or BCDCount.

*indexMode*

Selects which action is taken when the Index signal is received. The IndexMode must be set to *IndexDisabled* whenever a Quadrature is set to *NOQuad*, or when GateState is set to *Enabled*. Refer to [IndexModes parameter values](#) below.

*invertIndex*

Selects the polarity of the Index signal. Options are Enabled or Disabled. If set to *Enabled*, the Index signal is assumed to be negative polarity. If set to *Disabled*, the Index signal is assumed to be positive polarity.

*flagPins*

Selects which signals are routed to the FLG1 and FLG2 pins. Refer to the [FlagPins parameter values](#) below.

*gateState*

When gateState is set to *ENABLED*, the channel INDEX input is routed to the RCNTR pin of the LS7266 chip, and is used as a gating signal for the counter. When set to *ENABLED* indexMode must be set to *IndexDisabled*.

### Returns

- [Error code](#) or 0 if no errors

## CountingModes parameter values

NormalMode	Each counter operates as a 24 bit counter that rolls over to 0 when the maximum count is reached.
RangeLimit	In range limit count mode, an upper and lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by loading the PRESET register with the <a href="#">CLoad</a> method after the counter has been configured. The lower limit is always 0. When counting up, the counter freezes whenever the count reaches the value that was loaded into the PRESET register. When counting down, the counter freezes at 0. In either case the counting is resumed only when the count direction is reversed.
NoRecycle	In non-recycle mode the counter is disabled whenever a count overflow or underflow takes place. The counter is re-enabled when a reset or load operation is performed on the counter.
ModuloN	In ModuloN mode, an upper limit is set by loading the PRESET register with a maximum count. Whenever counting up, when the maximum count is reached, the counter will roll-over to 0 and continue counting up. Likewise when counting down, whenever the count reaches 0, it will roll over to the maximum count (in the PRESET register) and continue counting down.

## IndexModes parameter values

IndexDisabled	The Index signal is ignored.
LoadCtr	The channel INDEX input is routed to the LCNTR pin of the LS7266 counter chip. The counter is loaded whenever the signal occurs.
LoadOutLatch	The channel INDEX input is routed to the LCNTR pin of the LS7266 counter chip. The current count is latched whenever the signal occurs. When this mode is selected, the <a href="#">CIn()</a> method will return the same count value each time it is called until the Index signal occurs.
ResetCtr	The channel INDEX input is routed to the RCNTR pin of the LS7266 counter chip. The counter is reset whenever the signal occurs.

## FlagPins parameter values

CarryBorrow	FLG1 pin is Carry output, FLG2 is Borrow output.
CompareBorrow	FLG1 pin is Compare output, FLG2 is Borrow output.
CarryBorrowUpDown	FLG1 pin is Carry/Borrow output, FLG2 is Up/Down signal.
IndexError	FLG1 pin is Index output, FLG2 is Error output.

## C8254Config() method

Configures 8254 counter for desired operation. This method can only be used with 8254 counters. For more information, see the 82C54 data sheet in accompanying [82C54.pdf](#) file located in the *Documents* subdirectory where the UL is installed (C:/Program files/Measurement Computing/DAQ by default).



Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function C8254Config(ByVal counterNum As Integer, ByVal config As MccDaq.C8254Mode) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo C8254Config(int counterNum, MccDaq.C8254Mode config)
```

### Parameters

*counterNum*

Selects one of the counter channels. An 8254 has three counters. The value may be 1 – n, where n is the number of 8254 counters on the board (refer to the board-specific information in the *UL Users Guide*).

*config*

Refer to the 8254 data sheet for a detailed description of each of the configurations. Set it to one of the constants in the [config parameter values](#) section below.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### config parameter values

All of the config settings are MccDaq.C8254Mode enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the C8254Mode enumeration (for example, variable = MccDaq.C8254Mode.HighOnLastCount, variable = MccDaq.C8254Mode.LastShot, etc.).

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 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory of the installation.
HighOnLastCount	Output of counter (OUT N) transitions from low to high on terminal count and remains high until reset. See Mode 0 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory where the UL is installed (C:/Program files/Measurement Computing/DAQ by default).
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 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory of the installation.
RateGenerator	Output of counter (OUT N) pulses low for one clock cycle on terminal count, reloads counter and recycles. See Mode 2 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory of the installation.
SoftwareStrobe	Output of counter (OUT N) pulses low for one clock cycle on terminal count. Count starts after counter is loaded. See Mode 4 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory of the installation.
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 in the 8254 data sheet in the accompanying <a href="#">82C54.pdf</a> file located in the <i>Documents</i> subdirectory of the installation.

## C8536Config() method

Configures an 8536 counter for desired operation. This method can only be used with 8536 counters.

For more information, refer to the *Zilog 8536 product specification*. The document is available on our web site at <http://www.mccdaq.com/PDFmanuals/Z8536.pdf>.



Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

Configure for software triggering:

```
Public Shared Function C8536Config(ByVal counterNum As Integer, ByVal outputControl As MccDag.C8536OutputControl, ByVal recycleMode As MccDag.RecycleMode, ByVal trigType As MccDag.C8536TriggerType) As MccDag.ErrorInfo
```

Configure for hardware triggering; use when existing code includes MccDag.OptionState:

```
Public Function C8536Config(ByVal counterNum As Integer, ByVal outputControl As MccDag.C8536OutputControl, ByVal recycleMode As MccDag.RecycleMode, ByVal retrigger As MccDag.OptionState) As MccDag.ErrorInfo
```

C# .NET

Configure for software triggering:

```
public MccDag.ErrorInfo C8536Config(int counterNum, MccDag.C8536OutputControl outputControl, MccDag.RecycleMode recycleMode, MccDag.C8536TriggerType trigType)
```

Configure for hardware triggering; use when existing code includes MccDag.OptionState:

```
public MccDag.ErrorInfo C8536Config(int counterNum, MccDag.C8536OutputControl outputControl, MccDag.RecycleMode recycleMode, MccDag.OptionState retrigger)
```

### Parameters

*counterNum*

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

INT32 Series boards have two chips installed, so the counterNum value may be 1 to 6.

*outputControl*

Specifies the action of the output signal. Set it to one of the constants in the "[outputControl parameter values](#)" section below.

*retrigger*

If set to Recycle (as opposed to OneTime), the counter automatically reloads to the starting count every time it reaches 0, and then counting continues

*recycleMode*

If set to *Enabled*, every trigger on the counter's trigger input initiates loading of the initial count. Counting proceeds from the initial count.

*trigType*

Specifies the trigger type. Set it to one of the constants in the "[trigType parameter values](#)" section below.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### outputControl parameter values

All of the outputControl settings are MccDag.C8536OutputControl enumerated constants. To set a variable to one of these constants, refer to the MccDag object and the C8536OutputControl enumeration (for example, variable = MccDag.C8536OutputControl.HighPulseOnTc, variable = MccDag.C8536OutputControl.ToggleOnTc, etc.).

HighPulseOnTc	Output transitions from low to high for one clock pulse on terminal count
ToggleOnTc	Output changes state on the terminal count.
HighUntilTc	Output transitions to high at the start of counting then goes low on the terminal count.

### trigType parameter values

All of the trigType settings are MccDaq.C8536TriggerType enumerated constants. To set a variable to one of these constants, refer to the MccDaq object and the C8536TriggerType enumeration (for example, variable = MccDaq.C8536OutputControl.HighPulseOnTc, variable = MccDaq.C8536OutputControl.ToggleOnTc, etc.).

HWStartTrig	The first trigger on the counter's trigger input initiates loading of the initial count. Counting proceeds from the initial count.
HWRetrig	Every trigger on the counter's trigger input initiates loading of the initial count. Counting proceeds from the initial count.
SWStartTrig	The <a href="#">CLoad()</a> method initiates loading of the initial count. Counting proceeds from the initial count.

# C8536Init() method

Initializes the counter linking features of an 8536 counter chip. The linking of counters 1 and 2 must be accomplished prior to enabling the counters.

Refer to the Zilog 8536 product specification for a description of the hardware affected by this mode. This document is available on our web site at <http://www.mccdaq.com/PDFmanuals/Z8536.pdf>.



Member of the [MccBoard](#) class.

## Function Prototype

```
VB .NET
Public Function C8536Init(ByVal chipNum As Integer, ByVal ctrlOutput As MccDag.CtrlOutput) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo C8536Init(int chipNum, MccDag.CtrlOutput ctrlOutput)
```

## Parameters

- 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. Set it to one of the constants in the "[ctrlOutput parameter values](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## ctrlOutput parameter values

All of the ctrlOutput settings are MccDag.CtrlOutput enumerated constants. To set a variable to one of these constants, you must refer to the MccDag object and the CtrlOutput enumeration (for example, variable = MccDag.CtrlOutput.NotLinked, variable = MccDag.CtrlOutput.GateCtr2, etc.).

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 the counter #2 clock input.

## C9513Config() method

Sets all of the configurable options of a 9513 counter. For more information, see the AM9513A data sheet in accompanying [9513A.pdf](#) file located in the *Documents* subdirectory where the UL is installed (C:/Program files/Measurement Computing/DAQ by default).



Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function C9513Config(ByVal counterNum As Integer, ByVal gateControl As MccDaq.GateControl, ByVal counterEdge As MccDaq.CountEdge, ByVal counterSource As MccDaq.CounterSource, ByVal specialGate As MccDaq.OptionState, ByVal reload As MccDaq.Reload, ByVal recycleMode As MccDaq.RecycleMode, ByVal bcdMode As MccDaq.BCDMode, ByVal countDirection As MccDaq.CountDirection, ByVal outputControl As MccDaq.C9513OutputControl) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo C9513Config(int counterNum, MccDaq.GateControl gateControl, MccDaq.CountEdge counterEdge, MccDaq.CounterSource counterSource, MccDaq.OptionState specialGate, MccDaq.Reload reload, MccDaq.RecycleMode recycleMode, MccDaq.BCDMode bcdMode, MccDaq.CountDirection countDirection, MccDaq.C9513OutputControl outputControl)
```

### Parameters

*counterNum*

Counter number (1 - n) where n is the number of counters on the board. (For example, a CIO-CTR5 has 5, a CIO-CTR10 has 10, etc. See board specific info).

*gateControl*

Sets the gating response for level, edge, etc. Set it to one of the constants in the "[gateControl parameter values](#)" section below.

*counterEdge*

Which edge to count. Referred to as "Source Edge" in the 9513 data book. Can be set to *POSITIVEEDGE* (count on rising edge) or *NEGATIVEEDGE* (count on falling edge).

*counterSource*

Each counter may be set to count from one of 16 internal or external sources. Set it to one of the constants in the "[counterSource parameter values](#)" section below.

*specialGate*

Special gate may be enabled ([MccDaq.OptionState.Enabled](#)) or disabled ([MccDaq.OptionState.Disabled](#)).

*reload*

Reload the counter from the load register (reload = [MccDaq.Reload.LoadReg](#)) or alternately load from the load register, then the hold register (reload = [MccDaq.Reload.LoadAndHoldReg](#)).

*recycleMode*

Execute once ([MccDaq.RecycleMode.OneTime](#)) or reload and recycle ([MccDaq.RecycleMode.Recycle](#)) to count repetitively.

*bcdMode*

Counter may operate in binary coded decimal count ([MccDaq.BCDMode.BCDCount](#)) or binary count ([MccDaq.BCDMode.BinaryCount](#)).

*countDirection*

AM9513 may count up ([MccDaq.CountDirection.CountUp](#)) or down ([MccDaq.CountDirection.CountDown](#)).

*outputControl*

The type of output desired. Set it to one of the constants in the "[outputControl parameter values](#)" section below.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.



## gateControl parameter values

All of the gateControl settings are MccDaq.GateControl enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the GateControl enumeration (for example, variable = MccDaq.GateControl.NoGate, variable = MccDaq.GateControl.AhITcPrevCtr, etc.).

NoGate	No gating
AhITcPrevCtr	Active high TCN -1
AhINextGate	Active High Level GATE N + 1
AhIPrevGate	Active High Level GATE N - 1
AhIGate	Active High Level GATE N
AllGate	Active Low Level GATE N
AheGate	Active High Edge GATE N
Alegate	Active Low Edge GATE N

## counterSource parameter values

All of the counterSource settings are MccDaq.CounterSource enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the CounterSource enumeration (for example, variable = MccDaq.CounterSource.TcPrevCtr, variable = MccDaq.CounterSource.CtrInput1, etc.).

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	GATE1
Gate2	GATE2
Gate3	GATE3
Gate4	GATE4
Gate5	GATE5
Freq1	F1
Freq2	F2
Freq3	F3
Freq4	F4
Freq5	F5

## outputControl parameter values

All of the outputControl settings are MccDaq.9513OutputControl enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the 9513OutputControl enumeration (for example, variable = MccDaq.9513OutputControl.AlwaysLow, variable = MccDaq.9513OutputControl.HighPulseOnTc, etc.).

AlwaysLow	AlwaysLow
HighPulseOnTc	High pulse on Terminal Count
ToggleOnTc	TC Toggled
Disconnected	Inactive, Output High Impedance
LowPulseOnTc	Active Low Terminal Count Pulse
3, 6, 7	(numeric values) Illegal

## Notes

- The information provided here and in [C9513Init\(\)](#) will only help you understand how Universal Library syntax corresponds to the 9513 data sheet (refer to the accompanying 9513A.pdf file located in the *Documents* subdirectory of the installation). It is not a substitute for the data sheet. You cannot program and use a 9513 counter/timer without it.

## C9513Init() method

Initializes all of the chip level features of a 9513 counter chip. This method can only be used with 9513 counters. For more information see the AM9513A data sheet in accompanying [9513A.pdf](#) file located in the *Documents* subdirectory where the UL is installed (C:/Program files/Measurement Computing/DAQ by default).



Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function C9513Init(ByVal chipNum As Integer, ByVal foutDivider As Integer, ByVal foutSource As MccDaq.CounterSource,  
ByVal compare1 As MccDaq.CompareValue, ByVal compare2 As MccDaq.CompareValue,  
ByVal timeOfDay As MccDaq.TimeOfDay As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo C9513Init(int chipNum, int foutDivider, MccDaq.CounterSource foutSource,  
MccDaq.CompareValue compare1, MccDaq.CompareValue compare2, MccDaq.TimeOfDay timeOfDay)
```

### Parameters

*chipNum*

Specifies which 9513 chip is to be initialized. For a CTR05 board, set to 1. For a CTR10 board, set to either 1 or 2, and for a CTR20 set to 1-4

*foutDivider*

F-Out divider (0-15). If set to 0, foutDivider is the rate of foutSource divided by 16. If set to a number between 1 and 15, foutDivider is the rate of foutSource divided by foutDivider

*foutSource*

Specifies source of the signal for F-Out signal. Set it to one of the constants in the "[foutSource parameter values](#)" section below.

*compare1*

MccDaq.CompareValue.Enabled or MccDaq.CompareValue.Disabled

*compare2*

MccDaq.CompareValue.Enabled or MccDaq.CompareValue.Disabled

*timeOfDay*

MccDaq.TimeOfDay.Disabled, or three different enabled settings. Set it to one of the constants in the "[timeOfDay parameter values](#)" section below.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## foutSource parameter values

All of the foutSource settings are MccDaq.CounterSource enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the CounterSource enumeration (for example, variable = MccDaq.CounterSource.CtrInout1, variable = MccDaq.CounterSource.CtrInput2, etc.).

<b>foutSource</b>	<b>9513 Data Sheet Equivalent</b>
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	GATE1
Gate2	GATE2
Gate3	GATE3
Gate4	GATE4
Gate5	GATE5
Freq1	F1
Freq2	F2
Freq3	F3
Freq4	F4
Freq5	F5

## timeOfDay parameter values

All of the timeOfDay settings are MccDaq.TimeOfDay enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the TimeOfDay enumeration (for example, variable = MccDaq.TimeOfDay.Disable, variable = MccDaq.TimeOfDay.One, etc.).

<b>timeOfDay</b>	<b>9513 Data Sheet Equivalent</b>
Disabled	TOD Disabled
One	TOD Enabled/5 Input
Two	TOD Enabled/6 Input
Three	TOD Enabled/10 Input
<b>No parameters for:</b>	<b>9513 Data Sheet Equivalent</b>
0 (FOUT on)	FOUT Gate
0 (Data bus matches board)	Data Bus Width
1 (Disable Increment)	Data Pointer Control
1 (BCD Scaling)	Scalar Control

## Notes

- The information provided here and in [C9513Config\(\)](#) will only help you understand how Universal Library for .NET syntax corresponds to the 9513 data sheet (refer to the accompanying [9513A.pdf](#) file located in the *Documents* subdirectory of the installation). It is not a substitute for the data sheet. You cannot program and use a 9513 without it.

## CClear() method

Clears a scan counter value (sets it to zero). This method only works with counter boards that have counter scan capability.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function CClear(ByVal counterNum As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo CClear(int counterNum)
```

## Parameters

*counterNum*

The counter to clear.

**Note:** This parameter is zero-based (the first counter number to clear is "0").

## Returns

- [Error code](#) or 0 if no errors

## CConfigScan() method

Configures a counter channel. This method only works with counter boards that have counter scan capability.

Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function CConfigScan(ByVal counterNum As Integer, ByVal mode As MccDaq.CounterMode, ByVal
debounceTime As MccDaq.CounterDebounceTime, ByVal debounceMode As MccDaq.CounterDebounceMode, ByVal
edgeDetection As MccDaq.CounterEdgeDetection, ByVal tickSize As MccDaq.CounterTickSize, ByVal
mapCounter As Integer) As MccDaq.ErrorInfo

Public Function CConfigScan(ByVal counterNum As Integer, ByVal mode As MccDaq.CounterMode, ByVal
debounceTime As MccDaq.CounterDebounceTime, ByVal debounceMode As MccDaq.CounterDebounceMode, ByVal
edgeDetection As MccDaq.CounterEdgeDetection, ByVal tickSize As Integer, ByVal mapCounter As Integer)
As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo CConfigScan(int counterNum, MccDaq.CounterMode mode, MccDaq.CounterDebounceTime
debounceTime, MccDaq.CounterDebounceMode debounceMode, MccDaq.CounterEdgeDetection edgeDetection,
MccDaq.CounterTickSize tickSize, int mapCounter)

public MccDaq.ErrorInfo CConfigScan(int counterNum, MccDaq.CounterMode mode, MccDaq.CounterDebounceTime
debounceTime, MccDaq.CounterDebounceMode debounceMode, MccDaq.CounterEdgeDetection edgeDetection, int
tickSize, int mapCounter)
```

### Parameters

*counterNum*

The counter to set up. **Note:** This parameter is zero-based (the first counter number to set up is "0").

*mode*

Bit fields that control various options. All of the mode settings are [MccDaq.CounterMode](#) enumerated constants. Set it to one of the constants in the "[mode parameter values](#)" section below.

*debounceTime*

Used to bypass the debounce mode, or to set a channel's comparator output to one of 16 debounce times. Debounce is used to eliminate switch-induced transients typically associated with electromechanical devices including relays, proximity switches, and encoders.

All of the debounceTime settings are [MccDaq.CounterDebounceTime](#) enumerated constants. Set it to one of the constants in the "[debounceTime parameter values](#)" section below.

*debounceMode*

Sets the mode of the debounce module. The debounceMode settings are [MccDaq.CounterDebounceMode](#) enumerated constants. Set it to one of the constants in the "[debounceMode parameter values](#)" section below.

*edgeDetection*

Determines whether the rising edge or falling edge is to be detected. The edgeDetection settings are [MccDaq.CounterEdgeDetection](#) enumerated constants. The choices are RisingEdge and FallingEdge.

*tickSize*

Sets the tick size, which is the fundamental unit of time for period, pulsewidth, and timing measurements. All of the tickSize settings are [MccDaq.CounterTickSize](#) enumerated constants. Set it to one of the constants in the "[tickSize parameter values](#)" section below.

*mapCounter*

Used to select the mapped channel. A mapped channel is one of the counter input channels other than counterNum that can participate with the input signal of the counter defined by counterNum by gating the counter or decrementing the counter.

### Returns

- [Error code](#) or 0 if no errors

## mode parameter values

### ■ Totalize mode

Sets the specified counter to totalize mode. This mode may contain any combination of non-contradictory choices from the following list of options:

ClearOnRead	The counter counts up and is cleared at the beginning of every sample. By default, the counter counts up and only clears the counter at the start of a new scan command.
StopAtMax	The counter will stop at the top of its count. For the <a href="#">CIn32()</a> method, the top of the count depends on whether the Bit32 option is used. If it is, the top of the count is FFFFFFFF hex. If not, the top of the count is FFFF hex. By default, the counter counts upward and rolls over on the 32-bit boundary.
DecrementOn	Allows the mapped channel to decrement the counter. With this option, the main counter channel will increment the counter, and the mapped channel can be used to decrement the counter. By default, the counter decrement option is set to "off".  This mode is not compatible with <a href="#">CIn()</a> or <a href="#">CIn32()</a> . If a counter is configured for DecrementOn, calling CIn() or CIn32() for that counter will result in a <a href="#">BADCOUNTERMODE</a> error.
GatingOn	Selects gating "on." When "on", the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.  This mode is not compatible with <a href="#">CIn()</a> or <a href="#">CIn32()</a> . If a counter is configured for GatingOn, calling CIn() or CIn32() for that counter will result in a <a href="#">BADCOUNTERMODE</a> error.
LatchOnMap	Causes the count to be latched by the signal on the mapped channel. By default, the count is latched by the internal "start of scan" signal, so the count is updated each time it's read.  This mode is not compatible with <a href="#">CIn()</a> or <a href="#">CIn32()</a> . If a counter is configured for LatchOnMap, calling CIn() or CIn32() for that counter will result in a <a href="#">BADCOUNTERMODE</a> error.
Bit32	Selects a 32-bit counter for asynchronous mode. This argument value only affects counter resolution for asynchronous calls ( <a href="#">CIn()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn64()</a> ). Recommended for use only with CIn32(). (Using the Bit32 option with CIn() is not very useful, since the value returned by CIn() is only 16 bits. The effect is that the value returned by CIn() rolls over 4,294,967,295 times before stopping.) Refer to board-specific information for the product you are using for details on how this affects asynchronous reads on a specific device.
Bit48	Selects a 48-bit counter for asynchronous mode. This argument value only affects counter resolution for asynchronous calls ( <a href="#">CIn()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn64()</a> ). (Using the Bit48 option with CIn() and CIn32() is not very useful, since the value returned by CIn() is only 16 bits, and the value returned by CIn32() is only 32 bits. The effect is that the value returned by CIn() rolls over 4,294,967,295 times before stopping, and the value returned by CIn32() rolls over 65,535 times before stopping.) Refer to board-specific information for the product you are using for details on how this affects asynchronous reads on a specific device.
UpDownOn	Enables up/down counting mode.
RangeLimitOn	Enables range limit counting mode. In range limit mode, an upper and lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by loading the max limit register with the <a href="#">CLoad</a> , <a href="#">CLoad32</a> , or <a href="#">CLoad64</a> functions. The lower limit is always 0. When counting up, the counter freezes whenever the count reaches the value that was loaded into the max limit register.
NoRecycleOn	Enables non-recycle counting mode. In non-recycle mode, the counter is disabled whenever a count overflow or underflow takes place. The counter is re-enabled when a clear or a load operation is performed on the counter
ModuloNOn	Enables modulo-n counting mode. In modulo-n mode, an upper limit is set by loading the max limit register with a maximum count. When counting up, the counter will roll-over to 0 when the maximum count is reached, and then continue counting up. Likewise when counting down, the counter will roll over to the maximum count (in the max limit register) whenever the count reaches 0, and then continue counting down.

### ■ Encoder mode

Sets the specified counter to encoder measurement mode. This mode may contain any combination of non-contradictory choices from the following list of options:

EncoderModeX1	Sets the encoder measurement mode to X1.
EncoderModeX2	Sets the encoder measurement mode to X2.
EncoderModeX4	Sets the encoder measurement mode to X4.
EncoderModeLatchOnZ	Selects the Encoder Z mapped signal to latch the counter outputs. This allows the user to know the exact counter value when an edge is present on another counter.

EncoderModeClearOnZOn	Selects "clear on Z" on. The counter is cleared on the rising edge of the mapped (Z) channel. By default, the "ClearOnZ" option is off, and the counter is not cleared.
EncoderModeBit16	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with CIn().
EncoderModeBit32	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with CIn32(). (Using the EncoderModeBit32 option with CIn() is not very useful, since the value returned by CIn() is only 16 bits. The effect is that the value returned by CIn() rolls over 4,294,967,295 times before stopping.)
EncoderModeBit48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with CIn64(). (Using the EncoderModeBit48 option with CIn() and CIn32() is not very useful, since the value returned by CIn() is only 16 bits, and the value returned by CIn32() is only 32 bits. The effect is that the value returned by CIn() rolls over 4,294,967,295 times before stopping, and the value returned by CIn32() rolls over 65,535 times before stopping.)
EncoderModeRangeLimitOn	Enables Range Limit counting mode. In Range Limit mode, an upper and lower limit is set, mimicking limit switches in the mechanical counterpart. The upper limit is set by loading the max limit register with the <a href="#">CLoad</a> , <a href="#">CLoad32</a> , or <a href="#">CLoad64</a> functions. The lower limit is always 0. When counting up, the counter freezes whenever the count reaches the value that was loaded into the max limit register.
EncoderModeNoRecycleOn	Enables Non-recycle counting mode. In Non-recycle mode, the counter is disabled whenever a count overflow or underflow takes place. The counter is re-enabled when a clear or a load operation is performed on the counter
EncoderModeModuloNOn	Enables Modulo-N counting mode. In Modulo-N mode, an upper limit is set by loading the max limit register with a maximum count. When counting up, the counter will roll-over to 0 when the maximum count is reached, and then continue counting up. Likewise when counting down, the counter will roll over to the maximum count (in the max limit register) whenever the count reaches 0, and then continue counting down.

#### ■ Period mode

Sets the specified counter to period measurement mode. This mode may contain any combination of non-contradictory choices from the following list of options:

PeriodModeX1	The measurement is latched each time one complete period is observed.
PeriodModeX10	The measurement is latched each time 10 complete periods are observed.
PeriodModeX100	The measurement is latched each time 100 complete periods are observed.
PeriodModeX1000	The measurement is latched each time 1000 complete periods are observed.
PeriodModeGatingOn	<p>Selects gating "on." When "on", the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.</p> <p>This mode is not compatible with <a href="#">CIn()</a> or <a href="#">CIn32()</a>. If a counter is configured for PeriodModeGatingOn, calling CIn() or CIn32() for that counter will result in a BADCOUNTERMODE error.</p>
PeriodModeBit16	Selects a 16-bit counter for asynchronous mode. This argument value only affects counter resolution for asynchronous calls ( <a href="#">CIn()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn64()</a> ). Recommended for use only with CIn().
PeriodModeBit32	Selects a 32-bit counter for asynchronous mode. This argument value only affects counter resolution for asynchronous calls ( <a href="#">CIn()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn64()</a> ). Recommended for use only with CIn32(). (Using the PeriodModeBit32 option with CIn() is not very useful, since the value returned by CIn() is only 16 bits. The effect is that the value returned by CIn() rolls over at 64k 65,535 times before stopping.)
PeriodModeBit48	Selects a 48-bit counter for asynchronous mode. This argument value only affects counter resolution for asynchronous calls ( <a href="#">CIn()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn64()</a> ). Recommended for use only with CIn64(). (Using the PeriodModeBit48 option with CIn() and CIn32() is not very useful, since the value returned by CIn() is only 16 bits, and the value returned by CIn32() is only 32 bits. The effect is that the value returned by CIn() rolls over 4,294,967,295 times before stopping, and the value returned by CIn32() rolls over 65,535 times before stopping.)

## ■ PulseWidth mode

Sets the specified counter to Pulsewidth measurement mode. This mode may contain any combination of non-contradictory choices from the following list of options:

PulseWidthModeGatingOn	Selects gating "on." When "on", the counter is enabled when the mapped channel to gate the counter is high. When the mapped channel is low, the counter is disabled but holds the count value.  This mode is not compatible with <a href="#">CIn()</a> or <a href="#">CIn32()</a> . If a counter is configured for PulsewidthModeGatingOn, calling <a href="#">CIn()</a> or <a href="#">CIn32()</a> for that counter will result in a BADCOUNTERMODE error.
PulseWidthModeBit16	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with <a href="#">CIn()</a> .
PulseWidthModeBit32	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with <a href="#">CIn32()</a> . (Using the PulseWidthModeBit32 option with <a href="#">CIn()</a> and <a href="#">CIn32()</a> is not very useful, since the value returned by <a href="#">CIn()</a> is only 16 bits, and the value returned by <a href="#">CIn32()</a> is only 32 bits. The effect is that the value returned by <a href="#">CIn()</a> rolls over 4,294,967,295 times before stopping, and the value returned by <a href="#">CIn32()</a> rolls over 65,535 times before stopping.)
PulseWidthModeBit48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with <a href="#">CIn64()</a> . (Using the PulseWidthModeBit48 option with <a href="#">CIn()</a> and <a href="#">CIn32()</a> is not very useful, since the value returned by <a href="#">CIn()</a> is only 16 bits, and the value returned by <a href="#">CIn32()</a> is only 32 bits. The effect is that the value returned by <a href="#">CIn()</a> rolls over 4,294,967,295 times before stopping, and the value returned by <a href="#">CIn32()</a> rolls over 65,535 times before stopping.)

## ■ TIMING mode

Sets the specified counter to timing mode. This mode supports the following option:

TimingModeBit16	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with <a href="#">CIn()</a> .
TimingModeBit32	Selects a 32-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with <a href="#">CIn32()</a> . (Using the TimingModeBit32 option with <a href="#">CIn()</a> is not very useful, since the value returned by <a href="#">CIn()</a> is only 16 bits. The effect is that the value returned by <a href="#">CIn()</a> rolls over at 64k 65,535 times before stopping.)
TimingModeBit48	Selects a 48-bit counter for asynchronous mode. This argument value only affects <a href="#">CIn64()</a> , <a href="#">CIn32()</a> , and <a href="#">CIn()</a> . Recommended for use only with <a href="#">CIn64()</a> . (Using the TimingModeBit48 option with <a href="#">CIn()</a> and <a href="#">CIn32()</a> is not very useful, since the value returned by <a href="#">CIn()</a> is only 16 bits, and the value returned by <a href="#">CIn32()</a> is only 32 bits. The effect is that the value returned by <a href="#">CIn()</a> rolls over 4,294,967,295 times before stopping, and the value returned by <a href="#">CIn32()</a> rolls over 65,535 times before stopping.)

## debounceTime parameter values

Debounce500ns	Sets the counter channel's comparator output to 500 ns.
Debounce1500ns	Sets the counter channel's comparator output to 1500 ns.
Debounce3500ns	Sets the counter channel's comparator output to 3500 ns.
Debounce7500ns	Sets the counter channel's comparator output to 7500 ns.
Debounce15500ns	Sets the counter channel's comparator output to 15500 ns.
Debounce31500ns	Sets the counter channel's comparator output to 31500 ns.
Debounce63500ns	Sets the counter channel's comparator output to 63500 ns.
Debounce127500ns	Sets the counter channel's comparator output to 127500 ns.
Debounce100us	Sets the counter channel's comparator output to 100 us.
Debounce300us	Sets the counter channel's comparator output to 300 us.
Debounce700us	Sets the counter channel's comparator output to 700 us.
Debounce1500us	Sets the counter channel's comparator output to 1500 us.
Debounce3100us	Sets the counter channel's comparator output to 3100 us.
Debounce6300us	Sets the counter channel's comparator output to 6300 us.
Debounce12700us	Sets the counter channel's comparator output to 12700 us.
Debounce25500us	Sets the counter channel's comparator output to 25500 us.



### debounceMode parameter values

TriggerAfterStable	This mode rejects glitches and only passes state transitions after a specified period of stability (the debounce time). This mode is used with electromechanical devices like encoders and mechanical switches to reject switch bounce and disturbances due to a vibrating encoder that is not otherwise moving. The debounce time should be set short enough to accept the desired input pulse but longer than the period of the undesired disturbance.
TriggerBeforeStable	Use this mode when the input signal has groups of glitches and each group is to be counted as one. The trigger before stable mode will recognize and count the first glitch within a group but reject the subsequent glitches within the group if the debounce time is set accordingly. In this case the debounce time should be set to encompass one entire group of glitches.

### tickSize parameter values

Tick20pt83ns	Sets the counter channel's tick size to 20.83 ns.
Tick208pt3ns	Sets the counter channel's tick size to 208.3 ns.
Tick2083pt3ns	Sets the counter channel's tick size to 2083.3 ns.
Tick20833pt3ns	Sets the counter channel's tick size to 20833.3 ns.

## CFreqIn() method

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

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function CFreqIn(ByVal signalSource As MccDaq.SignalSource, ByVal gateInterval As Integer, ByRef count As Short, ByRef freq As Integer) As MccDaq.ErrorInfo

Public Function CFreqIn(ByVal signalSource As MccDaq.SignalSource, ByVal gateInterval As Integer, ByRef count As System.UInt16, ByRef freq As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo CFreqIn(MccDaq.SignalSource signalSource, int gateInterval, out short count, out int freq)

public MccDaq.ErrorInfo CFreqIn(MccDaq.SignalSource signalSource, int gateInterval, out ushort count, out int freq)
```

## Parameters

*signalSource*

Specifies the source of the signal to calculate the frequency from.

The signal to be measured is routed internally from the source specified by signalSource to the clock input of counter 5. On boards with more than one 9513 chip, there is more than one counter 5. Which counter 5 is used is also determined by signalSource. Set it to one of the constants in the "[signalSource parameter values](#)" section below.

The value of signalSource determines which chip will be used. CtrInput6 through CtrInput10, Freq6 through Freq10 and Gate6 through Gate9 indicate chip two will be used. The signal to be measured must be present at the chip two input specified by signalSource.

**Note:** The gating connection from counter 4 output to counter 5 gate must be made between counters 4 and 5 of *this chip* (refer to the [Notes](#) section below). Refer to board-specific information to determine valid values for your board.

*gateInterval*

Gating interval in milliseconds (must be > 0). Specifies the time, in milliseconds, that the counter will count. The optimum gateInterval depends on the frequency of the measured signal. The counter can count up to 65,535.

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 is too long, the counter will overflow and a FreqOverflow error will occur.

This method will not return until the gateInterval has expired. There is no background option. Under Windows, this means that window activity will stop for the duration of the call. Adjust the gateInterval so this does not pose a problem to your user interface.

*count*

The raw count.

*freq*

The measured frequency in Hz.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- count - Count that the frequency calculation is based on.
- freq - Measured frequency in Hz

## signalSource parameter values

All of the signalSource settings are MccDaq.SignalSource enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the SignalSource enumeration (for example, variable = MccDaq.SignalSource.CtrInput1, variable = MccDaq.SignalSource.Gate1, etc.).

One 9513 chip (Chip 1 used):	CtrInput1 through CtrInput5
	Gate1 through Gate4
	Freq1 through Freq5
Two 9513 chips (Chip 1 or Chip 2 used):	CtrInput1 through CtrInput10
	Gate1 through Gate 9 (excluding gate 5)
	Freq1 through Freq10
Four 9513 chips (Chips 1- 4 may be used):	CtrInput1 through CtrInput20
	Gate1 through Gate19 (excluding gates 5, 10, and 15)
	Freq1 through Freq20

## Notes

- This method requires an electrical connection between counter 4 output and counter 5 gate. This connection must be made between counters 4 and 5 *on the chip specified by signalSource*.
- [C9513Init\(\)](#) must be called for each chipNum that will be used by this method. The values of foutDivider, foutSource, compare1, compare2, and timeOfDay are irrelevant to this method and may be any value shown in the C9513Init() method description.
- If you select an external clock source for the counters, the gateInterval, count, and freq settings are only valid if the external source is 1 MHz. Otherwise, you need to scale the values according to the frequency of the external clock source.

For example, for an external clock source of 2 MHz, increase your gateInterval setting by a factor of 2, and also double the count and freq values returned when analyzing your results.

## CIn() method

Reads the current count from a counter.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function CIn(ByVal counterNum As Integer, ByRef count As Short) As MccDag.ErrorInfo  
Public Function CIn(ByVal counterNum As Integer, ByRef count As System.UInt16) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo CIn(int counterNum, out ushort count)  
public MccDag.ErrorInfo CIn(int counterNum, out short count)
```

## Parameters

*counterNum*

The counter to read the current count from. Valid values are 1 to 20, up to the number of counters on the board.

*count*

The counter value is returned here.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- Count: Refer to your BASIC manual for information on BASIC integer data types. -32,768 to 32,767 for BASIC languages. BASIC reads counters as:
  - -1 reads as 65,535
  - -32,768 reads as 32,768
  - 32,767 reads as 32,767
  - 2 reads as 2
  - 0 reads as 0
- CIn() vs [CIn32\(\)](#) vs [CIn64\(\)](#)

Although the CIn(), CIn32(), and CIn64() methods perform the same operation, CIn32() is the preferred method to use.

The only difference between the three is that CIn() returns a 16-bit count value, CIn32() returns a 32-bit value, and CIn64() returns a 64-bit value. Both CIn() and CIn32() can be used, but CIn64() is required whenever you need to read count values greater than 32-bits (counts >4,294,967,295) or the upper (more significant) bits will be truncated.

## CIn32() method

Reads the current count from a counter, and returns it as a 32 bit integer.

Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function CIn32(ByVal counterNum As Integer, ByRef count As Integer) As MccDaq.ErrorInfo
```

```
Public Function CIn32(ByVal counterNum As Integer, ByRef count As System.UInt32) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo CIn32(int counterNum, out uint count)
```

```
public MccDaq.ErrorInfo CIn32(int counterNum, out int count)
```

### Parameters

*counterNum*

The counter to read current count from. Valid values are 1 to *n*, where *n* is the number of counters on the board.

*count*

Current count value from selected counter.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

- [CIn\(\)](#) vs [CIn32\(\)](#) vs [CIn64\(\)](#)

Although the [CIn\(\)](#), [CIn32\(\)](#), and [CIn64\(\)](#) methods perform the same operation, [CIn32\(\)](#) is the preferred method to use.

The only difference between the three is that [CIn\(\)](#) returns a 16-bit count value, [CIn32\(\)](#) returns a 32-bit value, and [CIn64\(\)](#) returns a 64-bit value. Both [CIn\(\)](#) and [CIn32\(\)](#) can be used, but [CIn64\(\)](#) is required whenever you need to read count values greater than 32-bits (counts >4,294,967,295) or the upper (more significant) bits will be truncated.

## CIn64() method

Reads the current count from a counter, and returns it as a 64-bit double word. This function is not supported in Visual Basic, since no appropriate data type is available to accept the Count argument in those languages.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function CIn64(ByVal counterNum As Integer, ByRef count As Integer) As MccDag.ErrorInfo  
Public Function CIn64(ByVal counterNum As Integer, ByRef count As System.UInt32) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo CIn64(int counterNum, out uint count)  
public MccDag.ErrorInfo CIn64(int counterNum, out int count)
```

## Parameters

*counterNum*

The counter to read current count from. Valid values are 1 to  $n$ , where  $n$  is the number of counters on the board.

*count*

Current count value from the selected counter.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- [CIn\(\)](#) vs [CIn32\(\)](#) vs [CIn64\(\)](#)

Although the [CIn\(\)](#), [CIn32\(\)](#), and [CIn64\(\)](#) methods perform the same operation, [CIn32\(\)](#) is the preferred method to use.

The only difference between the three is that [CIn\(\)](#) returns a 16-bit count value, [CIn32\(\)](#) returns a 32-bit value, and [CIn64\(\)](#) returns a 64-bit value. Both [CIn\(\)](#) and [CIn32\(\)](#) can be used, but [CIn64\(\)](#) is required whenever you need to read count values greater than 32-bits (counts >4,294,967,295) or the upper (more significant) bits will be truncated.

## CInScan() method

Scans a range of scan counter channels, and stores the samples in an array. This method only works with counter boards that have counter scan capability.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function CInScan(ByVal firstCtr As Integer, ByVal lastCtr As Integer, ByVal numPoints As Integer, ByRef rate As Integer, ByVal memHandle As IntPtr, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo CInScan(int firstCtr, int lastCtr, int numPoints, ref int rate, IntPtr memHandle, MccDag.ScanOptions Options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function CInScan(ByVal firstCtr As Integer, ByVal lastCtr As Integer, ByVal numPoints As Integer, ByRef rate As Integer, ByVal memHandle As Integer, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo CInScan(int firstCtr, int lastCtr, int numPoints, int rate, int memHandle, MccDag.ScanOptions options)
```

## Parameters

*firstCtr*

First counter channel of the scan. This parameter is zero-based, so the first counter number is "0".

*lastCtr*

Last counter channel of the scan. This parameter is zero-based, so the first counter number is "0".

The maximum allowable channel for both firstCtr and lastCtr depends on how many scan counters are available on the Measurement Computing device in use.

*numPoints*

Number of counter samples to collect. Specifies the total number of counter 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 / (firstCtr – lastCtr + 1).

*rate*

The rate at which samples are taken – the counts are latched and saved in board memory, in samples per second.

Rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*memHandle*

The handle for the Windows buffer to store data (Windows). This buffer must have been previously allocated with the [WinBufAlloc32Ex\(\)](#) method.

*options*

Bit fields that control various options. All of the option settings are [MccDag.ScanOptions](#) enumerated constants. Set it to one of the constants in the "[options parameter values](#)" section below.

## Returns

- [Error code](#) or 0 if no errors
- rate – the actual sampling rate used.
- memHandle – the collected counter data returned via the Windows buffer.

## options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ScanOptions enumeration (for example, variable = MccDaq.ScanOptions.Continuous, variable = MccDaq.ScanOptions.Background, etc.).

Background	When the Background option is used, control returns immediately to the next line in your program, and the data collection from the counters into the buffer continues in the background. If the Background option is not used, the CInScan() method does not return to your program until all of the requested data has been collected and returned to the buffer.
Continuous	This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of the buffer and begins again. The only way to stop this operation is by using <a href="#">StopBackground()</a> with CtrFunction. Normally, you should use this option with Background so that your program regains control.
Ctr16Bit	Sets the counter resolution to 16-bits. When using devices that return data in a 16-bit format, create the buffer using <a href="#">WinBufAllocEx()</a> .
Ctr32Bit	Sets the counter resolution to 32-bits. When using devices that return data in a 32-bit format, create the buffer using <a href="#">WinBufAlloc32Ex()</a> .
Ctr48Bit	Sets the counter resolution to 48-bits. When using devices that return data in a 64-bit format, create the buffer using <a href="#">WinBufAlloc64Ex()</a> .
ExtClock	If this option is specified, conversions will be controlled by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal (refer to board-specific information in the <i>UL User's Guide</i> ). When this option is used the rate parameter is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to a transfer mode that will allow the maximum conversion rate to be attained unless otherwise specified.
ExtTrigger	If this option is specified, sampling does not begin until the trigger condition is met. You can set the trigger condition to rising edge, falling edge, or the level of the digital trigger input with the <a href="#">SetTrigger()</a> method. Refer to board-specific information in the <i>UL User's Guide</i> .
HighResRate	Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>rate</i> parameter above).



# CLoad() method

Loads the specified counter's Load, Hold, Alarm, QuadCount, QuadPreset or PreScaler register with a count. When loading a counter with a starting value, it is never loaded directly into the counter's count register. Rather, it is loaded into the load or hold register. From there, the counter, after being enabled, loads the count from the appropriate register, generally on the first valid pulse.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function CLoad(ByVal regNum As MccDaq.CounterRegister, ByVal loadValue As Integer) As MccDaq.ErrorInfo

Public Function CLoad(ByVal regNum As MccDaq.CounterRegister, ByVal loadValue As System.UInt32) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo CLoad(MccDaq.CounterRegister regNum, uint loadValue)

public MccDaq.ErrorInfo CLoad(MccDaq.CounterRegister regNum, int loadValue)
```

## Parameters

*regNum*

The register to load the count to. Set it to one of the constants in the "[RegNum parameter values](#)" section below.

*loadValue*

The value to be loaded. This value must be between 0 and  $2^{\text{resolution}} - 1$  of the counter. For example, a 16-bit counter is  $2^{16} - 1$ , or 65,535. Refer to the [notes on Basic integer types](#).

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## regNum parameter values

All of the regNum settings are MccDaq.CounterRegister enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the CounterRegister enumeration (for example, variable = MccDaq.CounterRegister.LoadReg0, variable = MccDaq.CounterRegister.HoldReg1, etc.).

LoadReg0 to LoadReg20	Load registers 1 to 20. This can span many chips.
HoldReg1 to HoldReg20	Hold registers 0 to 20. This can span several chips. (9513 only)
Alarm1Chip1	Alarm register 1 of the first counter chip. (9513 only)
Alarm2Chip1	Alarm register 2 of the first counter chip. (9513 only)
Alarm1Chip2	Alarm register 1 of the second counter chip. (9513 only)
Alarm2Chip2	Alarm register 2 of the second counter chip. (9513 only)
Alarm1Chip3	Alarm register 1 of the third counter chip. (9513 only)
Alarm2Chip3	Alarm register 2 of the third counter chip. (9513 only)
Alarm1Chip4	Alarm register 1 of the fourth counter chip. (9513 only)
Alarm2Chip4	Alarm register 2 of the fourth counter chip. (9513 only)
QuadCount1 to QuadCount4	Used to initialize the counter. (LS7266 only)
QuadPreset1 to QuadPreset4	Used to set the upper limit of the counter in some modes. (LS7266 only)
QuadPrescaler1 to QuadPrescaler4	Used for clock filtering (valid values: 0 to 255). (LS7266 only)
MaxLimitReg0 to MaxLimitReg7	Max limit register (USB-QUAD08 only)

## Notes

- You cannot load a count-down-only counter with less than 2.
- Counter types: Several counter types are supported. Refer to the counter chip's data sheet for the registers available for a counter type.
- CLoad() vs [CLoad32\(\)](#)

The CLoad() and CLoad32() perform the same operation. These methods differ in that CLoad() loads a 16-bit count value, while CLoad32() loads a 32-bit value. The only time you need to use CLoad32() is to load counts that are larger than 32-bits (counts >4,294,967,295).

# CLoad32() method

Loads the specified counter's CurrentCount, Preset, or PreScaler register with a count.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET:

```
Public Function CLoad32 (ByVal regNum As MccDaq.CounterRegister, ByVal loadValue As Integer) As MccDaq.ErrorInfo

Public Function CLoad32 (ByVal regNum As MccDaq.CounterRegister, ByVal loadValue As System.UInt32) As MccDaq.ErrorInfo
```

C# .NET:

```
public MccDaq.ErrorInfo CLoad32(MccDaq.CounterRegister regNum, uint loadValue)

public MccDaq.ErrorInfo CLoad32(MccDaq.CounterRegister regNum, int loadValue)
```

## Parameters

*regNum*

The register to load the value into. Set it to one of the constants in the "[regNum parameter values](#)" section below.

*loadValue*

The value to be loaded into regNum.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### regNum parameter values

All of the regNum settings are MccDaq.CounterRegister enumerated constants. To set a variable to one of these constants, refer to the MccDaq object and the CounterRegister enumeration (for example, variable = MccDaq.CounterRegister.LoadReg0, variable = MccDaq.CounterRegister.HoldReg1, etc.).

LoadReg0 to LoadReg20	Load registers 0 to 20. This can span many chips.
HoldReg1 to HoldReg20	Hold registers 1 to 20. This can span several chips. (9513 only)
Alarm1Chip1	Alarm register 1 of the first counter chip. (9513 only)
Alarm2Chip1	Alarm register 2 of the first counter chip. (9513 only)
Alarm1Chip2	Alarm register 1 of the second counter chip. (9513 only)
Alarm2Chip2	Alarm register 2 of the second counter chip. (9513 only)
Alarm1Chip3	Alarm register 1 of the third counter chip. (9513 only)
Alarm2Chip3	Alarm register 2 of the third counter chip. (9513 only)
Alarm1Chip4	Alarm register 1 of the fourth counter chip. (9513 only)
Alarm2Chip4	Alarm register 2 of the fourth counter chip. (9513 only)
QuadCount1 to QuadCount4	Used to initialize the counter. (LS7266 only)
QuadPreset1 to QuadPreset4	Used to set the upper limit of the counter in some modes. (LS7266 only)
QuadPreScaler1 to QuadPreScaler4	Used for clock filtering (valid values: 0 to 255). (LS7266 only)
MaxLimitReg0 to MaxLimitReg7	Max limit register (USB-QUAD08 only)

## Notes

- [CLoad\(\)](#) vs CLoad32()

Although the CLoad() and CLoad32() methods perform the same operation, CLoad32() is the preferred method to use. The only difference between the two is that CLoad() loads a 16-bit count value and CLoad32() loads a 32-bit value. The only time you need to use CLoad32() is to load counts that are larger than 16-bits (counts > 65,535).

# CLoad64() method

Loads the specified counter's CurrentCount, Preset, or PreScaler register with a count.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET:

```
Public Function CLoad64(ByVal regNum As MccDaq.CounterRegister, ByVal loadValue As Long) As MccDaq.ErrorInfo

Public Function CLoad64(ByVal regNum As MccDaq.CounterRegister, ByVal loadValue As ULong) As MccDaq.ErrorInfo
```

C# .NET:

```
public MccDaq.ErrorInfo CLoad64(MccDaq.CounterRegister regNum, uint loadValue)

public MccDaq.ErrorInfo CLoad64(MccDaq.CounterRegister regNum, int loadValue)
```

## Parameters

*regNum*

The register to load the value into. Set it to one of the constants in the "[regNum parameter values](#)" section below.

*loadValue*

The value to be loaded into regNum.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### regNum parameter values

All of the regNum settings are MccDaq.CounterRegister enumerated constants. To set a variable to one of these constants, refer to the MccDaq object and the CounterRegister enumeration (for example, variable = MccDaq.CounterRegister.LoadReg0, variable = MccDaq.CounterRegister.HoldReg1, etc.).

LoadReg0 to LoadReg20	Load registers 0 to 20. This can span many chips.
HoldReg1 to HoldReg20	Hold registers 1 to 20. This can span several chips. (9513 only)
Alarm1Chip1	Alarm register 1 of the first counter chip. (9513 only)
Alarm2Chip1	Alarm register 2 of the first counter chip. (9513 only)
Alarm1Chip2	Alarm register 1 of the second counter chip. (9513 only)
Alarm2Chip2	Alarm register 2 of the second counter chip. (9513 only)
Alarm1Chip3	Alarm register 1 of the third counter chip. (9513 only)
Alarm2Chip3	Alarm register 2 of the third counter chip. (9513 only)
Alarm1Chip4	Alarm register 1 of the fourth counter chip. (9513 only)
Alarm2Chip4	Alarm register 2 of the fourth counter chip. (9513 only)
QuadCount1 to QuadCount4	Used to initialize the counter. (LS7266 only)
QuadPreset1 to QuadPreset4	Used to set the upper limit of the counter in some modes. (LS7266 only)
QuadPreScaler1 to QuadPreScaler4	Used for clock filtering (valid values: 0 to 255). (LS7266 only)
MaxLimitReg0 to MaxLimitReg7	Max limit register (USB-QUAD08 only)

## Notes

- [CLoad\(\)](#) vs CLoad64()

Although the CLoad() and CLoad32() methods perform the same operation, CLoad64() is the preferred method to use. The only difference between the two is that CLoad() loads a 16-bit count value and CLoad64() loads a 64-bit value. The only time you need to use CLoad64() is to load counts that are larger than 32-bits (counts >4,294,967,295).

# CStatus() method

Returns status information about the specified counter (7266 counters only).

Member of the [MccBoard](#) class.

## Function Prototype

```
VB .NET
Public Function CStatus(ByVal counterNum As Integer, ByRef statusBits As MccDag.StatusBits) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo CStatus(int counterNum,out MccDag.StatusBits statusBits)
```

## Parameters

- CounterNum*  
The number of the counter whose status bits you want to read. Valid values are 1 to *n*, where *n* is the number of counters on the board.
- statusBits*  
Current status from selected counter is returned here. The status consists of individual bits that indicate various conditions within the counter. Set it to one of the constants in the "[statusBits parameter values](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### statusBits parameter values

All of the statusBits settings are MccDag.StatusBits enumerated constants. To set a variable to one of these constants, you must refer to the MccDag object and the StatusBits enumeration (for example, variable = MccDag.StatusBits.UnderFlow, variable = MccDag.StatusBits.Overflow, and so on).

Compare	Set to 1 whenever the count matches the preset register. Is cleared to 0 whenever CStatus () is called.
Error	Set to 1 whenever an error occurs due to excessive noise on the input. Is cleared to 0 by calling <a href="#">C7266Config()</a> .
Index	Set to 1 when index is valid. Is cleared to 0 when index is not valid.
Overflow	Set to 1 whenever the count increments past it's upper limit. Is cleared to 0 whenever CStatus() is called.
Sign	Set to 1 when the MSB of the count is 1. Is cleared to 0 whenever the MSB of the count is set to 0.
Underflow	Set to 1 whenever the count decrements past 0. Is cleared to 0 whenever CStatus() is called.
UpDown	Set to 1 when counting up. Is cleared to 0 when counting down

## CStoreOnInt() method

Installs an interrupt handler that will store the current count whenever an interrupt occurs. This method can only be used with 9513 counters. This method will continue to operate in the background until either `intCount` is satisfied or [StopBackground\(\)](#) with `CtrFunction` is called.

Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function CstoreOnInt(ByVal intCount As Integer, cntrControl As MccDaq.CounterControl[], ByVal memHandle As IntPtr) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo CstoreOnInt(int intCount, MccDaq.CounterControl cntrControl, IntPtr memHandle)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function CstoreOnInt(ByVal intCount As Integer, ByRef cntrControl As MccDaq.CounterControl, ByVal memHandle As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo CstoreOnInt(int intCount, ref MccDaq.CounterControl cntrControl, int memHandle)
```

### Parameters

*intCount*

The counters will be read every time an interrupt occurs, until `IntCount` number of interrupts have occurred. If `IntCount = 0`, the method will run until [StopBackground\(\)](#) is called. (Refer below to *memHandle*).

*cntrControl*

The array should have an element for each counter on the board. (5 elements for a CTR05 device, 10 elements for a CTR10 device, and so on). Each element corresponds to a counter channel. Each element should be set to either `MccDaq.CounterControl.Disabled` or `MccDaq.CounterControl.Enabled`

All channels that are set to `MccDaq.CounterControl.Enabled` will be read when an interrupt occurs.

*memHandle*

Handle for Windows buffer. If `intCount` is non-zero, the buffer referenced by `memHandle` must be of sufficient size to hold (`intCount × Number of Counters`) points.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

If the Library Revision is set to 4.0 or greater, the following code changes are required:

- If `intCount` is non-zero, the buffer referenced by `memHandle` must be able to hold (`intCount × Number of Counters`) points. For example, if you set *intCount* to 100 for a CTR05 device, you must allocate the size of the buffer to be  $(100 \times 5) = 500$ . This new functionality keeps the user application from having to move the data out of the buffer for every interrupt, before it is overwritten. Now, for each interrupt, the counter values will be stored in adjacent memory locations in the buffer.
- **Important:** Allocate the proper buffer size for non-zero `intCount` settings. Specifying `intCount` as a non-zero value and failing to allocate the proper sized buffer results in a runtime error. There is no way for the Universal Library to determine if the buffer has been allocated with the proper size.

If `intCount = 0`, the functionality is unchanged.

# PulseOutStart() method

Starts a timer to generate digital pulses at a specified frequency and duty cycle. Use [PulseOutStop\(\)](#) to stop the output. Use this method with counter boards that have a timer-type counter.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function PulseOutStart(ByVal timerNum As Integer, ByRef frequency As Double, ByRef dutyCycle As Double, ByVal pulseCount As Integer, ByRef initialDelay As Double, ByVal idleState As MccDaq.IdleState, ByVal options As MccDaq.PulseOutOptions) As MccDaq.ErrorInfo

Public Function PulseOutStart(ByVal timerNum As Integer, ByRef frequency As Double, ByRef dutyCycle As Double, ByVal pulseCount As UInteger, ByRef initialDelay As Double, ByVal idleState As MccDaq.IdleState, ByVal options As MccDaq.PulseOutOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo PulseOutStart(int timerNum, ref double frequency, ref double dutyCycle, uint pulseCount, ref double initialDelay, MccDaq.IdleState idleState, MccDaq.PulseOutOptions options)

public MccDaq.ErrorInfo PulseOutStart(int timerNum, ref double frequency, ref double dutyCycle, int pulseCount, ref double initialDelay, MccDaq.IdleState idleState, MccDaq.PulseOutOptions options)
```

## Parameters

*timerNum*

The timer to start output pulses. Valid values are zero (0) up to the number of timers on the board – 1.

*frequency*

The desired square wave frequency. The timer clock will be divided down by integer values to produce the frequency. The actual frequency output will be returned. Valid values are dependent on the timer's clock and the timer resolution.

*dutyCycle*

The width of the pulse divided by the pulse period. This ratio is used with the frequency value to determine the pulse width and the interval between pulses.

*pulseCount*

The number of pulses to generate. Setting the pulse count to zero will result in pulses being generated until the [PulseOutStop\(\)](#) method is called.

*initialDelay*

The amount of time to delay before starting the timer output after enabling the output.

*idleState*

The resting state of the output. Set it to one of the constants in the "[idleState parameter values](#)" section below.

*options*

Reserved for future use.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## idleState parameter values

All of the idleState settings are MccDaq.IdleState enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the IdleState enumeration, for example "variable = MccDaq.IdleState.IdleHigh".

IdleHigh	Sets the output resting state high.
IdleLow	Sets the output resting state low.

## PulseOutStop() method

Stops a timer output. Use [PulseOutStart\(\)](#) to start the output. Use this method with counter boards that have a timer-type counter.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function PulseOutStop (ByVal timerNum As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo PulseOutStop(int timerNum)
```

## Parameters

*timerNum*

The timer to stop. Valid values are zero (0) up to the number of timers on the board – 1.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## TimerOutStart() method

Starts a timer square wave output. Use [TimerOutStop\(\)](#) to stop the output. Use this method with counter boards that have a timer-type counter.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function TimerOutStart(ByVal timerNum As Integer, ByRef frequency As Double) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo TimerOutStart(int timerNum, double frequency)
```

## Parameters

*timerNum*

The timer to output the square wave from. Valid values are zero up to the number of timers on the board – 1.

*frequency*

The desired square wave frequency. The timers clock will be divided down by integer values to produce the frequency. The actual frequency output will be returned. Valid values are dependent on the timer's clock and the timer resolution.

## Returns

- [Error code](#) or 0 if no errors
- frequency – the actual frequency set.



## TimerOutStop() method

Stops a timer square wave output. Use [TimerOutStart\(\)](#) to stop the output. Use this method with counter boards that have a timer-type counter.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function TimerOutStop(ByVal timerNum As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo TimerOutStop(int timerNum)
```

## Parameters

*timerNum*

The timer to stop. Valid values are zero up to the number of timers on the board – 1.

## Returns

- [Error code](#) or 0 if no errors

## ConvertFile() method

Converts a binary log file to a comma-separated values (.CSV) text file or another text file format that you specify.

Member of the [DataLogger](#) class.

## Function Prototype

VB .NET

```
Public Function ConvertFile(ByRef destFileName As String, ByVal startSample As Integer, ByVal count As Integer, ByVal delimiter As MccDaq.FieldDelimiter) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo ConvertFile(string destFileName, int startSample, int count, MccDaq.FieldDelimiter delimiter)
```

## Parameter

*destFileName*

The name and destination path of the converted file. Use the file extension of the file type that you want to create.

*startSample*

The first sample to read.

*count*

The number of samples to read.

*delimiter*

Specifies the character to use between fields in the converted file.

All of the delimiter settings are MccDaq.FieldDelimiter enumerated constants. Choices are MccDaq.FieldDelimiter.Comma, MccDaq.FieldDelimiter.Semicolon, MccDaq.FieldDelimiter.Space, and MccDaq.FieldDelimiter.Tab.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- Time stamp data is stored according to the timeZone preference and timeFormat preference. Refer to [SetPreferences\(\)](#).

Time stamps in the converted file may be in either 12-hour or 24-hour format based on the value of the timeFormat preference. Time stamps can optionally be converted to local time based on the value of the timeZone preference.

- AI temperature data is returned according to the units preference. Refer to [SetPreferences\(\)](#).

The units preference is only applied to the AI data if the data was logged as temperature data. Refer to [GetAIInfo\(\)](#). This value is ignored if the AI data was logged as raw data.

The units preference is always applied to CJC data, since it is always logged as temperature data.

- If the destFileName argument ends with a .csv extension, the delimiter parameter must be set to MccDaq.FieldDelimiter.Comma. Otherwise, an [INVALIDDELIMITER](#) error is returned.

You can open a comma-separated values text file (.csv) directly in Microsoft Excel. Text files with extensions other than .csv can only be imported into Excel.

## FileName property

Returns the file name associated with the current instance of the DataLogger class.

Member of the [DataLogger](#) class.

## Property prototype

VB .NET

```
Public Shared ReadOnly Property DataLogger As String
```

C# .NET

```
public string FileName [get]
```

## GetAIChannelCount() method

Returns the total number of analog channels that were logged in a binary file.

Member of the [DataLogger](#) class.

### Function Prototype

VB .NET

```
Public Function GetAIChannelCount(ByRef aiCount As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetAIChannelCount(ref int aiCount)
```

### Parameter

*aiCount*

The number of analog input channels logged in the file.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- aiCount – Returns the number of analog input channels logged in the binary file.

## GetAIInfo() method

Returns the channel number and unit value of each analog input channel logged in a binary file.

Member of the [DataLogger](#) class.

## Function Prototype

VB .NET

```
Public Function GetAIInfo(ByRef channelNumbers As Integer, ByRef units As MccDaq.LoggerUnits, As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetAIInfo(ref int channelNumbers, ref MccDaq.LoggerUnits units)
```

## Parameter

*channelNumbers*

An array that contains the analog input channel numbers logged in the file.

*units*

An array that contains the unit values set by the device in InstaCal for each analog input channel logged in the file.

The units settings are MccDaq.LoggerUnits enumerated constants. Choices are MccDaq.LoggerUnits.Temperature, and MccDaq.LoggerUnits.Raw.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- channelNumbers – Returns the analog input channel numbers logged in the binary file.
- units – Returns the unit values set by the device in InstaCal for each analog input channel logged in the binary file (MccDaq.LoggerUnits.Temperature or MccDaq.LoggerUnits.Raw.)

## GetCJCInfo() method

Returns the number of CJC temperature channels logged in a binary file.

Member of the [DataLogger](#) class.

### Function Prototype

VB .NET

```
Public Function GetCJCInfo(ByRef cjcCount As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetCJCInfo(ref int cjcCount)
```

### Parameter

*cjcCount*

The number of CJC temperature channels logged in the file.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- cjcCount – Returns the number of CJC temperature channels logged in the binary file.

## GetDIOInfo() method

Returns the number of digital I/O channels logged in a binary file.

Member of the [DataLogger](#) class.

## Function Prototype

VB .NET

```
Public Function GetDIOInfo(ByRef dioCount As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetDIOInfo(ref int dioCount)
```

## Parameter

*dioCount*

The number of digital I/O channels logged in the file.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dioCount – Returns the number of digital I/O channels logged in the file.

## GetFileInfo() method

Returns file information from the file associated with the current instance of the DataLogger.

Member of the [DataLogger](#) class.

## Function Prototype

VB .NET

```
Public Function GetFileInfo(ByRef version As Integer, ByRef size As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetFileInfo(ref int version, ref int size)
```

## Parameter

*version*

The version level of the file.

*size*

The size in bytes of the file.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- version – Returns the version level of the binary file.
- size – Returns the size in bytes of the binary file.



## GetFileName() method

Returns the name and path of the  $n^{\text{th}}$  file in the directory containing binary log files.

Member of the [DataLogger](#) class.

### Function Prototype

VB .NET

```
Public Shared Function GetFileName(ByVal fileNumber As Integer, ByRef path As String, ByRef fileName As String) As MccDag.ErrorInfo
```

C# .NET

```
public static MccDag.ErrorInfo GetFileName(int fileNumber, ref string path, ref string fileName)
```

### Parameter

*fileNumber*

Index of the file whose name you want to return. Specify one of the following:

- The number (n) that represents the location of the file in the directory (where n = 0, 1, 2, and so on), or
- MccService.GetFirst – get the first file in the directory, or
- MccService.GetNext – get the next file in the directory, based on the current index.

This parameter is the index of the file in the directory, and is not part of the filename.

*path*

The full path of the directory containing the log files.

*fileName*

The full path and name of the binary file. The path must be null-terminated and cannot be longer than 256 characters.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- fileName – Returns the file name and path of the binary file.

### Notes

- To access all of the files in a directory, first call GetFileName() with fileNumber set to MccService.GetFirst, then again with fileNumber set to MccService.GetNext until the method returns the error code [NOMOREFILES](#).

## GetPreferences() method

Returns API preference settings for time stamp data, analog temperature data, and CJC temperature data. Returns the default values unless changed using [SetPreferences\(\)](#).

Member of the [DataLogger](#) class.

## Function Prototype

VB .NET

```
Public Shared Function GetPreferences(ByRef timeFormat As MccDaq.TimeFormat, ByRef timeZone As MccDaq.TimeZone, ByRef units As MccDaq.TempScale) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetPreferences(ref MccDaq.TimeFormat timeFormat, ref MccDaq.TimeZone timeZone, ref MccDaq.TempScale units)
```

## Parameter

*timeFormat*

Returns the format used to display time stamp data.

All of the timeFormat settings are MccDaq.TimeFormat enumerated constants. Choices are MccDaq.TimeFormat.TwelveHour (for example 2:32:51PM) and MccDaq.TimeFormat.TwentyFourHour (for example 14:32:51).

*timeZone*

Returns the time zone to store time stamp data.

All of the timeZone settings are MccDaq.TimeZone enumerated constants. Choices are MccDaq.TimeZone.Local and MccDaq.TimeZone.GMT.

*units*

Returns the unit to use for analog temperature data. This value is ignored if raw data values are logged.

All of the units settings are MccDaq.TempScale enumerated constants. Choices are MccDaq.TempScale.Celsius, MccDaq.TempScale.Fahrenheit, and MccDaq.TempScale.Kelvin.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- timeFormat – Returns the format to apply to time stamp data from API functions that return time data.
- timeZone – Returns the time zone to apply to time stamp data from API functions that return time data.
- units – Returns the unit to use when converting temperature data from API functions that return temperature data.

## GetSampleInfo() method

Returns the sample interval, sample count, and the date and time of the first data point in a binary file.

Member of the [DataLogger](#) class.

## Function Prototype

VB .NET

```
Public Function GetSampleInfo(ByRef sampleInterval As Integer, ByRef sampleCount As Integer, ByRef  
startDate As Integer, ByRef startTime As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetSampleInfo(ref int sampleInterval, ref int sampleCount, ref int startDate,  
ref int startTime)
```

## Parameter

*sampleInterval*

The time, in seconds, between samples.

*sampleCount*

The number of samples contained in the file.

*startDate*

The date of the first data point logged in the file. Date values are packed in the following format:

Byte 0: day  
Byte 1: month  
Byte 2 - 3: year

*startTime*

The time when the first data point was logged in the file. Time values are packed in the following format:

Byte 0: seconds  
Byte 1: minutes  
Byte 2: hours  
Byte 3: 0xff = 24hour format, 0x0 = AM, 0x1 = PM

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- sampleInterval – Returns the time, in seconds, between samples.
- sampleCount – Returns the number of samples in the file.
- startDate – Returns the date of the first data point logged in the file.
- startTime – Returns the time when the first data point was logged in the file.

## Notes

- Time stamped data is returned according to the timeZone and timeFormat preferences. Refer to [SetPreferences\(\)](#) for more information.

# ReadAIChannels() method

Reads analog input data from a binary file, and stores the values in an array.

Member of the [DataLogger](#) class.

## Function Prototype

```
VB .NET
Public Function ReadAIChannels(ByVal startSample As Integer, ByVal count Integer, ByRef aiChannels As
Single) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo ReadAIChannels(int startSample, int count, ref float [] aiChannels)
```

## Parameter

- startSample*  
The first sample to read from the binary file.
- count*  
The number of samples to read from the binary file.
- aiChannels*  
Receives the analog input values.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- aiChannels – Returns the analog input values logged in the file.

## Notes

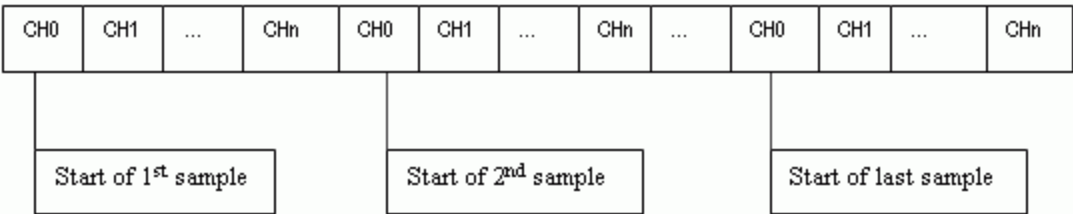
- The unit of the analog input data that is returned is set by the value of the Units preference. Refer to [SetPreferences\(\)](#).  
The units preference is only applied if the logged data is temperature data. This value is ignored if the data logged is raw.

## Analog array

The user is responsible for allocating the size of the analog data array, and ensuring that it is large enough to hold the data that will be returned. You can calculate the array allocation using the sampleCount value from [GetSampleInfo\(\)](#), and the aiCount value from [GetAIInfo\(\)](#):

```
float* aiChannels = new float[sampleCount * aiCount];
```

The figure below shows the layout of the analog array, and how the elements should be indexed.



Where:

- n* is (numberOfChannels – 1).
- CH0* – *CHn* refer to the channels in the array, not the input channels of the device.

For example, assume that all of the even number input channels are logged. The analog array channels are mapped as shown here:

Array Channel	Device Input Channel
0	0
1	2
2	4
3	6

Use the following code fragment to access the elements of the analog array:

```
for (i=0; i<numberOfSamples; i++)
{
    for (j=0; j<numberOfAIChannels; j++)
    {
        a = analogArray[(i * numberOfAIChannels) + j];
    }
}
```

where

numberOfSamples is set by the sampleCount value from [GetSampleInfo\(\)](#)

numberOfAIChannels is set by the aiCount value from [GetAIChannelCount\(\)](#)

# ReadCJCChannels() method

Reads CJC temperature data from a binary file, and stores the values in an array.

Member of the [DataLogger](#) class.

## Function Prototype

```
VB .NET
Public Function ReadCJCChannels (ByVal startSample As Integer, ByVal count Integer, ByRef cjcChannels As
Single) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo ReadCJCChannels (int startSample, int count, ref float [] cjcChannels)
```

## Parameter

- startSample*  
The first sample to read from the binary file.
- count*  
The number of samples to read from the binary file.
- cjcChannels*  
Receives the CJC temperature values.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- cjcChannels – Returns the CJC temperature values logged in the file.

## Notes

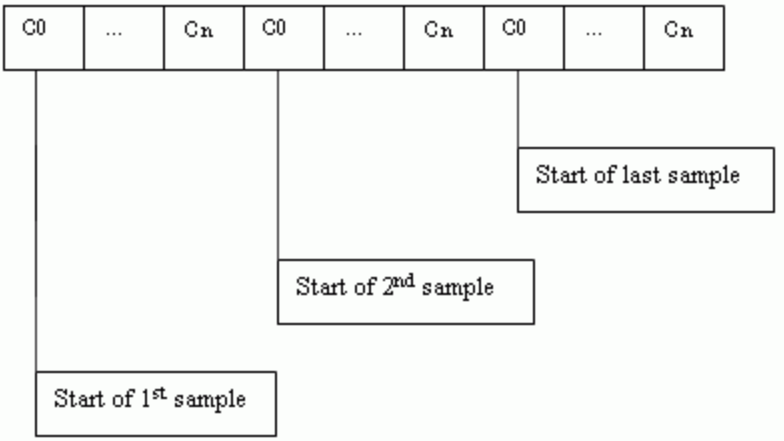
- The unit of the CJC temperature data that is returned is set by the value of the units preference. Refer to [SetPreferences\(\)](#).  
The units preference is only valid if the logged data is temperature data. This value is ignored if the data logged is raw.

## CJC array

The user is responsible for allocating the size of the CJC array, and ensuring that it is large enough to hold the data that will be returned. You can calculate the array allocation using the sampleCount value from [GetSampleInfo\(\)](#), and the cjcCount value from [GetCJCInfo\(\)](#):

```
float* cjcChannels = new float[sampleCount * cjcCount];
```

The figure below shows the layout of the CJC array, and how the elements should be indexed.



Where n is (CJCCount - 1)

Use the following code fragment to access the elements of the CJC array:

```
for (i=0; i<numberOfSamples; i++)
{
    for (j=0; j<numberOfCJCChannels; j++)
    {
        c = cjcArray[(i * numberOfCJCChannels) + j];
    }
}
```

where

numberOfSamples is set by the sampleCount value from [GetSampleInfo\(\)](#).

numberOfCJCChannels is set by the cjcCount value from [GetCJCInfo\(\)](#).

# ReadDIOChannels() method

Reads digital I/O channel data from a binary file, and stores the values in an array.

Member of the [DataLogger](#) class.

## Function Prototype

```
VB .NET
Public Function ReadDIOChannels(ByVal startSample As Integer, ByVal count Integer, ByRef dioChannels As
Single) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo ReadDIOChannels(int startSample, int count, ref float [] dioChannels)
```

## Parameter

- startSample*  
The first sample to read from the binary file.
- count*  
The number of samples to read from the binary file.
- dioChannels*  
Receives the DIO channel values.

## Returns

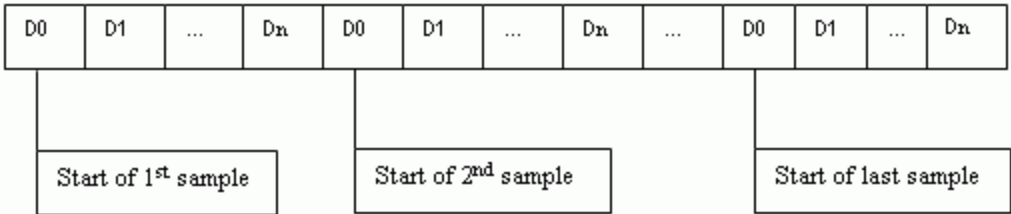
- An [ErrorInfo object](#) that indicates the status of the operation.
- dioChannels – Returns the DIO channel values logged in the file.

## DIO array

The user is responsible for allocating the size of the DIO array, and ensuring that it is large enough to hold the data that will be returned. You can calculate the array allocation using the sampleCount value from [GetSampleInfo\(\)](#), and the dioCount value from [GetDIOInfo\(\)](#).

```
float* dioChannels = new float[sampleCount * dioCount];
```

The figure below shows the layout of the DIO array, and how the elements should be indexed.



Where n is (dioCount - 1)

Use the following code fragment to access the elements of the DIO array:

```
for (i=0; i<numberOfSamples; i++)
{
    for (j=0; j<numberOfDIOChannels; j++)
    {
        d = dioArray[(i * numberOfDIOChannels) + j];
    }
}
```

where:

- numberOfSamples is set by the sampleCount value from [GetSampleInfo\(\)](#)
- numberOfDIOChannels is set by the dioCount value from [GetDIOInfo\(\)](#)



## ReadTimeTags() method

Reads the date and time values logged in a binary file. This method stores the date values in a dateTags array, and the time values in a timeTags array.

Member of the [DataLogger](#) class.

### Function Prototype

VB .NET

```
Public Function ReadTimeTags(ByVal startSample As Integer, ByVal count Integer, ByRef dateTags As Integer, ByRef timeTags As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo ReadTimeTags(int startSample, int count, ref int [] dateTags, ref int [] timeTags)
```

### Parameter

*startSample*

The first sample to read from the binary file.

*count*

The number of samples to read from the binary file.

*dateTags*

Receives the date tag values. Dates are packed in the following format:

Byte 0: day  
Byte 1: month  
Byte 2 - 3: year

*timeTags*

Receives the time tag values. Times are packed in the following format:

Byte 0: seconds  
Byte 1: minutes  
Byte 2: hours  
Byte 3: 0xff = 24hour format, 0x0 = AM, 0x1 = PM

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dateTags – Returns the date value for each sample logged in the file.
- timeTags – Returns the time value for each sample logged in the file.

### Notes

- Time stamped data is stored according to the timeZone preference and the timeFormat preference. Refer to [SetPreferences\(\)](#).
- Time stamped data is logged in the file if InstaCal is configured to do so. If time stamps are not logged, the time array is filled with values calculated from the file header information.

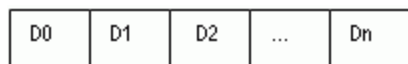
### Date and Time array size

The user is responsible for allocating the size of the date and time arrays, and ensuring that they are large enough to hold the data that is returned. You can calculate the array allocation using the sampleCount value from [GetSampleInfo\(\)](#).

```
int* dates = new int[sampleCount];  
int* times = new int[sampleCount];
```

### dateTags array

The figure below shows the layout of the dateTags array, and how the elements should be indexed.



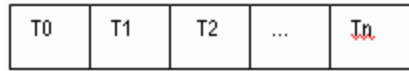
where n is (numberOfSamples – 1)

Each sample has only one date. Use the following code fragment to access the elements of the dateTags array:

```
for (i=0; i<numberOfSamples; i++)
{
    d = dateTagsArray[i];
}
```

#### **timeTags** array

The figure below shows the layout of the timeTags array, and how the elements should be indexed.



where n is (numberOfSamples - 1)

Each sample has only one time stamp. Use the following code fragment to access the elements of the timeTags array:

```
for (i=0; i<numberOfSamples; i++)
{
    t = timeTagsArray[i];
}
```

## SetPreferences() method

Sets preferences for returned time stamped data, analog temperature data, and CJC temperature data.

Member of the [DataLogger](#) class.

## Function Prototype

VB .NET

```
Public Shared Function SetPreferences(ByVal timeFormat As MccDaq.TimeFormat, ByVal timeZone As MccDaq.TimeZone, ByVal units As MccDaq.TempScale) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo SetPreferences(MccDaq.TimeFormat timeFormat, ref MccDaq.TimeZone timeZone, ref MccDaq.TempScale units)
```

## Parameter

*timeFormat*

Specifies the time format to apply when returning time stamp data (when using [ReadTimeTags\(\)](#) for example).

All of the timeFormat settings are MccDaq.TimeFormat enumerated constants. Choices are MccDaq.TimeFormat.TwelveHour (for example 2:32:51) and MccDaq.TimeFormat.TwentyFourHour (for example 14:32:51).

timeFormat defaults to MccDaq.TimeFormat.TwelveHour.

*timeZone*

Specifies whether to convert time stamped data that is returned (when using [ReadTimeTags\(\)](#) for example) to the local time zone or to return the time stamps as they are stored in the file (in the GMT time zone).

All of the timeZone settings are MccDaq.TimeZone enumerated constants. Choices are MccDaq.TimeZone.Local and MccDaq.TimeZone.GMT.

timeZone defaults to MccDaq.TimeZone.Local.

*units*

Specifies the unit for analog data. This value is ignored if counts are logged.

All of the units settings are MccDaq.TempScale enumerated constants. Choices are MccDaq.TempScale.Celsius, MccDaq.TempScale.Fahrenheit, and MccDaq.TempScale.Kelvin.

units defaults to MccDaq.TempScale.Fahrenheit.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- The timeFormat and timeZone preferences are applied to all time data returned using API methods that return time data.
- The units preference specifies the temperature scale that the API applies when reading and converting analog, CJC, and time stamped data.

## DBitIn() method

Reads the state of a single digital input bit.

This method treats all of the DIO ports of a particular type on a board as a single port. It lets you read the state of any individual bit within this port. Note that for some port types, such as 8255 ports, if the port is configured for DigitalOut, this method provides readback of the last output value.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O methods.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DBitIn(ByVal portType As MccDag.DigitalPortType, ByVal bitNum As Integer, ByRef  
bitValue As MccDag.DigitalLogicState) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DBitIn(MccDag.DigitalPortType portType, int bitNum, out  
MccDag.DigitalLogicState bitValue)
```

## Parameters

### [portType](#)

There are three general types of digital ports — ports that are programmable as input or output, ports that are fixed input or output, and ports for which each bit may be programmed as input or output. For the first of these types, set PortType to FirstPortA. For the latter two types, set portType to AuxPort. Some boards have both types of digital ports (DAS1600). Set portType to either FirstPortA or AuxPort depending on which digital port you wish to write to.

### *bitNum*

Specifies the bit number within the single large port.

### [bitValue](#)

Place holder for return value of bit. Value will be 0 or 1. A 0 indicates a logic low reading, a 1 indicates a logic high reading. Logic high does not necessarily mean 5 V. Refer to the board's user's guide for chip specifications.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- BitValue – value (0 or 1) of specified bit returned here.

## DBitOut() method

Sets the state of a single digital output bit.

This method treats all of the DIO chips of a particular type on a board as a single large port. It lets you set the state of any individual bit within this large port.

Most configurable ports require configuration before writing. Check the board-specific information in the *Universal Library User's Guide* to determine if the port should be configured for your hardware. When configurable, use [DConfigPort\(\)](#) to configure a port for output, and [DConfigBit\(\)](#) to configure a bit for output.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O methods.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DBitOut(ByVal portType As MccDaq.DigitalPortType, ByVal bitNum As Integer, ByVal  
bitValue As MccDaq.DigitalLogicState) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo DBitOut(MccDaq.DigitalPortType portType, int bitNum, MccDaq.DigitalLogicState  
bitValue)
```

## Parameters

### [portType](#)

There are three general types of digital ports — ports that are programmable as input or output, ports that are fixed input or output, and ports for which each bit may be programmed as input or output. For the first of these types, set portType to FirstPortA. For the latter two types, set portType to AuxPort.

For boards that have both types of digital ports (such as the DAS1600), set portType 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.

### [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 5 V. See the board manual for chip specifications.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## DConfigBit() method

Configures a specific digital bit as Input or Output. This method treats all DIO ports of the AuxPort type on a board as a single port. This method is NOT supported by 8255 type DIO ports. Refer to board-specific information for details.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DConfigBit(ByVal portNum As MccDag.DigitalPortType, ByVal bitNum As Integer, ByVal direction As MccDag.DigitalPortDirection) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DConfigBit(MccDag.DigitalPortType portNum, int bitNum, MccDag.DigitalPortDirection direction)
```

## Parameters

[portNum](#)

The port (AuxPort) whose bits are to be configured. The port specified must be bitwise configurable. See board specific information for details.

[bitNum](#)

The bit number to configure as input or output. See board specific information for details.

[direction](#)

MccDag.DigitalPortDirection.DigitalOut or MccDag.DigitalPortDirection.DigitalIn configures the specified bit for output or input, respectively.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## DConfigPort() method

Configures a digital port as input or output.

This method is for use with ports that may be programmed as input or output, such as those on the 82C55 chips and 8536 chips. Refer to the board's hardware *User Guide* for details of chip operation.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O methods.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DConfigPort(ByVal portNum As MccDaq.DigitalPortType, ByVal direction As MccDaq.DigitalPortDirection) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo DConfigPort(MccDaq.DigitalPortType portNum, MccDaq.DigitalPortDirection direction)
```

## Parameters

[portNum](#)

The specified port must be configurable. For most boards, AuxPort is not configurable; check the board-specific information in the *Universal Library User's Guide* for details.

[direction](#)

MccDaq.DigitalPortDirection.DigitalOut or MccDaq.DigitalPortDirection.DigitalIn configures the entire eight-bit or four-bit port for output or input, respectively.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Note

- When used on ports within an 8255 chip, this method will reset all ports on that chip configured for output to a zero state. This means that if you set an output value on FirstPortA and then change the configuration on FirstPortB from Output to Input, the output value at FirstPortA will be all zeros. You can, however, set the configuration on SecondPortx without affecting the value at FirstPortA. For this reason, this method is usually called at the beginning of the program for each port requiring configuration.

## DIn() method

Reads a digital input port.

Note that for some port types, such as 8255 ports, if the port is configured for DigitalOut, this method will provide readback of the last output value.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O methods.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DIn(ByVal portNum As MccDaq.DigitalPortType, ByRef dataValue As Short) As MccDaq.ErrorInfo
```

```
Public Function DIn(ByVal portNum As MccDaq.DigitalPortType, ByRef dataValue As System.UInt16) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo DIn(MccDaq.DigitalPortType portNum, out ushort dataValue)
```

```
public MccDaq.ErrorInfo DIn(MccDaq.DigitalPortType portNum, out short dataValue)
```

## Parameters

[portNum](#)

Specifies which digital I/O port to read. Some hardware does allow readback of the state of the output using this method. Check the board-specific information in the *Universal Library User's Guide* for details.

*dataValue*

Digital input value returned here.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataValue – Digital input value returned here

## Notes

- The size of the ports vary. If it is an eight bit port, the returned value is in the 0 – 255 range. If it is a four bit port, the value is in the 0 - 15 range.
- Refer to the board-specific information contained in the *Universal Library User's Guide* for clarification of valid portNum values.



## DInScan() method

Performs multiple reads of a digital input port of a high speed digital port on a board with a pacer clock, such as the CIO-PDMA16.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DInScan(ByVal portNum As MccDag.DigitalPortType, ByVal numPoints As Integer, ByRef rate As Integer, ByVal memHandle As IntPtr, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DInScan(MccDag.DigitalPortType portNum, int numPoints, ref int rate, IntPtr memHandle, MccDag.ScanOptions options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function DInScan(ByVal portNum As MccDag.DigitalPortType, ByVal numPoints As Integer, ByRef rate As Integer, ByVal memHandle As Integer, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DInScan(MccDag.DigitalPortType portNum, int numPoints, ref int rate, int memHandle, MccDag.ScanOptions options)
```

## Parameters

[portNum](#)

Specifies which digital I/O port to read (usually FirstPortA or FirstPortB). The specified port must be configured as an input.

[numPoints](#)

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 parameter.

[memHandle](#)

Handle for Windows buffer to store data. This buffer must have been previously allocated with the [WinBufAllocEx\(\)](#) or [WinBufAlloc32Ex\(\)](#) method.

[options](#)

Bit fields that control various options. Set it to one of the constants in the "[options parameter values](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- rate – actual sampling rate returned.
- memHandle – digital input value returned via allocated Windows buffer.

## options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ScanOptions enumeration (for example, variable = MccDaq.ScanOptions.Background, variable = MccDaq.ScanOptions.Continuous, etc.).

Background	<p>If the Background option is not used, the <a href="#">DInScan()</a> method will not return to your program until all of the requested data has been collected and returned to memHandle.</p> <p>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 memHandle will continue in the background. Use <a href="#">GetStatus()</a> with DiFunction to check on the status of the background operation. Use <a href="#">StopBackground()</a> with DiFunction to terminate the background process before it has completed.</p>
Continuous	<p>This option puts the method in an endless loop. Once it transfers the required number of bytes it resets to the start of the buffer and begins again. The only way to stop this operation is by calling <a href="#">StopBackground()</a> with DiFunction. Normally this option should be used in combination with Background so that your program will regain control.</p>
ExtClock	<p>If this option is used then transfers will be controlled by the signal on the trigger input line rather than by the internal pacer clock. Each transfer will be triggered on the appropriate edge of the trigger input signal (refer to board-specific information in the <i>Universal Library User's Guide</i>). When this option is used, the rate parameter is ignored. The transfer rate is dependent on the trigger signal.</p>
ExtTrigger	<p>If this option is used then the scan will not begin until the signal on the trigger input line meets the trigger criteria.</p>
HighResRate	<p>Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>rate</i> parameter above).</p>
WordXfer	<p>Normally this method reads a single (byte) port. If WordXfer is specified, 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 the buffer.</p> <p>When WordXfer is used, it is generally required to set portNum to <i>FirstPortA</i>.</p>

## Notes

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

## DOut() method

Writes a byte to a digital output port.

Most configurable ports require configuration before writing. Check the board-specific information in the *Universal Library User's Guide* to determine if the port should be configured for your hardware. When configurable, use [DConfigPort\(\)](#) to configure a port for output.

Refer to the "[Introduction: Digital Input / Output Boards](#)" topic for additional details on using digital I/O boards with the Universal Library's digital I/O methods.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DOut(ByVal portNum As MccDag.DigitalPortType, ByVal dataValue As Short) As MccDag.ErrorInfo

Public Function DOut(ByVal portNum As MccDag.DigitalPortType, ByVal dataValue As System.UInt16) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DOut(MccDag.DigitalPortType portNum, ushort dataValue)

public MccDag.ErrorInfo DOut(MccDag.DigitalPortType portNum, short dataValue)
```

## Parameters

[portNum](#)

There are three general types of digital ports - ports that are programmable as input or output, ports that are fixed input or output, and ports for which each bit may be programmed as input or output. For the first of these types, set portNum to FirstPortA. For the latter two types, set portNum to AuxPort. Some boards have both types of digital ports (DAS1600). Set portNum to either FirstPortA or AuxPort depending on which digital port you wish to write to.

[dataValue](#)

Digital value to be written.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- The size of the ports vary. If it is an eight bit port, the output value is in the 0 – 255 range. If it is a four bit port, the value is in the 0 - 15 range.
- Refer to the board-specific information in the *Universal Library User's Guide* for valid portNum values.

## DOutScan() method

Writes a series of bytes or words to the digital output port on a board with a pacer clock.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DOutScan(ByVal portNum As MccDag.DigitalPortType, ByVal count As Integer, ByRef rate As Integer, ByVal memHandle As IntPtr, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DOutScan(MccDag.DigitalPortType portNum, int count, ref int rate, IntPtr memHandle, MccDag.ScanOptions options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function DOutScan(ByVal portNum As MccDag.DigitalPortType, ByVal count As Integer, ByRef rate As Integer, ByVal memHandle As Integer, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DOutScan(MccDag.DigitalPortType portNum, int count, ref int rate, int memHandle, MccDag.ScanOptions options)
```

## Parameters

[portNum](#)

Specifies which digital I/O port to write (usually FirstPortA or FirstPortB). The specified port must be configured as an output.

[count](#)

The number of times to write digital output.

[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 parameter.

[memHandle](#)

Handle for Windows buffer to store data in (Windows). This buffer must have been previously allocated with the [WinBufAlloc\(\)](#) method.

[options](#)

Bit fields that control various options. Set it to one of the constants in the "[options parameter values](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- rate – actual sampling rate returned.

## options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ScanOptions enumeration (for example, variable = MccDaq.ScanOptions.Background, variable = MccDaq.ScanOptions.Continuous, etc.).

ADCClock	Paces the data output operation using the ADC clock.
ADCClockTrig	Triggers a data output operation when the ADC clock starts.
Background	<p>If the Background option is not used, the DOutScan() method will not return control to your program until all of the requested data has been output.</p> <p>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 memHandle will continue in the background. Use <a href="#">GetStatus()</a> with DoFunction to check on the status of the background operation. Use <a href="#">StopBackground()</a> with DoFunction to terminate the background process before it has completed.</p>
Continuous	This option puts the method in an endless loop. Once it transfers the required number of bytes it resets to the start of the buffer and begins again. The only way to stop this operation is by calling <a href="#">StopBackground()</a> with DoFunction. Normally this option should be used in combination with Background so that your program will regain control.
EXTCLOCK	<p>When this option is used, transfers are controlled by the signal on the external clock input rather than by the internal pacer clock. Each transfer will be triggered on the appropriate edge of the clock input signal (refer to board-specific information contained in the <i>UL Users Guide</i>).</p> <p>When this option is used, the rate parameter is used for reference only. The transfer rate is dependent on the clock signal. An approximation of the external clock rate is used to determine the size of the packets to transfer from the board. Set the rate parameter to an approximate maximum value.</p>
NonStreamedIO	<p>When this option is used, you can output non-streamed data to a specific DAC output channel.</p> <p>To load the data output buffer into the device's internal output FIFO, the aggregate size of the data output buffer must be less than or equal to the size of the internal data output FIFO in the device. Once the sample data are transferred or downloaded to the device, the device is responsible for outputting the data. You can't make any changes to the output buffer once the output begins.</p> <p>With NonStreamedIO mode, you do not have to periodically feed output data through the program to the device for the data output to continue. However, the size of the buffer is limited.</p>
WordXfer	<p>Normally this method writes a single (byte) port. If WordXfer is specified, it will write two adjacent ports as the low and high byte of a single array element in dataBuffer.</p> <p>When WordXfer is used, it is generally required to set portNum to FirstPortA.</p>

## Notes

- MccDaq.ScanOptions.ByteXfer is the default option. Make sure you are using an array when your data is arranged in bytes. Use the MccDaq.ScanOptions.WordXfer option for word array transfers.
- NonStreamedIO can only be used with the number of samples (count) set equal to the size of the FIFO or less.
- Transfer Method may not be specified. DMA is used.

# ErrHandling() method

Sets the error handling for all subsequent method calls. Most methods return error codes after each call. In addition, other error handling features are built into the library. This method controls those features. If the Universal Library cannot find the configuration file CB.CFG, it always terminates the program, regardless of the ErrHandling() setting.

Member of the [MccService class](#).

## Function Prototype

```
VB .NET
Public Shared Function ErrHandling(ByVal errorReporting As MccDaq.ErrorReporting, ByVal errorHandling
As MccDaq.ErrorHandling) As MccDaq.ErrorInfo

C# .NET
public static MccDaq.ErrorInfo ErrHandling(MccDaq.ErrorReporting errorReporting, MccDaq.ErrorHandling
errorHandling)
```

## Parameters

### errorReporting

This parameter controls when the library will print error messages on the screen. The default is DontPrint. Set it to one of the constants in the "[errorReporting parameter values](#)" section below.

### errorHandling

This parameter specifies what class of error will cause the program to halt. The default is DontStop. Set it to one of the constants in the "[errorHandling parameter values](#)" section below.

## Returns

- Returns an [ErrorInfo object](#) that always has [ErrorInfo.Value](#) = NoErrors.

## errorReporting parameter values

All of the errorReporting settings are MccDaq.ErrorReporting enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ErrorReporting enumeration (for example, variable = MccDaq.ErrorReporting.DontPrint, variable = MccDaq.ErrorReporting.PrintWarnings, etc.).

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.

## errorHandling parameter values

All of the errorReporting settings are MccDaq.ErrorHandling enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ErrorHandling enumeration (for example, variable = MccDaq.ErrorHandling.DontStop, variable = MccDaq.ErrorHandling.StopFatal, etc.).

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. If you are running in an Integrated Development Environment (IDE) then when errors occur, the environment may be shut down along with the program. If your IDE behaves this way, then you should set ErrHandling() to DONTSTOP. You can check error codes to determine the cause of the error.

## Note

### 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."

## LogToFile() property

Set the `ErrorInfo.LogToFile` property to *true* to record time-stamped error codes to a file. Most UL for .NET method returns either an `ErrorInfo` object or 0. If an error occurs, an `ErrorInfo` object is returned.

Member of the [ErrorInfo class](#).

## Property prototype

VB .NET

```
Public Property LogToFile As Boolean
```

C# .NET

```
public bool LogToFile {get, set}
```

## Notes

- Refer to the [ErrHandling\(\)](#) method for an alternate method of handling errors.

## Message() property

Returns the error message associated with an `ErrorInfo` object.

Most UL for .NET methods return an `ErrorInfo` object. If no error occurred, an `ErrorInfo` object is returned with the `Message` property set to "No error has occurred".

Member of the [ErrorInfo class](#).

## Property prototype

VB .NET

```
Public ReadOnly Property Message As String
```

C# .NET

```
public string Message {get}
```

## Notes

- Refer to the [ErrHandling\(\)](#) method for an alternate method of handling errors.



## Value() property

Use the `ErrorInfo.Value` property to get the error constant associated with an `ErrorInfo` object. Most UL for .NET methods return an `ErrorInfo` object. If an error occurs, an `ErrorInfo` object is returned with a non-zero value in the `Value` property.

Member of the [ErrorInfo class](#).

## Property prototype

VB .NET

```
Public ReadOnly Property Value As MccDaq.ErrorInfo.ErrorCode
```

C# .NET

```
public MccDaq.ErrorInfo.ErrorCode Value [get]
```

## Notes

- Refer to the [ErrHandling\(\)](#) method for an alternate method of handling errors.

## MemRead() method

Reads data from a memory board into an array.

Member of the [MccBoard](#) class.

## Function Prototype

### VB .NET

```
Public Function MemRead(ByVal dataBuffer As Short(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo

Public Function MemRead(ByVal dataBuffer As System.UInt16(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo MemRead(short[] dataBuffer, int firstPoint, int numPoints)

public MccDaq.ErrorInfo MemRead(ushort[] dataBuffer, int firstPoint, int numPoints)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

### VB .NET

```
Public Function MemRead(ByRef dataBuffer As Short, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo

Public Function MemRead(ByRef dataBuffer As System.UInt16, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo MemRead(out short dataBuffer, int firstPoint, int numPoints)

public MccDaq.ErrorInfo MemRead(out ushort dataBuffer, int firstPoint, int numPoints)
```

## Parameter

### *dataBuffer*

Reference to the data array.

### *firstPoint*

Index of first point to read, or FromHere. Use the firstPoint parameter to specify the first point to be read. For example, to read data sample numbers 200 through 250, set firstPoint = 200 and numPoints = 50.

### *numPoints*

Number of data points (words) to read.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataBuffer - data read from the memory board.

## Notes

- If you are going to read a large amount of data from the board in small chunks, set firstPoint to FromHere to read each successive chunk. Using FromHere speeds up the operation of MemRead() 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:

```
DaqBoard0.MemRead(dataBuffer, 0, 100000)
DaqBoard0.MemRead(dataBuffer, FROMHERE, 1000000)
DaqBoard0.MemRead(dataBuffer, FROMHERE, 1000000)
```

- **DT-Connect conflicts:** The MemRead() method cannot 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 [AInScan\(\)](#) with the DTConnect + Background options) you cannot call MemRead() until the AInScan() has completed. If you do you will get a [DTACTIVE](#) error.

## MemReadPretrig() method

Reads pre-trigger data from a memory board that has been collected with the [APretrig\(\)](#) method and re-arranges the data in the correct order (pre-trigger data first, then post-trigger data). This method can only be used to retrieve data that has been collected with the [APretrig\(\)](#) method with ExtMemory set in the options parameter. After each [APretrig\(\)](#) call, all data must be unloaded from the memory board with this method. If any more data is sent to the memory board then the pre-trigger data will be lost.

Member of the [MccBoard](#) class.

### Function Prototype

#### VB .NET

```
Public Function MemReadPretrig(ByVal dataBuffer As Short(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDag.ErrorInfo

Public Function MemReadPretrig(ByVal dataBuffer As System.UInt16(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDag.ErrorInfo
```

#### C# .NET

```
public MccDag.ErrorInfo MemReadPretrig(short[] dataBuffer, int firstPoint, int numPoints)

public MccDag.ErrorInfo MemReadPretrig(ushort[] dataBuffer, int firstPoint, int numPoints)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

#### VB .NET

```
Public Function MemReadPretrig(ByRef dataBuffer As Short, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDag.ErrorInfo

Public Function MemReadPretrig(ByRef dataBuffer As System.UInt16, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDag.ErrorInfo
```

#### C# .NET

```
public MccDag.ErrorInfo MemReadPretrig(out short dataBuffer, int firstPoint, int numPoints)

public MccDag.ErrorInfo MemReadPretrig(out ushort dataBuffer, int firstPoint, int numPoints)
```

### Parameters

#### *dataBuffer*

Reference to the data array.

#### *firstPoint*

Index of first point to read or FromHere. Use the FirstPoint parameter to specify the first point to be read. For example, to read data sample numbers 200 through 250, set firstPoint = 200 and numPoints = 50.

#### *numPoints*

Number of data samples (words) to read.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataBuffer - data read from memory board

### Notes

- If you are going to read a large amount of data from the board in small chunks, set FirstPoint to FromHere to read each successive chunk. Using FromHere speeds up the operation of [MemReadPretrig\(\)](#) when working with large amounts of data.

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

```
DaqBoard0.MemReadPretrig(dataBuffer, 0, 100000)
DaqBoard0.MemReadPretrig(dataBuffer, FROMHERE, 1000000)
DaqBoard0.MemReadPretrig(dataBuffer, FROMHERE, 1000000)
```

- **DT-Connect conflicts:** The [MemReadPretrig\(\)](#) method cannot 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 [AInScan\(\)](#) with the DTConnect + Background options) you cannot call [MemReadPretrig\(\)](#) until the [AInScan\(\)](#) has completed. If you do you will get a [DTACTIVE](#) error.

## MemReset() method

Resets the memory board reference 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.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function MemReset\(\) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo MemReset\(\)
```

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

This method is used to reset the counter back to the start of the memory. Between successive calls to [AInScan\(\)](#), you should call this method so that the second AInScan() overwrites the data from the first call. Otherwise, the data from the first AInScan() will be followed by the data from the second AInScan() in the memory on the card.

Likewise, anytime you call [MemRead\(\)](#) or [MemWrite\(\)](#), it will leave the counter pointing to the next memory location after the data that you read or wrote. Call [MemReset\(\)](#) to reset back to the start of the memory buffer before the next call to AInScan().

## MemSetDTMode() method

Sets the DT-Connect Mode of a memory board.

Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function MemSetDTMode (ByVal mode As MccDag.DTMode) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo MemSetDTMode (MccDag.DTMode mode)
```

### Parameters

[mode](#)

Must be set to either DTIn or DTOut. Set the mode on the memory board to DTIn to transfer data from an A/D board to the memory board. Set mode = DTOut to transfer data from a memory board to a D/A board.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

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

If using the ExtMemory option for [AInScan\(\)](#), etc., *this method should not be used*. The Memory Board mode is already set through the ExtMemory option.

- *Use this method only if the parent board is not supported by the Universal Library.*

## MemWrite() method

Writes data from an array to the memory card.

Member of the [MccBoard](#) class.

## Function Prototype

### VB .NET

```
Public Function MemWrite(ByVal dataBuffer As Short(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo

Public Function MemWrite(ByVal dataBuffer As System.UInt16(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo MemWrite(short[] dataBuffer, int firstPoint,int numPoints)

public MccDaq.ErrorInfo MemWrite(ushort[] dataBuffer, int firstPoint, int numPoints)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

### VB .NET

```
Public Function MemWrite(ByRef dataBuffer As Short, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo

Public Function MemWrite(ByRef dataBuffer As System.UInt16, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo MemWrite(ref short dataBuffer, int firstPoint, int numPoints)

public MccDaq.ErrorInfo MemWrite(ref ushort dataBuffer, int firstPoint, int numPoints)
```

## Parameters

### *dataBuffer*

Reference to the data array.

### *firstPoint*

Index of first point to write or FromHere. Use the firstPoint parameter to specify where in the board's memory to write the first point. For example, to write to location numbers 200 through 250, set firstPoint = 200 and numPoints = 50.

### *numPoints*

Number of data points (words) to write.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- To write large amounts of data to the board in small chunks, set firstPoint to FromHere to write each successive chunk. Using FromHere speeds up the operation of MemWrite() 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:

```
DaqBoard1.MemWrite(dataBuffer, 0, 100000)
DaqBoard1.MemWrite(dataBuffer, FromHere, 100000)
DaqBoard1.MemWrite(dataBuffer, FromHere, 100000)
```

- **DT-Connect conflicts:** The MemWrite() method cannot 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 [AInScan\(\)](#) with the DTCONNECT + BACKGROUND options). You cannot call MemWrite() until the AInScan() has completed. If you do, you will get a [DTACTIVE](#) error.

## DeclareRevision() method

Initializes the Universal Library with the revision number of the library used to write the program. This method must be the first Universal Library method to be called by the program.

Member of the [MccService class](#).

## Function Prototype

VB .NET

```
Public Shared Function DeclareRevision(ByRef revNum As Single) As MccDag.ErrorInfo
```

C# .NET

```
public static MccDag.ErrorInfo DeclareRevision(ref float revNum)
```

## Parameters

*revNum*

Revision number of the Universal Library used to interpret method parameters.

*default*

Any program using the 32-bit library and *not* containing this line of code will be defaulted to revision 5.4 parameter assignments.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- By default, any program using the 32-bit library and *not* containing this line of code will be defaulted to revision 5.4 parameter assignments.
- As new revisions of the library are released, bugs from previous revisions are fixed and occasionally new properties and methods are added. It is Measurement Computing's goal to preserve existing programs you have written and therefore to never change the order or number of parameters in a method.
- With the DeclareRevision() method, programs do not have to be rewritten in each line where new functions are used, and the program then recompiled. The revision control method initializes the DLL so that the functions are interpreted according to the format of the revision that you wrote and compiled your program in. The method works by interpreting the UL function call from your program and filling in any arguments needed to run with the new revision.
- If your program has declared you are running code written for an earlier revision and you call a new method, you must rewrite your program to include the new parameter, and declare the current revision in the DeclareRevision() method call.

## GetRevision() method

Gets the revision number of the Universal Library DLL and VXD.

Member of the [MccService class](#).

## Function Prototype

VB .NET

```
Public Shared Function GetRevision(ByRef revNum As Single, ByRef vxdRevNum As Single) As MccDag.ErrorInfo
```

C# .NET

```
public static MccDag.ErrorInfo GetRevision(out float revNum, out float vxdRevNum)
```

## Parameters

*revNum*

Place holder for the revision number of Library DLL.

*VXDRevNum*

Place holder for the revision number of Library VXD.

## Returns

- An [ErrorInfo object](#) that indicates if the revision levels of VXD and DLL are incompatible.
- revNum – Revision number of the Universal Library DLL
- vxdRevNum – Revision number of the Universal Library VXD



## Version property

This information is used by the library to determine compatibility.

Member of the [GlobalConfig class](#).

## Property prototype

VB .NET

```
Public Shared ReadOnly Property Version As Integer
```

C# .NET

```
public int Version {get}
```

## FileAInScan() method

Scans a range of A/D channels and stores the samples in a disk file. FileAInScan() reads the specified number of A/D samples at the specified sampling rate from the specified range of A/D channels from the board. If the A/D board has programmable gain, it sets the gain to the specified range.

The collected data is returned to a file in binary format. Use [FileRead\(\)](#) to load data from that file into an array. Refer to board-specific information to determine if this method is supported on your device.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function FileAInScan(ByVal lowChan As Integer, ByVal highChan As Integer, ByVal numPoints As Integer, ByRef rate As Integer, ByVal range As MccDag.Range, ByVal fileName As String, ByVal options As MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo FileAInScan(int lowChan, int highChan, int numPoints, ref int rate, MccDag.Range range, string fileName, MccDag.ScanOptions options)
```

## Parameters

*lowChan*

First A/D channel of the scan.

*highChan*

Last A/D channel of the scan.

The maximum allowable channel depends on which type of A/D board is being used. For boards with both single ended and differential inputs, the maximum allowable channel number also depends on how the board is configured (for example, eight channels for differential, 16 for single ended).

*numPoints*

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

*rate*

Sample rate in samples per second (Hz) per channel. The maximum sampling rate depends on the A/D board that is being used (refer to the rate description in [AInScan\(\)](#)).

*range*

If the selected A/D board does not have a programmable range feature, this parameter is ignored. Otherwise set the range parameter to any range that is supported by the selected A/D board. Refer to board-specific information in the *Universal Library User's Guide* for a list of the [supported A/D ranges](#) of each board.

*fileName*

The name of the file in which to store the data. If the file doesn't exist, it will be created.

*options*

Bit fields that control various options. Set it to one of the constants in the [options parameter values](#) section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- rate – actual sampling rate.

## options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, refer to the MccDaq object and the ScanOptions enumeration (for example, variable = MccDaq.ScanOptions.ExtClock, variable = MccDaq.ScanOptions.ExtTrigger, and so on).

ExtClock	If this option is used, conversions are controlled by the signal on the external clock input rather than by the internal pacer clock. Each conversion is triggered on the appropriate edge of the trigger input signal (see board specific info). Additionally, the Rate parameter is ignored. The sampling rate is dependent on the trigger signal.
ExtTrigger	<p>If this option is specified, the sampling does not begin until the trigger condition is met.</p> <p>On many boards, this trigger condition is programmable (refer to the <a href="#">SetTrigger()</a> method and board-specific info for details) and can be programmed for rising or falling edge or an analog level.</p> <p>On other boards, only <i>polled gate</i> triggering is supported. Assuming active high operation, data acquisition commences immediately if the trigger input is high. If the trigger input is low, acquisition is held off until it goes high. Acquisition continues until numPoints samples are taken, regardless of the state of the trigger input. For <i>polled gate</i> triggering, this option is most useful if the signal is a pulse with a very low duty cycle (trigger signal is in a TTL low state most of the time) to hold off triggering until the pulse occurs.</p>
DtConnect	Samples are sent to the DT-Connect port if the board is equipped with one.

## Notes

- [OverRun error](#) - (Error code 29) 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 returned from [FileGetInfo\(\)](#) in totalCount is the number of points that were successfully collected.

## Important!

In order to understand the methods, read the board-specific information in the *Universal Library User's Guide* and also in the ReadMe files installed with the Universal Library. We also urge you to examine and run one or more of the example programs supplied prior to attempting any programming of your own. Following this advice may save you hours of frustration, and wasted time.

This note, which appears elsewhere, is especially applicable to this method. Now is the time to read the board-specific information for your board. We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.

## FileGetInfo() method

Returns information about a streamer file.

When [FileAInScan\(\)](#) or [FilePretrig\(\)](#) fills the streamer file, information is stored about how the data was collected (sample rate, channels sampled etc.). This method returns that information. Refer to board-specific information in the *Universal Library User's Guide* to determine if your device supports [FileAInScan\(\)](#) and/or [FilePretrig\(\)](#).

Member of the [MccService class](#).

## Function Prototype

VB .NET

```
Public Shared Function FileGetInfo(ByVal fileName As String, ByRef lowChan As Short, ByRef highChan As Short, ByRef pretrigCount As Integer, ByRef totalCount As Integer, ByRef rate As Integer, ByRef range As MccDaq.Range) As MccDaq.ErrorInfo
```

C# .NET

```
public static MccDaq.ErrorInfo FileGetInfo(string fileName, out short lowChan, out short highChan, out int pretrigCount, out int totalCount, out int rate, out MccDaq.Range range)
```

## Parameters

*fileName*

Name of streamer file.

*lowChan*

Variable to return lowChan to.

*highChan*

Variable to return highChan to.

*pretrigCount*

Variable to return pretrigCount to.

*totalCount*

Variable to return totalCount to.

*rate*

Variable to return sampling rate to.

*range*

Variable to return the A/D range code to. Refer to board-specific information in the *Universal Library User's Guide* for a list of the [supported A/D ranges](#) of each device.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- lowChan – low A/D channel of the scan.
- highChan – high A/D channel of the scan.
- totalCount – total number of points collected.
- pretrigCount – number of pre-trigger points collected.
- rate – sampling rate when data was collected.
- range – Range of the A/D when data was collected.

## FilePretrig() method

Scan a range of channels continuously while waiting for a trigger.

Once the trigger occurs, FilePretrig() returns the specified number of samples, including the specified number of pre-trigger samples to a disk file. This method 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. See board specific info to determine if this method is supported by your board.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function FilePretrig(ByVal lowChan As Integer, ByVal highChan As Integer, ByRef pretrigCount As Integer, ByRef totalCount As Integer, ByRef rate As Integer, ByVal range As MccDaq.Range, ByVal fileName As String, ByVal options As MccDaq.ScanOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo FilePretrig(int lowChan, int highChan, ref int pretrigCount, ref int totalCount, ref int rate, MccDaq.Range range, string fileName, MccDaq.ScanOptions options)
```

## Parameters

*lowChan*

First A/D channel of the scan.

*highChan*

Last A/D channel of the scan.

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 Refer to board-specific information for the maximum number of channels allowed in differential and single ended modes.

*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 than 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*

Sets the total number of samples to be collected and stored in the file. totalCount must be greater than or equal to pretrigCount + 512.

If the trigger occurs too early, fewer than the requested number of samples will be collected and a [TooFew](#) error will occur. The totalCount will be set to indicate how many samples were actually collected.

*rate*

Sample rate in samples per second (Hz) per channel. The maximum sampling rate depends on the A/D board that is being used. 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 (10 kHz) will result in the A/D converter rate of 40 kHz: 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. This parameter also returns the value of the actual set. This may be different from the requested rate because of pacer limitations.

*range*

If the selected A/D board does not have a programmable range feature, this parameter is ignored. Otherwise, set the Range parameter to any range that is supported by the selected A/D board. Refer to board-specific information for a list of the [supported A/D ranges](#) of each device.

*fileName*

The name of the file in which to store the data. If the file doesn't exist, it will be created.

*options*

Bit fields that control various options. Set it to one of the constants in the [options parameter values](#) section below.

Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- preTrigCount – actual number of pre-trigger samples collected
- totalCount – actual number of samples collected
- rate – the actual sampling rate

options parameter values

All of the options settings are MccDaq.ScanOptions enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ScanOptions enumeration (for example, variable = MccDaq.ScanOptions.ExtClock or variable = MccDaq.ScanOptions.DtConnect).

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 appropriate edge of the trigger input signal (see board specific info). When this option is used the Rate parameter 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.

Notes

- [OverRun error](#) - (Error code 29) 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 TotalCount will be the number of points that were successfully collected.

## FileRead() method

This method reads data from a streamer file, and returns the data in a one-dimensional or two-dimensional array.

When [FileAInScan\(\)](#) or [FilePreTrig\(\)](#) fills the streamer file, this method returns the content of that file. Refer to information on your board in the *Universal Library User's Guide* to determine if your board supports [FileAInScan\(\)](#) and/or [FilePreTrig\(\)](#).

Member of the [MccService class](#).

## Function Prototype

### VB .NET

```
Public Shared Function FileRead(ByVal fileName As String, ByVal firstPoint As Integer, ByRef numPoints As Integer, ByVal dataBuffer As Short()) As MccDaq.ErrorInfo

Public Shared Function FileRead(ByVal fileName As String, ByVal firstPoint As Integer, ByRef numPoints As Integer, ByVal dataBuffer As System.UInt16()) As MccDaq.ErrorInfo

Public Shared Function FileRead(ByVal fileName As String, ByVal firstPoint As Integer, ByRef numPoints As Integer, ByVal dataBuffer As System.double(,)) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo FileRead(string fileName, int firstPoint, ref int numPoints, short[] dataBuffer)

public MccDaq.ErrorInfo FileRead(string fileName, int firstPoint, ref int numPoints, ushort[] dataBuffer)

public MccDaq.ErrorInfo FileRead(string fileName, int firstPoint, ref int numPoints, double[, ] dataBuffer)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

### VB .NET

Returns a one-dimensional array of short values:

```
Public Shared Function FileRead(ByVal fileName As String, ByVal firstPoint As Integer, ByRef numPoints As Integer, ByRef dataBuffer As Short) As MccDaq.ErrorInfo
```

Returns a one-dimensional array of System.UInt16 values:

```
Public Shared Function FileRead(ByVal fileName As String, ByVal firstPoint As Integer, ByRef numPoints As Integer, ByRef dataBuffer As System.UInt16) As MccDaq.ErrorInfo
```

Returns a two-dimensional array of double values:

```
Public Shared Function FileRead(ByVal fileName As String, ByVal firstPoint As Integer, ByRef numPoints As Integer, ByRef dataBuffer As Double(,), ByVal numChannels As Integer) As MccDaq.ErrorInfo
```

### C# .NET

Returns a one-dimensional array of short values:

```
public static MccDaq.ErrorInfo FileRead(string fileName, int firstPoint, ref int numPoints, out short dataBuffer)
```

Returns a one-dimensional array of System.UInt16 values:

```
public static MccDaq.ErrorInfo FileRead(string fileName, int firstPoint, ref int numPoints, out ushort dataBuffer)
```

Returns a two-dimensional array of double values:

```
public static MccDaq.ErrorInfo FileRead(string fileName, int firstPoint, ref int numPoints, out double dataBuffer[, ], int numChannels)
```

## Parameters

### *fileName*

The name of the streamer file to read.

### *firstPoint*

The index of the first point to read.

*numPoints*

The number of points to read in the file.

*dataBuffer*

A reference to the array in the data buffer that the data is read into.

*numChannels*

The number of channels to read into dataBuffer.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataBuffer – the data read from a file.
- numPoints – number of points actually read. numPoints may be less than the requested number of points if an error occurs.

## Notes

- Data format:

The data is returned as 16-bits. The 16-bits may represent 12 bits of analog, 12-bits of analog plus 4 bits of channel, or 16-bits of analog data.

- Loading portions of files:

The file may contain more data than can fit in dataBuffer. Use numPoints and firstPoint to read a selected piece of the file into dataBuffer. Call [FileGetInfo\(\)](#) first to find out how many points are in the file.



## DaqInScan() method

Scans analog, digital, counter, and temperature input channels synchronously, and stores the samples in an array. This method only works with boards that support synchronous input.

Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function DaqInScan(ByVal chanArray As Short(), ByVal chanTypeArray As MccDag.ChannelType, ByVal gainArray as MccDag.Range, ByVal chanCount As Integer, ByRef rate As Integer, ByRef pretrigCount As Integer, ByRef totalCount As Integer, ByVal memHandle As IntPtr, ByVal options MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DaqInScan(short[] chanArray, MccDag.ChannelType[] chanTypeArray, MccDag.Range[] gainArray, int chanCount, ref int rate, ref int pretrigCount, ref int totalCount, IntPtr memHandle, MccDag.ScanOptions options)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function DaqInScan(ByVal chanArray As Short( ), ByVal chanTypeArray As MccDag.ChannelType, ByVal gainArray as MccDag.Range, ByVal chanCount As Integer, ByRef rate As Integer, ByRef pretrigCount As Integer, ByRef totalCount As Integer, ByVal memHandle As Integer, ByVal options MccDag.ScanOptions) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DaqInScan(short[] chanArray, MccDag.ChannelType[] chanTypeArray, MccDag.Range[] gainArray, int chanCount, ref int rate, ref int pretrigCount, ref int totalCount, int memHandle, MccDag.ScanOptions options)
```

### Parameters

*chanArray*

Array containing channel values. Valid channel values are analog input channels, digital ports, counter input channels, and temperature input channels on the device.

*chanTypeArray*

Array containing channel types. Each element of this array defines the type of the corresponding element in the chanArray.

All of the chanTypeArray settings are MccDag.ChannelType enumerated constants. Set it to one of the constants in the [chanTypeArray parameter values](#) section below.

*gainArray*

Array containing A/D range codes. If the corresponding element in the chanArray is not an analog input channel, the range code for this channel is ignored.

All of the gainArray settings are [MccDag.Range](#) enumerated constants. Set to any range that is supported by the selected A/D board. Refer to the board-specific information in the *Universal Library User's Guide* for a list of the supported A/D ranges of each device.

*chanCount*

Number of elements in each of the three arrays - chanArray, chanTypeArray, and gainArray.

*rate*

The sample rate at which samples are acquired, in samples per second per channel. rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*pretrigCount*

Sets the number of pre-trigger samples to collect. Specifies the number of samples to collect before the trigger occurs. This method won't run in pre-trigger mode if preTrigCount is set to zero. preTrigCount is ignored if the ExtTrigger option is not specified.

pretrigCount also returns the value of the actual pre-trigger count set, which may be different from the set pre-trigger count because pre-trigger count must be a multiple of the channel count (chanCount).

pretrigCount must be evenly divisible by the number of channels being scanned (chanCount). If it is not, this method adjusts the number (down) to the next valid value, and returns that value to the pretrigCount parameter.

### totalCount

Total number of samples to collect. Specifies the total number of samples to collect and store in the buffer. totalCount must be greater than preTrigCount.

totalCount also returns the value of the actual total count set, which may be different from the requested total count, because total count must be a multiple of the channel count (chanCount).

totalCount must be evenly divisible by the number of channels being scanned (chanCount). If it is not, this method adjusts the number (down) to the next valid value, and returns that value to the totalCount parameter.

### memHandle

Handle for the Windows buffer to store data. This buffer must have been previously allocated with the [WinBufAllocEx\(\)](#), [WinBufAlloc32Ex\(\)](#), or [WinBufAlloc64Ex\(\)](#) method.

### options

Bit fields that control various options. All of the options settings are [MccDaq.ScanOptions](#) enumerated constants. This field may contain any combination of non-contradictory choices in the [options parameter values](#) section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- rate – Actual sampling rate used.
- preTrigCount – Actual pre-trigger count used.
- totalCount – Actual total count used.
- memHandle - Collected data returned via the Windows buffer.

### chanTypeArray parameter values

Analog	Analog input channel.
Digital8	8-bit digital input port.
Digital16	16-bit digital input port. (FirstPortA only)
Ctr16	16-bit counter.
Ctr32Low	Lower 16-bits of a 32-bit counter.
Ctr32High	Upper 16-bits of a 32-bit counter.
CJC	CJC channel.
TC	Thermocouple channel.  The <a href="#">GetTCValues()</a> method can be used to convert raw thermocouple data to data on a temperature scale (MccDaq.TempScale.Celsius, MccDaq.TempScale.Fahrenheit or MccDaq.TempScale.Kelvin).  <b>Note:</b> If at least one TC channel is listed in the channel array, and averaging is enabled for that channel, the averaging will be applied to all of the channels listed in the channel array.
SetpointStatus	The setpoint status register. This is a bit field indicating the state of each of the setpoints. A "1" indicates that the setpoint criteria has been met.

### chanTypeArray flag values

SetpointEnable	Enables a setpoint. When this option is specified, it must be OR'ed with the ChanTypeArray parameter values.  You set the setpoint criteria with the <a href="#">DaqSetSetpoints()</a> method. The number of channels set with the SetpointEnable flag must match the number of setpoints set by the <a href="#">DaqSetSetpoints()</a> method's setpointCount parameter.
----------------	--

### options parameter values

Background	When the Background option is used, control returns immediately to the next line in your program, and the data collection from the counters into the buffer continues in the background. If the Background option is not used, the <a href="#">DaqInScan()</a> method does not return to your program until all of the requested data has been collected and returned to the buffer.  Use <a href="#">GetStatus()</a> with <a href="#">DaqiFunction</a> to check on the status of the background operation. Use <a href="#">StopBackground()</a> with <a href="#">DaqiFunction</a> to terminate the background process before it has completed. Execute <a href="#">StopBackground()</a> after normal termination of all background functions in order to clear variables and flags.
Continuous	This option puts the function in an endless loop. Once it collects the required number of samples, it resets to the start of the buffer and begins again. The only way to stop this operation is to use <a href="#">StopBackground()</a> with <a href="#">DaqiFunction</a> . Normally, this option should be used in combination with Background so that your program will regain control.

ExtClock	If this option is used, conversions will be controlled by the signal on the external clock input rather than by the internal pacer clock. Each conversion will be triggered on the appropriate edge of the clock input signal. When this option is used the rate argument is ignored. The sampling rate is dependent on the clock signal. Options for the board will default to a transfer mode that will allow the maximum conversion rate to be attained unless otherwise specified.
ExtTrigger	If this option is specified, the sampling will not begin until the trigger condition is met (refer to the <a href="#">DagSetTrigger()</a> method).
HighResRate	Acquires data at a high resolution rate. When specified, the rate at which samples are acquired is in "samples per 1000 seconds per channel". When this option is not specified, the rate at which samples are acquired is in "samples per second per channel" (refer to the <i>rate</i> parameter above).

## DaqOutScan() method

Outputs values synchronously to analog output channels and digital output ports. This function only works with boards that support synchronous output.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DaqOutScan(ByVal chanArray As Short(), ByVal chanTypeArray As MccDaq.ChannelType
(),ByVal gainArray As MccDaq.Range, ByVal chanCount As Integer, ByRef rate As Integer, ByVal count As
Integer, ByVal memHandle As IntPtr, ByVal options As MccDaq.ScanOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo DaqOutScan (short[] chanArray, MccDaq.ChannelType[] chanTypeArray, MccDaq.Range
[] gainArray, int chanCount, ref int rate, int count, IntPtr memHandle, MccDaq.ScanOptions options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function DaqOutScan(ByVal chanArray As Short(), ByVal chanTypeArray As MccDaq.ChannelType
(),ByVal gainArray As MccDaq.Range, ByVal chanCount As Integer, ByRef rate As Integer, ByVal count As
Integer, ByVal memHandle As Integer, ByVal options As MccDaq.ScanOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo DaqOutScan(short[] chanArray, MccDaq.ChannelType[] chanTypeArray, MccDaq.Range
[] gainArray, int chanCount, ref int rate, int count, int memHandle, MccDaq.ScanOptions options)
```

## Parameters

*chanArray*

Array containing channel values. Valid channel values are analog output channels and digital ports.

*chanTypeArray*

Array containing channel types. Each element of this array defines the type of the corresponding element in the chanArray.

The chanTypeArray settings are [MccDaq.ChannelType](#) enumerated constants. Choices are:

- Analog: Analog output channel.
- Digital16: 16-bit digital output port (FirstPortA only).

*gainArray*

Array containing D/A range codes. If the corresponding element in the chanArray is not an analog output channel, the range code for this channel is ignored. If the board does not have programmable gain, this parameter is ignored, and therefore can be set to null.

*chanCount*

Number of elements in each of the three arrays - chanArray, chanTypeArray, and gainArray.

*rate*

Sample rate in scans per second. Rate also returns the value of the actual rate set, which may be different from the requested rate because of pacer limitations.

*count*

Sets the total number of values to output. count must be a multiple of chanCount.

*memHandle*

Handle for the Windows buffer from which data is output. This buffer must have been previously allocated with the [WinBufAllocEx\(\)](#), [WinBufAlloc32Ex\(\)](#), or [WinBufAlloc64Ex\(\)](#) method, and data values loaded, for example using [WinArrayToBuf\(\)](#).

*options*

Bit fields that control various options. All of the options settings are [MccDaq.ScanOptions](#) enumerated constants. This field may contain any combination of non-contradictory choices in the [options parameter values](#) section below.

## Returns

- An [ErrorInfo](#) object that indicates the status of the operation.

- rate – Actual sampling rate used.

## options parameter values

ADCClock	When this option is used, the data output operation is paced by the ADC clock.
ADCClockTrig	If this option is used, the data output operation is triggered upon the start of the ADC clock.
Background	<p>When this option is used, the output operations begin running in the background, and control immediately returns to the next line of your program.</p> <p>Use <a href="#">GetStatus()</a> with <a href="#">DaqoFunction</a> to check the status of background operation. Use <a href="#">StopBackground()</a> method with DaqoFunction to terminate background operations before they are completed. Execute StopBackground() with DaqoFunction after normal termination of all background functions in order to clear variables and flags.</p>
Continuous	This option puts the method in an endless loop. Once it outputs the specified number (count) of output values, it resets to the start of the buffer and begins again. The only way to stop this operation is by calling <a href="#">StopBackground()</a> with DaqoFunction. This option should only be used in combination with Background so that your program regains control.
ExtClock	<p>If this option is used, conversions are paced by the signal on the external clock input rather than by the internal pacer clock. Each conversion is triggered on the appropriate edge of the clock input signal.</p> <p>When this option is used, the rate parameter is ignored. The sampling rate is dependent on the clock signal. Options for the board default to transfer types that allow the maximum conversion rate to be attained unless otherwise specified.</p>
NonStreamedIO	<p>This option allows non-streamed data output to be generated to a specified output channel.</p> <p>In this mode, the aggregate size of data output buffer must be less than or equal to the size of the internal data output FIFO on the Measurement Computing device. This allows the data output buffer to be loaded into the device's internal output FIFO.</p> <p>Once the sample updates are transferred (or downloaded) to the device, the device is responsible for outputting the data. While the size is limited, and the output buffer cannot be changed once the output is started, this mode has the advantage being able to continue data output without having to periodically feed output data through the program to the device.</p>

## DaqSetSetpoints() method

Configures up to 16 detection setpoints associated with the input channels within a scan group. This method only works with boards that support synchronous input.

Member of the [MccBoard](#) class.

### Function Prototype

VB .NET

```
Public Function DaqSetSetpoints(ByVal limitAArray As Single(), ByVal limitBArray As Single(), ByVal reserved As Single(), ByVal setpointFlagsArray As MccDaq.SetpointFlag, ByVal setpointOutputArray As MccDaq.SetpointOutput, ByVal output1Array As Single(), ByVal output2Array As Single(), outputMask1Array As Single(), outputMask2Array As Single(), ByVal setpointCount As Integer) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo DaqSetSetpoints(float[] limitAArray, float[] limitBArray, float[] reserved, MccDaq.SetpointFlag[] setpointFlagsArray, MccDaq.SetpointOutput[] setpointOutputArray, float[] output1Array, float[] output2Array, float[] outputMask1Array, float[] outputMask2Array, int setpointCount)
```

### Parameters

*limitAArray*

Array containing the limit A values for the input channels used for the setpoint. Limit A specifies a value used to determine if the setpoint criteria are met.

*limitBArray*

Array containing the limit B values for the input channels used for the setpoint. Limit B specifies a value used to determine if the setpoint criteria are met.

*reserved*

Reserved for future use.

*setpointFlagsArray*

Array containing the setpoint flags.

All of the setpointFlagsArray settings are [MccDaq.SetpointFlag](#) enumerated constants. Set it to one of the constants in the [setpointFlagsArray parameter values](#) section below.

*setpointOutputArray*

Array containing output sources.

All of the setpointOutputArray settings are [MccDaq.SetPointOutput](#) enumerated constants. Set it to one of the constants in the [setpointOutputArray parameter values](#) section below.

*output1Array*

Array containing the values for the output channels used for the setpoint.

*output2Array*

Array containing the values for the output channels used for the setpoint.

*outputMask1Array*

Array containing the output masks for output value 1 (for FIRSTPORTC only).

*outputMask2Array*

Array containing the output masks for output value 2 (for FIRSTPORTC only).

*setpointCount*

Number of setpoints to configure (0 to 16). Set to 0 to disable the setpoints.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### **setpointFlagsArray** parameter values

Flag	Description
EqualLimitA	Setpoint criteria: The input channel = limit A.
LessThanLimitA	Setpoint criteria: The input channel < limit A.
GreaterThanLimitB	Setpoint criteria: The input channel > limit B.
OutsideLimits	Setpoint criteria: The input channel < limit A and > limit B.
InsideLimits	Setpoint criteria: The input channel > limit A and < limit B.
Hysteresis	Setpoint criteria: If the input channel > limit A then output value 1. If the input channel < limit B then output value 2.
UpdateOnTrueOnly	If the criteria is met then output value 1.
UpdateOnTrueAndFalse	If the criteria is met then output value 1, else output value 2.

### **setpointOutputArray** parameter values

Output source	Description
None	Perform no outputs.
FirstPortC	Output to FirstPortC when the criteria is met.
DigitalPort	Output to digital port when the criteria is met.
DAC0	Output to DAC0 when the criteria is met. You must have a device with DAC0.
DAC1	Output to DAC1 when the criteria is met. You must have a device with DAC1.
DAC2	Output to DAC2 when the criteria is met. You must have a device with DAC2.
DAC3	Output to DAC3 when the criteria is met. You must have a device with DAC3.
TMR0	Output to timer 0 when the criteria is met.
TMR1	Output to timer 1 when the criteria is met.

## DaqSetTrigger() method

Selects the trigger source and sets up its parameters. This trigger is used to initiate or terminate an acquisition using the [DaqInScan\(\)](#) function if the ExtTrigger option is selected. This method only works with boards that support synchronous output.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function DaqSetTrigger(ByVal trigSource As MccDaq.TriggerSource, ByVal trigSense MccDaq.TriggerSensitivity, ByVal trigChan As Integer, ByVal chanType As MccDaq.ChannelType, ByVal gain As MccDaq.Range, ByVal level As Single, ByVal variance As Single, ByVal trigEvent As MccDaq.TriggerEvent) As MccDaq.ErrorInfo
```

C#.NET

```
public MccDaq.ErrorInfo DaqSetTrigger(MccDaq.TriggerSource trigSource, MccDaq.TriggerSensitivity trigSense, int trigChan, MccDaq.ChannelType chanType, MccDaq.Range gain, float level, float variance, MccDaq.TriggerEvent trigEvent)
```

## Parameters

*trigSource*

Specifies the type of triggering based on the external trigger source.

All of the trigSource settings are [MccDaq.TriggerSource](#) enumerated constants. Set it to one of the constants in the [trigSource parameter values](#) section below.

*trigSense*

Specifies the trigger sensitivity. The trigger sensitivity normally defines the way in which a trigger event is detected based upon the characteristics of the trigger input signal. Often, it defines the way in which the trigger input signal(s) should be compared to the trigger level parameter value.

All of the trigSense settings are [MccDaq.TriggerSensitivity](#) enumerated constants. Set it to one of the constants in the [trigSense parameter values](#) section below.

*trigChan*

The trigger channel. This channel must be a configured channel in the channel array (refer to [DaqInScan\(\)](#)).

*chanType*

The channel type. All of the chanType settings are [MccDaq.ChannelType](#) enumerated constants. chanType should match the channel type setting for the trigger channel configured using the [DaqInScan\(\)](#) method.

*gain*

The trigger channel gain code. If the device has programmable gain, this parameter should match the gain code setting when the channel is configured using the [DaqInScan\(\)](#) method. The gain parameter is ignored if trigChan is not an analog channel.

*level*

A single precision floating point value which represents, in engineering units, the level at or around which the trigger event should be detected.

This option is used for trigger types that depend on an input channel comparison to detect the start trigger or stop trigger event.

The actual level at which the trigger event is detected depends upon trigger sensing and variability. Refer to the [Trigger levels](#) section below for more information.

*variance*

A single-precision floating point value which represents, in engineering units, the amount that the trigger event can vary from the level parameter.

While the TrigSense parameter indicates the direction of the input signal relative to the level parameter, the variance parameter specifies the degree to which the input signal can vary relative to the level parameter.

*trigEvent*

Specifies the trigger event type. Valid values indicate either a start trigger event ([MccDaq.TriggerEvent.Start](#)) or a stop trigger event ([MccDaq.TriggerEvent.Stop](#)).

**Start:** The start trigger event defines the conditions under which post-trigger acquisition data collection should be initiated or triggered. The start trigger event can vary in complexity from starting immediately, to starting on complex channel value definitions.

**Stop:** The stop trigger event signals the current data acquisition process to terminate. The stop event can be as simple as that of a scan count, or as complex as involving a channel value level condition.



## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## trigSource parameter values

TrigImmediate	Start trigger event only. Acquisition begins immediately upon invocation the <a href="#">DagInScan()</a> method. No pre-trigger data acquisition is possible with this trigger type.
TrigExtTTL	Start trigger event only. Acquisition begins on the selectable edge of an external TTL signal. No pre-trigger data acquisition is possible with this trigger type.
TrigAnalogHW	Start trigger event only. Acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.) trigChan must be defined as the first channel in the channel scan group. No pre-trigger data acquisition is possible with this trigger type.
TrigAnalogSW	Post-trigger data acquisition begins upon a selectable criteria of the input signal (above level, below level, rising edge, etc.)
TrigDigPattern	Post-trigger data acquisition beings upon receiving a specified digital pattern on the specified digital port.
TrigCounter	Post-trigger data acquisition begins upon detection of specified counter criteria.
TrigScanCount	Stop trigger event only. Stops collecting post-trigger data when the specified number of post-trigger scans are completed.

## trigSense parameter values

RisingEdge	Triggers when the signal goes from low to high (TTL trigger), or rises through a specified level (hardware analog, software analog, and counter).
FallingEdge	Triggers when the signal goes from high to low (TTL trigger), or falls through a specified level (hardware analog, software analog, and counter).
AboveLevel	Triggers when the signal is above a specified level (hardware analog, software analog, counter, and digital pattern).
BelowLevel	Triggers when the signal is below a specified level (hardware analog, software analog, counter, and digital pattern).
EqLevel	Triggers when the signal equals a specified level (hardware analog, software analog, counter, and digital pattern).
NeLevel	Triggers when the signal does not equal a specified level (hardware analog, software analog, counter, and digital pattern).

## Trigger levels

The actual level at which the trigger event is detected depends upon trigger sensing and variability. The various ranges of possible values for the level parameter based on the trigger source are:

- TrigAnalogHW: The voltage used to define the trigger level. Trigger detection is performed in hardware.
- TrigAnalogSW: The voltage used to define the trigger level. Trigger detection is performed in software.
- TrigDigPattern: Sets the bit pattern for the digital channel trigger. Choices are:  
0.0 (no bits set): 255.0 (all bits set) for 8-bit digital ports.  
0.0 (no bits set): 65,535.0 (all bits set) for 16-bit digital ports.
- TrigCounter: Selects either Pulse or Totalize counter values (0.0 – 65,535).
- TrigImmediate: Ignored
- TrigScanCount: Ignored

## Trigger start and stop criteria

The table below lists the trigger start and stop criteria based on the selected trigger type and sensitivity.

Trigger Start/Stop Source (trigSource)	Trigger Sensitivity (trigSense)	Trigger Start/Stop Criteria
TrigAnalogHW (Start trigger event only)	RisingEdge	Triggers when the signal value $< (\text{level} - \text{variance})$ . Then, the signal value $> \text{level}$ .
	FallingEdge	Triggers when the signal value $> (\text{level} + \text{variance})$ . Then, the signal value $< \text{level}$ .
	AboveLevel	Triggers when the signal value $> (\text{level})$ .
	BelowLevel	Triggers when the signal value $< (\text{level})$ .
TrigAnalogSW	RisingEdge	Triggers/stops when the signal value $< (\text{level} - \text{variance})$ . Then, the signal value $> \text{level}$ .
	FallingEdge	Triggers/stops when the signal value $> (\text{level} + \text{variance})$ . Then, the signal value $< \text{level}$ .
	AboveLevel	Triggers/stops when the signal value $> (\text{level})$ .
	BelowLevel	Triggers/stops when the signal value $< (\text{level})$ .
	EqLevel	Triggers/stops when $(\text{level} - \text{variance}) < \text{signal value} < (\text{level} + \text{variance})$ .
	NeLevel	Triggers/stops when the signal value $< (\text{level} - \text{variance})$ OR when the signal value $> (\text{level} + \text{variance})$ .
TrigDigPattern	AboveLevel	Triggers/stops when the (digital port value AND (bitwise) variance) $> (\text{level AND (bitwise) variance})$ .
	BelowLevel	Triggers/stops when the (digital port value AND (bitwise) variance) $< (\text{level AND (bitwise) variance})$ .
	EqLevel	Triggers/stops when the (digital port value AND (bitwise) variance) $= (\text{level AND (bitwise) variance})$ .
	NeLevel	Triggers/stops when the (digital port value AND (bitwise) variance) $\neq (\text{level AND (bitwise) variance})$ .
TrigCounter	RisingEdge	Triggers/stops when the counter channel $< (\text{level} - \text{variance})$ . Then, the counter channel $> \text{level}$ .
	FallingEdge	Triggers/stops when counter channel $> (\text{level} + \text{variance})$ . Then, the counter channel $< \text{level}$ .
	AboveLevel	Triggers/stops when the counter channel $> (\text{level} - \text{variance})$ .
	BelowLevel	Triggers/stops when the counter channel $< (\text{level} + \text{variance})$ .
	EqLevel	Triggers/stops when $(\text{level} - \text{variance}) < \text{counter channel} < (\text{level} + \text{variance})$ .
	NeLevel	Triggers/stops when the counter channel $< (\text{level} - \text{variance})$ OR when the counter channel $> (\text{level} + \text{variance})$ .

# TIn() method

Reads an analog input channel, linearizes it according to the selected temperature sensor type, if required, and returns the temperature in units determined by the Scale argument.

The CJC channel, the gain, and sensor type are read from the InstaCal configuration file. Run the InstaCal configuration program to set these items.

Member of the [MccBoard](#) class.

## Function Prototype

```
VB .NET
Public Function TIn(ByVal chan As Integer, ByVal scale As MccDaq.TempScale ByRef tempValue As Single,
ByVal options As MccDaq.ThermocoupleOptions) As MccDaq.ErrorInfo

C# .NET
public MccDaq.ErrorInfo TIn(int chan, MccDaq.TempScale scale, out float tempValue,
MccDaq.ThermocoupleOptions options)
```

## Parameters

- chan*  
Input channel to read.
- scale*  
Specifies the temperature scale that the input is converted to. Choices are [MccDaq.TempScale.Celsius](#), [MccDaq.TempScale.Fahrenheit](#), [MccDaq.TempScale.Kelvin](#), [MccDaq.TempScale.Volts](#), and [MccDaq.TempScale.NoScale](#).
- tempValue*  
The temperature in units determined by the Scale argument is returned here.
- options*  
Bit fields that control various options. Set it to one of the constants in the "[options parameter values](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- tempValue - The temperature is returned here.

## options parameter values

All of the options settings are [MccDaq.ThermocoupleOptions](#) enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ThermocoupleOptions enumeration (for example, variable=[MccDaq.ThermocoupleOptions.Filter](#) or variable = [MccDaq.ThermocoupleOptions.NoFilter](#)).

Filter	When selected, a smoothing function is applied to temperature readings, very much like the electrical smoothing inherent in all hand held temperature sensor instruments. This is the default. Ten samples are read from the specified channel and averaged. The average is the reading returned. Averaging removes normally distributed signal line noise.
NoFilter	When selected, the temperature readings are not smoothed, resulting in a scattering of readings around a mean.

Refer to the board-specific information in the *Universal Library User's Guide* to determine if your hardware supports these options.

## Notes

### scale options

- Specify the *NoScale* option to retrieve raw data from the device. When NoScale is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.
- Specify the *Volts* option to read the voltage input of a thermocouple.

Refer to board-specific information in the *Universal Library User's Guide* to determine if your hardware supports these options.

## Using CIO-EXP boards

For CIO-EXP boards, the channel number is calculated using the following formula:

$$\text{chan} = (\text{ADChan} \times 16) + (16 + \text{MuxChan})$$

where:

ADChan is the A/D channel that is connected to the multiplexer.

MuxChan is a number ranging from 0 to 15 that specifies the channel number on a particular bank of the multiplexer board.

For example, you have an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0. (Remember that DAS08 channels are numbered 0, 1, 2, 3, 4, 5, 6 and 7). If you connect a thermocouple to channel 5 of the EXP16, the value for Chan would be  $(0 \times 16) + (16 + 5) = 0 + 21 = 21$ .

## CJC channel

The CJC channel is set in the InstaCal installation and configuration program. If you have multiple EXP boards, the Universal Library will apply the CJC reading to the linearization formula in the following manner:

- 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.
- If you 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: You have four CIO-EXP16 boards connected to a CIO-DAS08 on channel 0, 1, 2 and 3. You choose CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7, AND, you leave the CIO-EXP16's 2, 3 and 4 CJC channels to NOT SET. Result: The CIO-EXP boards will all use the CJC reading from CIO-EXP16 #1, connected to channel 7 for linearization. *It is important to keep the CIO-EXP boards in the same case and out of any breezes to ensure a clean CJC reading.*

## Specific Errors

If an [OUTOFRANGE](#) or [OPENCONNECTION](#) error occurs, the value returned in TempValue is -9999.0. If a [NOTREADY](#) error occurs, the value returned in TempValue is -9000.

## 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 range is read from the configuration file (CB.CFG). In most cases, hardware selectable ranges should be set to  $\pm 5$  V for thermocouples, and 0–10 V for RTDs. Refer to the board-specific information in the *Universal Library User's Guide* or in the user manual for your board. If the board has programmable gains, the TIn() method will set the appropriate A/D range.

# TInScan() method

Reads a range of channels from an analog input board, linearizes them according to temperature sensor type, if required, and returns the temperatures to an array in units determined by the Scale argument.

The CJC channel, the gain, and temperature sensor type are read from the configuration file. Use the InstaCal configuration program to change any of these options.

Member of the [MccBoard](#) class.

## Function Prototype

VB .NET

```
Public Function TInScan(ByVal lowChan As Integer, ByVal highChan As Integer, ByVal scale As MccDaq.TempScale, ByVal dataBuffer As Single(), ByVal options As MccDaq.ThermocoupleOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo TInScan(int lowChan, int highChan, TempScale scale, float[] dataBuffer, MccDaq.ThermocoupleOptions options)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Function TInScan(ByVal lowChan As Integer, ByVal highChan As Integer, ByVal scale As MccDaq.TempScale, ByRef dataBuffer As Single, ByVal options As MccDaq.ThermocoupleOptions) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo TInScan(int lowChan, int highChan, MccDaq.TempScale scale, out float dataBuffer, MccDaq.ThermocoupleOptions options)
```

## Parameters

*lowChan*

Low channel of the scan.

*highChan*

High channel of the scan.

*scale*

Specifies the temperature scale that the input is converted to. Choices are MccDaq.TempScale.Celsius, MccDaq.TempScale.Fahrenheit, MccDaq.TempScale.Kelvin, MccDaq.TempScale.Volts, and MccDaq.TempScale.NoScale.

*dataBuffer*

The temperature is returned in units determined by the Scale argument. 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.

*options*

Bit fields that control various options. Set it to one of the constants in the "[options parameter values](#)" section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataBuffer[] - Temperature values in degrees are returned here for each channel in the scan.

## options parameter values

All of the options settings are [MccDaq.ThermocoupleOptions](#) enumerated constants. To set a variable to one of these constants, you must refer to the MccDaq object and the ThermocoupleOptions enumeration (for example, variable = MccDaq.ThermocoupleOptions.Filter or variable = MccDaq.ThermocoupleOptions.NoFilter).

Filter	When selected, a smoothing function is applied to temperature readings, very much like the electrical smoothing inherent in all hand held temperature sensor instruments. This is the default. Ten samples are read from the specified channel and averaged. The average is the reading returned. Averaging removes normally distributed signal line noise.
NoFilter	When selected, the temperature readings are not smoothed, resulting in a scattering of readings around a mean.

## Notes

### scale options

- Specify the *NoScale* option to retrieve raw data from the device. When NoScale is specified, calibrated data is returned, although a cold junction compensation (CJC) correction factor is not applied to the returned values.
- Specify the *Volts* option to read the voltage input of a thermocouple.

Refer to board-specific information in the *Universal Library User's Guide* to determine if your hardware supports these options.

### Using CIO-EXP boards

For CIO-EXP boards, the channel number is calculated using the following formula:

$$\text{Chan} = (\text{ADChan} \times 16) + (16 + \text{MuxChan})$$

where:

ADChan is the A/D channel that is connected to the multiplexer.

MuxChan is a number ranging from 0 to 15 that specifies the channel number on a particular bank of the multiplexer board.

For example, you have an EXP16 connected to a CIO-DAS08 via the CIO-DAS08 channel 0. (Remember that DAS08 channels are numbered 0, 1, 2, 3, 4, 5, 6 and 7). If you connect a thermocouple to channel 5 of the EXP16, the value for Chan would be  $(0 \times 16) + (16 + 5) = 0 + 21 = 21$ .

### CJC channel

The CJC channel is set in the InstaCal installation and configuration program. If you have multiple EXP boards, the Universal Library will apply the CJC reading to the linearization formula in the following manner:

- 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.
- If you 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: You have four CIO-EXP16 boards connected to a CIO-DAS08 on channel 0, 1, 2 and 3. You choose CIO-EXP16 #1 (connected to CIO-DAS08 channel 0) to have its CJC read on CIO-DAS08 channel 7, AND, you leave the CIO-EXP16's 2, 3 and 4 CJC channels to NOT SET. Result: The CIO-EXP boards will all use the CJC reading from CIO-EXP16 #1, connected to channel 7 for linearization. *It is important to keep the CIO-EXP boards in the same case and out of any breezes to ensure a clean CJC reading.*

### Specific Errors

For most boards, if an [OUTOFRANGE](#) or [OPENCONNECTION](#) error occurs, the value in the array element associated with the channel causing the error returned will be -9999.0.

## 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 range is read from the configuration file (CB.CFG). In most cases, hardware selectable ranges should be set to  $\pm 5$  V for thermocouples, and 0–10 V for RTDs. Refer to the board-specific information in the *Universal Library User's Guide* or in the user manual for your board. If the board has programmable gains, the `TIn()` method will set the appropriate A/D range.

## WinArrayToBuf() method

Copies data from a one-dimensional or two-dimensional array into a Windows memory buffer.

Member of the [MccService class](#).

### Function Prototype

VB .NET

Copies data from a one-dimensional array of short values:

```
Public Shared Function WinArrayToBuf(ByVal dataArray As Short(), ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data from a one-dimensional array of System.UInt16 values:

```
Public Shared Function WinArrayToBuf(ByVal dataArray As System.UInt16(), ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data from a two-dimensional array of double values:

```
Public Shared WinArrayToBuf(ByVal dataArray As Double(), ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal numPoints As Integer, ByVal numChannels As Integer) As MccDaq.ErrorInfo
```

C# .NET

Copies data from a one-dimensional array of short values:

```
public MccDaq.ErrorInfo WinArrayToBuf(short[] dataArray, IntPtr memHandle, int firstPoint, int numPoints)
```

Copies data from a one-dimensional array of System.UInt16 values:

```
public MccDaq.ErrorInfo WinArrayToBuf(ushort[] dataArray, IntPtr memHandle, int firstPoint, int numPoints)
```

Copies data from a two-dimensional array of double values:

```
public MccDaq.ErrorInfo WinArrayToBuf(double[,] dataArray, IntPtr memHandle, int firstPoint, int numPoints, int numChannels)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

Copies data from a one-dimensional array of short values:

```
Public Shared Function WinArrayToBuf(ByRef dataArray As Short, ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data from a one-dimensional array of System.UInt16 values:

```
Public Shared Function WinArrayToBuf(ByRef dataArray As System.UInt16, ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data from a two-dimensional array of double values:

```
Public Shared WinArrayToBuf(ByRef dataArray As Double(), ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal numPoints As Integer, ByVal numChannels As Integer) As MccDaq.ErrorInfo
```

C# .NET

Copies data from a one-dimensional array of short values:

```
public static MccDaq.ErrorInfo WinArrayToBuf(ref short dataArray, int memHandle, int firstPoint, int numPoints)
```

Copies data from a one-dimensional array of System.UInt16 values:

```
public static MccDaq.ErrorInfo WinArrayToBuf(ref ushort dataArray, int memHandle, int firstPoint, int numPoints)
```

Copies data from a two-dimensional array of double values:

```
public static MccDaq.ErrorInfo WinArrayToBuf(ref double[,] dataArray, int memHandle, int firstPoint, int numPoints, int numChannels)
```

### Parameters

*dataArray*

The array containing the data to be copied. The first dimension should equal the number of channels. The second dimension

should equal the number of points/channel.

*memHandle*

This must be a memory handle that was returned by [WinBufAllocEx\(\)](#) when the buffer was allocated. The data will be copied into this buffer.

*firstPoint*

Index of the first point in the memory buffer where data will be copied to.

*numPoints*

Number of data points to copy from dataArray.

*numChannels*

Number of channels to copy from dataArray.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- This method copies data from an array to a Windows global memory buffer. This would typically be used to initialize the buffer with data before doing an output scan. You can use the firstPoint and numPoints parameters to fill a portion of the buffer. This is useful if you want to send new data to the buffer after a Background + Continuous output scan has been started, for example during circular buffering.



## WinBufAlloc() method (deprecated)

Allocates a Windows global memory buffer which can be used with the scan methods, and returns a memory handle for it.

**Deprecated**; unless your application calls deprecated methods, use [WinBufAllocEx\(\)](#).

Most devices return data in a 16-bit format. For these devices, the buffer can be created using WinBufAlloc(). Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using [WinBufAlloc32\(\)](#) or [WinBufAlloc64\(\)](#). See hardware-specific information to determine the type of buffer needed. If not specifically mentioned, use WinBufAlloc().

Member of the [MccService class](#).

## Function Prototype

VB .NET

```
Public Shared Function WinBufAlloc(ByVal numPoints As Integer) As Integer
```

C# .NET

```
public static int WinBufAlloc(int numPoints)
```

## Parameters

*numPoints*

The size of buffer to allocate. Specifies how many data points (16-bit integers, NOT bytes) can be stored in the buffer.

## Returns

- 0, if the buffer could not be allocated, or a non-zero integer handle to the buffer.

## Notes

- Unlike most other methods in the library, this method does not return an [ErrorInfo object](#). It returns a Windows global memory handle, which can then be passed to the scan methods in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.

## WinBufAllocEx() method

Allocates a Windows global memory buffer which can be used with the scan methods, and returns a memory handle for it.

Most devices return data in a 16-bit format. For these devices, the buffer can be created using WinBufAlloc(). Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using [WinBufAlloc32Ex\(\)](#) or [WinBufAlloc64Ex\(\)](#). See hardware-specific information to determine the type of buffer needed. If not specifically mentioned, use WinBufAllocEx().

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Function WinBufAllocEx(ByVal numPoints As Integer) As IntPtr
```

C# .NET

```
public IntPtr WinBufAllocEx(int numPoints)
```

### Parameters

*numPoints*

The size of buffer to allocate. Specifies how many data points (16-bit integers, NOT bytes) can be stored in the buffer.

### Returns

- 0, if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other methods in the library, this method does not return an [ErrorInfo object](#). It returns a Windows global memory handle, which can then be passed to the scan methods in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.
- This method is preferred over the deprecated method [WinBufAlloc\(\)](#). Only use WinBufAlloc() in 32-bit legacy applications that call deprecated methods.

## WinBufAlloc32() method (deprecated)

Allocates a 32-bit Windows global memory buffer for use with 32-bit scan functions, and returns a memory handle for the buffer.

**Deprecated;** unless your application calls deprecated methods, use [WinBufAlloc32Ex\(\)](#).

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Shared Function WinBufAlloc32(ByVal numPoints As Integer) As Integer
```

C# .NET

```
public int WinBufAlloc32(int numPoints)
```

### Parameters

*numPoints*

The size of the buffer to allocate. Specifies how many data points (32-bit integers, NOT bytes) that the buffer will hold.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other methods in the library, this method does not return an [ErrorInfo object](#). It returns a Windows global memory handle which can then be passed to the scan methods in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.
- When using devices that return data in a 16-bit format, the buffer can be created using [WinBufAlloc\(\)](#). Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using [WinBufAlloc32\(\)](#) or [WinBufAlloc64\(\)](#). See hardware-specific information to determine the type of buffer needed.

## WinBufAlloc32Ex() method

Allocates a 32-bit Windows global memory buffer for use with 32-bit scan functions, and returns a memory handle for the buffer.

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Function WinBufAlloc32Ex(ByVal numPoints As Integer) As IntPtr
```

C# .NET

```
public IntPtr WinBufAlloc32Ex(int numPoints)
```

### Parameters

*numPoints*

The size of the buffer to allocate. Specifies how many data points (32-bit integers, NOT bytes) that the buffer will hold.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other methods in the library, this method does not return an [ErrorInfo object](#). It returns a Windows global memory handle which can then be passed to the scan methods in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.
- When using devices that return data in a 16-bit format, the buffer can be created using [WinBufAllocEx\(\)](#). Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using WinBufAlloc32Ex() or [WinBufAlloc64Ex\(\)](#). See hardware-specific information to determine the type of buffer needed.
- This method is preferred over the deprecated method [WinBufAlloc32\(\)](#). Only use WinBufAlloc32() in 32-bit legacy applications that call deprecated methods.

## WinBufAlloc64() method (deprecated)

Allocates a 64-bit Windows global memory buffer for use with 64-bit scan functions, and returns a memory handle for the buffer.

**Deprecated**; unless your application calls deprecated methods, use [WinBufAlloc64Ex\(\)](#).

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Shared Function WinBufAlloc64(ByVal numPoints As Integer) As Integer
```

C# .NET

```
public static int WinBufAlloc64(int numPoints)
```

### Parameters

*numPoints*

The size of the buffer to allocate. Specifies how many data points (64-bit integers, NOT bytes) that the buffer will hold.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other methods in the library, this function does not return an [ErrorInfo object](#). It returns a Windows global memory handle which can then be passed to the scan functions in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.
- When using devices that return data in a 16-bit format, the buffer can be created using [WinBufAlloc\(\)](#). Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using [WinBufAlloc32\(\)](#) or [WinBufAlloc64\(\)](#). See hardware-specific information to determine the type of buffer needed.

## WinBufAlloc64Ex() method

Allocates a 64-bit Windows global memory buffer for use with 64-bit scan functions, and returns a memory handle for the buffer.

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Function WinBufAlloc64Ex(ByVal numPoints As Integer) As IntPtr
```

C# .NET

```
public IntPtr WinBufAlloc64Ex(int numPoints)
```

### Parameters

*numPoints*

The size of the buffer to allocate. Specifies how many data points (64-bit integers, NOT bytes) that the buffer will hold.

### Returns

- 0 if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- Unlike most other methods in the library, this function does not return an [ErrorInfo object](#). It returns a Windows global memory handle which can then be passed to the scan functions in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.
- When using devices that return data in a 16-bit format, the buffer can be created using [WinBufAllocEx\(\)](#). Some devices return data in higher resolution formats, or the resolution of the data can vary depending on various options used to collect the data. In these cases, determine if the buffer needs to be created using [WinBufAlloc32Ex\(\)](#) or [WinBufAlloc64Ex\(\)](#). See hardware-specific information to determine the type of buffer needed.
- This method is preferred over the deprecated method [WinBufAlloc64\(\)](#). Only use [WinBufAlloc64\(\)](#) in 32-bit legacy applications that call deprecated methods.

## WinBufFree() method (deprecated)

Frees a Windows global memory buffer which was previously allocated with [WinBufAlloc\(\)](#), [WinBufAlloc32\(\)](#), or [WinBufAlloc64\(\)](#).

**Deprecated**; unless your application calls deprecated methods, use [WinBufFreeEx\(\)](#).

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Shared Function WinBufFree (ByVal memHandle As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public static MccDag.ErrorInfo WinBufFree (int memHandle)
```

### Parameters

*memHandle*

A Windows memory handle. This must be a memory handle that was returned by [WinBufAlloc\(\)](#), [WinBufAlloc32\(\)](#), or [WinBufAlloc64\(\)](#) when the buffer was allocated.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## WinBufFreeEx() method

Frees a Windows global memory buffer which was previously allocated with [WinBufAllocEx\(\)](#), [WinBufAlloc32Ex\(\)](#), or [WinBufAlloc64Ex\(\)](#).

**Deprecated**; unless your application calls deprecated methods, use [WinBufFreeEx\(\)](#).

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Function WinBufFreeEx(ByVal memHandle As IntPtr) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo WinBufFreeEx(IntPtr memHandle)
```

### Parameters

*memHandle*

A Windows memory handle. This must be a memory handle that was returned by [WinBufAllocEx\(\)](#), [WinBufAlloc32Ex\(\)](#), or [WinBufAlloc64Ex\(\)](#) when the buffer was allocated.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

- This method is preferred over the deprecated method [WinBufFree\(\)](#). Only use ScaledWinBufAlloc() in 32-bit legacy applications that call deprecated methods.



## WinBufToArray() method

Copies data from a Windows memory buffer into a one-dimensional or two-dimensional array.

Member of the [MccService class](#).

### Function Prototype

VB .NET

Copies data to a one-dimensional array of short values:

```
Public Function WinBufToArray(ByVal memHandle As IntPtr, ByVal dataArray As Short(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data to a one-dimensional array of System.UInt16 values:

```
Public Function WinBufToArray(ByVal memHandle As IntPtr, ByVal dataArray As System.UInt16(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data to a two-dimensional array of double values:

```
public ErrorInfo WinBufToArray(ByVal memHandle As IntPtr, ByVal dataArray As Double(), ByVal firstPoint As Integer, ByVal numPoints As Integer, ByVal numChannels As Integer) As MccDaq.ErrorInfo
```

C# .NET

Copies data to a one-dimensional array of short values:

```
public MccDaq.ErrorInfo WinBufToArray(IntPtr memHandle, short[] dataArray, int firstPoint, int numPoints)
```

Copies data to a one-dimensional array of System.UInt16 values:

```
public MccDaq.ErrorInfo WinBufToArray(IntPtr memHandle, ushort[] dataArray, int firstPoint, int numPoints)
```

Copies data to a two-dimensional array of double values:

```
public MccDaq.ErrorInfo WinBufToArray(IntPtr memHandle, double[,] dataArray, int firstPoint, int numPoints, int numChannels)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

Copies data to a one-dimensional array of short values:

```
Public Shared Function WinBufToArray(ByVal memHandle As Integer, ByRef dataArray As Short, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data to a one-dimensional array of System.UInt16 values:

```
Public Shared Function WinBufToArray(ByVal memHandle As Integer, ByRef dataArray As System.UInt16, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data to a two-dimensional array of double values:

```
public static ErrorInfo WinBufToArray(ByVal memHandle As Integer, ByRef dataArray As Double(), ByVal firstPoint As Integer, ByVal numPoints As Integer, ByVal numChannels As Integer) As MccDaq.ErrorInfo
```

C# .NET

Copies data to a one-dimensional array of short values:

```
public static MccDaq.ErrorInfo WinBufToArray(int memHandle, out short dataArray, int firstPoint, int numPoints)
```

Copies data to a one-dimensional array of System.UInt16 values:

```
public static MccDaq.ErrorInfo WinBufToArray(int memHandle, out ushort dataArray, int firstPoint, int numPoints)
```

Copies data to a two-dimensional array of double values:

```
public static MccDaq.ErrorInfo WinBufToArray(int memHandle, out double[,] dataArray, int firstPoint, int numPoints, int numChannels)
```

### Parameters

*memHandle*

This must be a memory handle that was returned by [WinBufAllocEx\(\)](#) when the buffer was allocated. The data will be copied from this buffer

*dataArray*

The array that the data will be copied to. The first dimension should equal the number of channels. The second dimension should equal the number of points/channel.

*firstPoint*

Index of the first point in the memory buffer that data will be copied from.

*numPoints*

Number of data points to copy into dataArray.

*numChannels*

Number of channels to copy into dataArray.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- This method copies data from a Windows global memory buffer to a single value or into an array of doubles. This would typically be used to retrieve data from the buffer after executing an input scan method.
- You can use the firstPoint and numPoints parameters to copy only a portion of the buffer to the array. This can be useful if you want foreground code to manipulate previously collected data while a Background scan continues to collect new data.

## WinBufToArray32() method

Copies 32-bit data from a Windows global memory buffer into a one-dimensional or two-dimensional array. This method is typically used to retrieve data from the buffer after executing an input scan method.

Member of the [MccService class](#).

### Function Prototype

#### VB .NET

Copies data into a two-dimensional array of double values:

```
Public Shared Function WinBufToArray32(ByVal memHandle As IntPtr, ByVal dataArray(,) As Double, ByVal firstPoint As Integer, ByVal numPoints As Integer, ByVal numChannels As Integer) As MccDaq.ErrorInfo
```

Copies data into an array of integer values:

```
Public Shared Function WinBufToArray32(ByVal memHandle As IntPtr, ByVal dataArray As Integer(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data into an array of System.UInt32 values:

```
Public Shared Function WinBufToArray32(ByVal memHandle As IntPtr, ByVal dataArray As System.UInt32(), ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

#### C# .NET

Copies data into a two-dimensional array of double values:

```
public static MccDaq.ErrorInfo WinBufToArray32(IntPtr memHandle, double[,] dataArray, int firstPoint, int numPoints, int numChannels)
```

Copies data into an array of integer values:

```
public MccDaq.ErrorInfo WinBufToArray32(IntPtr memHandle, int[] dataArray, int firstPoint, int numPoints)
```

Copies data into an array of System.UInt32 values:

```
public MccDaq.ErrorInfo WinBufToArray32(int memHandle, uint[] dataArray, int firstPoint, int numPoints)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

#### VB .NET

Copies data into a two-dimensional array of double values:

```
Public Shared Function WinBufToArray32(ByVal memHandle As Integer, ByRef dataArray(,) As Double, ByVal firstPoint As Integer, ByVal numPoints As Integer, ByVal numChannels As Integer) As MccDaq.ErrorInfo
```

Copies data into an array of integer values:

```
Public Shared Function WinBufToArray32(ByVal memHandle As Integer, ByRef dataArray As Integer, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

Copies data into an array of System.UInt32 values:

```
Public Shared Function WinBufToArray32(ByVal memHandle As Integer, ByRef dataArray As System.UInt32, ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDaq.ErrorInfo
```

#### C# .NET

Copies data into a two-dimensional array of double values:

```
public MccDaq.ErrorInfo WinBufToArray32(int memHandle, out double[,] dataArray, int firstPoint, int numPoints, int numChannels)
```

Copies data into an array of integer values:

```
public MccDaq.ErrorInfo WinBufToArray32(int memHandle, out int dataArray, int firstPoint, int numPoints)
```

Copies data into an array of System.UInt32 values:

```
public MccDaq.ErrorInfo WinBufToArray32(int memHandle, out uint dataArray, int firstPoint, int numPoints)
```

## Parameters

### *memHandle*

The memory handle that was returned by [WinBufAlloc32\(\)](#) when the buffer was allocated. The buffer should contain the data that you want to copy.

### *dataArray*

The array where the data is copied.

### *firstPoint*

The index of the first point in the memory buffer that data is copied from.

### *numPoints*

The number of data points to copy.

### *numChannels*

The number of channels to copy into dataArray.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- You can copy only a portion of the buffer to the array using the firstPoint and numPoints argument. This is useful if you want foreground code to manipulate previously collected data while a Background scan continues to collect new data.
- Although this method is available to Windows C programs, it is not necessary, since you can manipulate the memory buffer directly by casting the MemHandle returned from WinBufAlloc32() to the appropriate type. This method avoids having to copy the data from the memory buffer to an array.

Refer to the following example:

```
/*declare and initialize the variables*/
long numPoints= 1000;
unsigned short *dataArray = NULL;
int MemHandle = 0;

/*allocate the buffer and cast it to a pointer to an unsigned long*/
MemHandle = WinBufAlloc32(numPoints);
dataArray = (unsigned long*)MemHandle;

/*scan in the data*/
CInScan(.....,MemHandle,...);

/*print the results*/
for(int i=0; i<numPoints; ++i)
    printf("Data[%d]=%d\n", i, dataArray[i]);

/*free the buffer and NULL the pointer*/
WinBufFree(MemHandle);
dataArray = NULL;
```

## ScaledWinArrayToBuf() method

Copies double-precision values from an array into a Windows memory buffer.

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Function ScaledWinArrayToBuf(ByVal dataArray As Double(), ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal numPoints As Integer) As Integer As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo ScaledWinArrayToBuf(double[] dataArray, IntPtr memHandle, int firstPoint, int numPoints)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Shared Function ScaledWinArrayToBuf(ByRef dataArray As Double, ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal numPoints As Integer) As Integer As MccDag.ErrorInfo
```

C# .NET

```
public static MccDag.ErrorInfo ScaledWinArrayToBuf(ref double dataArray, int memHandle, int firstPoint, int numPoints)
```

### Parameters

*dataArray*

The array containing the data to be copied.

*memHandle*

This must be a memory handle that was returned by [ScaledWinBufAllocEx\(\)](#) when the buffer was allocated. The data will be copied into this buffer.

*firstPoint*

Index of the first point in the memory buffer where the data will be copied.

*numPoints*

Number of data points to copy.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

- This method is used in conjunction with the ScaleData scan option and [ScaledWinBufAllocEx\(\)](#).

## ScaledWinBufAlloc() method (deprecated)

Allocates a Windows global memory buffer large enough to hold scaled data obtained from scan operations in which the ScaleData scan option is selected, and returns a memory handle for the buffer.

**Deprecated**; unless your application calls deprecated methods, use [ScaledWinBufAllocEx\(\)](#).

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Shared Function ScaledWinBufAlloc(ByVal numPoints As Integer) As Integer
```

C# .NET

```
public static int ScaledWinBufAlloc(int numPoints)
```

### Parameters

*numPoints*

The size of the buffer to allocate. Specifies the number of double precision values (8-byte or 64-bit) that the buffer will hold.

### Returns

- 0, if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- This method is used in conjunction with the ScaleData scan option and [ScaledWinBufToArray\(\)](#) or [ScaledWinArrayToBuf\(\)](#).
- Unlike most other methods in the library, this method does not return an [ErrorInfo object](#). It returns a Windows global memory handle, which can then be passed to the scan methods in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.

## ScaledWinBufAllocEx() method

Allocates a Windows global memory buffer large enough to hold scaled data obtained from scan operations in which the ScaleData scan option is selected, and returns a memory handle for the buffer.

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Function ScaledWinBufAllocEx(ByVal numPoints As Integer) As IntPtr
```

C# .NET

```
public IntPtr ScaledWinBufAllocEx(int numPoints)
```

### Parameters

*numPoints*

The size of the buffer to allocate. Specifies the number of double precision values (8-byte or 64-bit) that the buffer will hold.

### Returns

- 0, if the buffer could not be allocated, or a non-zero integer handle to the buffer.

### Notes

- This method is used in conjunction with the ScaleData scan option and [ScaledWinBufToArray\(\)](#) or [ScaledWinArrayToBuf\(\)](#).
- Unlike most other methods in the library, this method does not return an [ErrorInfo object](#). It returns a Windows global memory handle, which can then be passed to the scan methods in the library. If an error occurs, the handle will come back as 0 to indicate that the buffer was not allocated.
- This method is preferred over the deprecated method [ScaledWinBufAlloc\(\)](#). Only use ScaledWinBufAlloc() in 32-bit legacy applications that call deprecated methods.

## ScaledWinBufToArray() method

Copies double-precision values from a Windows memory buffer into an array.

Member of the [MccService class](#).

### Function Prototype

VB .NET

```
Public Shared Function ScaledWinBufToArray(ByVal memHandle As IntPtr, ByVal dataArray As Double(),  
ByVal firstPoint As Integer, ByVal numPoints As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo ScaledWinBufToArray(IntPtr memHandle, double[] dataArray, int firstPoint, int  
numPoints)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

```
Public Shared Function ScaledWinBufToArray(ByVal memHandle As Integer, ByRef dataArray As Double, ByVal  
firstPoint As Integer, ByVal numPoints As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public static MccDag.ErrorInfo ScaledWinBufToArray(int memHandle, out double dataArray, int firstPoint,  
int numPoints)
```

### Parameters

*memHandle*

The memory handle that was returned by [ScaledWinBufAllocEx\(\)](#) when the buffer was allocated. The buffer should contain the data that you want to copy.

*dataArray*

A pointer to the start of the destination array to which the data samples are copied.

*firstPoint*

The index of the first sample to copy from the buffer.

*numPoints*

The number of samples to copy into dataArray.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

- This method is used in conjunction with the ScaleData scan option and [ScaledWinBufAllocEx\(\)](#).



## BoardName property

The name of the board associated with an instance of the MccBoard class. The board name is returned as a null-terminated string.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public ReadOnly Property BoardName As String
```

C# .NET

```
public string BoardName [get]
```

## DeviceLogin() method

Opens a device session with a shared device.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function DeviceLogin(ByVal userName As String, ByVal password As String) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DeviceLogin(System.String userName, System.String password)
```

## Parameters

*userName*

A null-terminated string that identifies the user name used to log in to a device session.

*password*

A null-terminated string that identifies the password used to log in to a device session.

## Returns

- [Error code](#) or 0 if no errors.

## Notes

- If the user name or password is invalid, the INVALIDLOGIN error is returned.
- If the session is already opened by another user, the SESSIONINUSE error is returned.

## DeviceLogout() method

Releases the device session with a shared device.

Member of the [MccBoard class](#).

### Function Prototype

VB .NET

```
Public Function DeviceLogout\(\) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DeviceLogout\(\)
```

## DisableEvent() method

Disables one or more event conditions, and disconnects their user-defined handlers.

Member of the [MccBoard class](#).

### Function Prototype

VB .NET

```
Public Function DisableEvent (ByVal eventType As MccDag.EventType As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo DisableEvent (MccDag.EventType eventType)
```

### Parameters

*eventType*

Specifies one or more event conditions that will be disabled. More than one event type can be specified by bitwise OR'ing the event types. Note that specifying an event that has not been enabled is benign and will not cause any errors. Refer to [EnableEvent\(\)](#) for a list of valid Event Types.

To disable all events in a single call, use AllEventTypes.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### Notes

- For most event types, this method cannot be called while any background operations ([AInScan\(\)](#), [APretrig\(\)](#), or [AOutScan\(\)](#)) are active. Perform a [StopBackground\(\)](#) before calling DisableEvent(). However, for OnExternalInterrupt events, you can call DisableEvent() while the board is actively generating events.

### Important!

In order to understand the methods, you must read the board-specific information contained in the *Universal Library User's Guide*. Review and run the example programs before attempting any programming of your own. Following this advice will save you hours of frustration, and possibly time wasted holding for technical support.

This note, which appears elsewhere, is especially applicable to this method. Now is the time to read the board-specific information for your board (see the *Universal Library User's Guide*). We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.

## EnableEvent() method

Binds one or more event conditions to a user-defined callback function. Upon detection of an event condition, the user-defined function is invoked with board- and event-specific data. Detection of event conditions occurs in response to interrupts. Typically, this method is used in conjunction with interrupt driven processes such as [AInScan\(\)](#), [APretrig\(\)](#), or [AOut\(\)](#).

Member of the [MccBoard class](#).

## Function Prototype

### VB .NET

```
Public Function EnableEvent(ByVal eventType As MccDaq.EventType, ByVal eventParameter As Integer, ByVal callbackFunc As MccDaq.EventCallback, ByVal userData As IntPtr) As MccDaq.ErrorInfo

Public Function EnableEvent(ByVal eventType As MccDaq.EventType, ByVal eventParameter As System.UInt32, ByVal callbackFunc As MccDaq.EventCallback, ByVal userData As IntPtr) As MccDaq.ErrorInfo

Public Function EnableEvent(ByVal eventType As MccDaq.EventType, ByVal eventParameter As MccDaq.EventParameter, ByVal callbackFunc As MccDaq.CallbackFunction, ByVal userData As IntPtr) As MccDaq.ErrorInfo
```

### C# .NET

```
public MccDaq.ErrorInfo EnableEvent(MccDaq.EventType eventType, uint eventParameter, MccDaq.EventCallback callbackFunc, System.IntPtr userData)

public MccDaq.ErrorInfo EnableEvent(MccDaq.EventType eventType, int eventParameter, MccDaq.EventCallback callbackFunc, System.IntPtr userData)

public MccDaq.ErrorInfo EnableEvent(MccDaq.EventType eventType, MccDaq.EventParameter eventParameter, MccDaq.CallbackFunction callbackFunc, System.IntPtr userData)
```

## Parameters

### [eventType](#)

Specifies one or more event conditions that will be bound to the user-defined callback function. More than one event type can be specified by bitwise OR'ing the event types. Set it to one of the constants in the [eventType Parameter Values](#) section below.

### [eventParameter](#)

Additional data required to specify some event conditions, such as an OnDataAvailable event or OnExternalInterrupt event.

For OnDataAvailable events, eventParameter is used to determine the minimum number of samples to acquire during an analog input scan before generating the event. For OnExternalInterrupt events, eventParameter is used to latch digital bits on supported hardware by setting it to one of the constants in the [eventParameter parameter Values](#) section below.

Most event conditions ignore this value.

### [callbackFunc](#)

A delegate type that is the user-defined callback function to handle the above event type(s). A *delegate* is a data structure that refers either to a static method, or to a class instance and an instance method of that class.

The callbackFunc needs the same parameters as the [EventCallback delegate](#) declaration. Refer to the "EventCallback delegate" section for proper syntax and return values.

### [userData](#)

Reference to user-defined data that is passed to the EventCallback delegate. This parameter is NOT de-referenced by the library or its drivers; as a consequence, a NULL pointer can be supplied.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## eventType Parameter Values

OnDataAvailable	Generates an event whenever the number of samples acquired during an analog input scan increases by eventParameter samples or more. Note that for BlockIo scans, events will be generated on packet transfers; for example, even if EventParameter is set to 1, events will only be generated every packet-size worth of data (256 samples for the PCI-DAS1602) for aggregate rates greater than 1 kHz for the default <a href="#">AInScan()</a> mode.  For <a href="#">APretrig()</a> , the first event is not generated until a minimum of EventParameter samples after the pretrigger.
OnEndOfAiScan	Generates an event upon completion or fatal error of <a href="#">AInScan()</a> or <a href="#">APretrig()</a> .  Some devices, such as the USB-1208FS and USB-1408FS, will generate an end of scan event after <a href="#">StopBackground()</a> is called, but most devices do not. Handle post-scan tasks directly after calling StopBackground.
OnEndOfAoScan	Generates an event upon completion or fatal error of <a href="#">AOutScan()</a> .  Some devices, such as the USB-1208FS and USB-1408FS, will generate an end of scan event after <a href="#">StopBackground()</a> is called, but most devices do not. Handle post-scan tasks directly after calling StopBackground.
OnExternalInterrupt	For some digital and counter boards, generates an event, latches digital input data, or latches digital output data upon detection of a pulse at the External Interrupt pin.
OnPretrigger	For <a href="#">APretrig()</a> , generates an event upon detection of the first trigger.
OnScanError	Generates an event upon detection of a driver error during Background input and output scans. This includes OverRun, UnderRun, and TooFew errors.

## eventParameter Parameter Values

LatchDI	Returns the data that was latched in at the most recent interrupt edge.
LatchDO	Latches out the data most recently written to the hardware.

## Callback Function Prototypes

### C# .NET

```
public delegate void EventCallback( int BoardNum, MccDaq.EventType EventType, uint EventData, IntPtr pUserData);
```

### VB .NET

```
Public Sub MyCallback(ByVal BoardNum As Integer, ByVal EventType As MccDaq.EventType, ByVal EventData As UInt32, ByVal pUserData As System.IntPtr)
```

## Notes

- EnableEvent() cannot be called while any background operations ([AInScan\(\)](#), [APretrig\(\)](#), or [AOutScan\(\)](#)) are active. If a background operation is in progress when EnableEvent() is called, EnableEvent() will return the AlreadyActive error. Perform a [StopBackground\(\)](#) call before calling EnableEvent().
- Events will not be generated faster than the user callback function can handle them. If an event type becomes multi-signaled before the event handler returns, events are merged. The event handler is called once per event type, and is supplied with the event data corresponding to the latest event. When more than one event type is generated, the event handler for each event type is called in the same order in which they are enabled.
- Events are generated while handling board-generated interrupts. As a consequence, using [StopBackground\(\)](#) to abort background operations will *not* generate OnEndOfAoScan or OnEndOfAiScan events. However, the event handlers can be called directly immediately after calling [StopBackground\(\)](#).

## Important!

In order to understand the methods, you must read the board-specific information contained in the *Universal Library User's Guide*. Review and run the example programs before attempting any programming of your own. Following this advice will save you hours of frustration, and possibly time wasted holding for technical support.

This note, which appears elsewhere, is especially applicable to this method. Now is the time to read the board-specific information for your board (see the *Universal Library User's Guide*). We suggest that you make a copy of that page to refer to as you read this manual and examine the example programs.

## EngArrayToWinBuf() method

Transfers a 2D array of engineering unit values to a Windows buffer as integer values.

The conversion from engineering unit values to integer values uses the D/A resolution of the board associated with the MccBoard object.

This method is usually used to obtain values compatible with the [AOutScan\(\)](#) method or the [DagOutScan\(\)](#) method from a 2D array of engineering unit values, such as those provided by Measurement Studio signal generation methods. The converted values are transferred to the buffer based on the gain, firstPoint, count, and numChannels parameters.

Member of the [MccBoard](#) class.

## Function Prototype

### VB .NET

```
Public Function EngArrayToWinBuf(ByVal gain As MccDag.Range, ByVal engUnits As Double(), ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

```
Public Function EngArrayToWinBuf(ByVal gainArray As MccDag.Range(), ByVal gainCount As Integer, ByVal engUnits As Double(), ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

### C# .NET

```
public MccDag.ErrorInfo EngArrayToWinBuf(MccDag.Range Gain, double[,] EngUnits, IntPtr MemHandle, int FirstPoint, int Count, int NumChannels)
```

```
public MccDag.ErrorInfo EngArrayToWinBuf(MccDag.Range[] GainArray, int GainCount, double [,] EngUnits, IntPtr MemHandle, int FirstPoint, int Count, int NumChannels)
```

## Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

### VB .NET

```
Public Function EngArrayToWinBuf(ByVal gain As MccDag.Range, ByVal engUnits As Double(), ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

```
Public Function EngArrayToWinBuf(ByVal gainArray As MccDag.Range(), ByVal gainCount As Integer, ByVal engUnits As Double(), ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

### C# .NET

```
public MccDag.ErrorInfo EngArrayToWinBuf(MccDag.Range gain, double [,] engUnits, int memHandle, int firstPoint, int count, int numChannels)
```

```
public MccDag.ErrorInfo EngArrayToWinBuf(MccDag.Range[] gainArray, int gainCount, double [,] engUnits, int memHandle, int firstPoint, int count, int numChannels)
```

## Parameters

### *gain*

The range to use for converting the data. This range should be the same as the range specified for [AOutScan\(\)](#) or [DagOutScan\(\)](#).

### *gainArray*

The array containing the D/A range values used during the analog output scan.

If a gain queue was not used for the scan, this array should only contain 1 element whose value matches the gain used during the scan. If a gain queue was used during the scan, this array should match the gainArray value used in [DagOutScan\(\)](#).

If the corresponding range in the gainArray is set to NotUsed (MccDag.Range.NotUsed), engineering unit values are returned as integer values.

### *gainCount*

The number of array elements in gainArray. Set gainCount to 1 when no gain queue was used for the scan. If a gain queue was used for the scan, this number should match the number of gain queue pairs defined in [DagOutScan\(\)](#).

### *engUnits*

The array of data to convert to binary units and store in the windows memory buffer. With the engUnits array, the channel

numbers are stored in the first dimension, and the number of points/channel is stored in the second dimension.

*memHandle*

The handle to the windows memory buffer that holds the binary data that is output. This value should be large enough to hold (count x numChannels) samples.

*firstPoint*

The index into the windows memory buffer that will hold the first sample of the converted first channel. The index into the raw memory is (firstPoint x numChannels) so that converted data always starts with the first channel specified in the scan. For example, if firstPoint is 14 and the number of channels is 8, the index of the first converted sample is 112.

*count*

The number of samples per channel to convert from engineering units. count should not exceed Windows buffer size / (numChannels - firstPoint).

*numChannels*

The number of channels of data stored in the existing array to be transferred.



## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- This method stores the samples specified by firstPoint in the windows memory buffer. Each sample is converted using the ranges set by gain.
- If the corresponding range in the gainArray is set to NotUsed, engineering unit values are returned as integer values.

## EventCallback delegate

The EventCallback delegate is called as a parameter of the [EnableEvent\(\)](#) method. A delegate is a data structure that refers either to a static method, or to a class instance and an instance method of that class.

You create the data structure using the prototype shown below. You call the delegate by passing either its address or a pointer to the delegate to the callbackFunc parameter of the EnableEvent() method.

## Delegate Prototype

```
VB .NET
Public Sub MyCallback(ByVal BoardNum As Integer, ByVal EventType As MccDag.EventType, ByVal EventData
As UInt32, ByVal pUserData As System.IntPtr)

C# .NET
public delegate void EventCallback(int BoardNum, MccDag.EventType EventType, uint EventData, IntPtr
pUserData);
```

## Parameters

- BoardNum*  
Indicates which board caused the event.
- EventType*  
Indicates which event occurred.
- EventData*  
Board-specific data associated with this event. Returns the value of the EventType as listed in the "[EventData parameter values](#)" section below.
- pUserData*  
Pointer to or reference of data supplied by the userData parameter in the [EnableEvent\(\)](#) method. Note that before using this parameter value, it must be cast to the same data type as it was passed to EnableEvent().

## Returns

- pUserData – Returns the value specified by the userData parameter in [EnableEvent\(\)](#).

## EventData parameter values

EventType	Value of EventData
OnEndOfAiScan	The total number of samples acquired upon the scan completion or end.
OnEndOfAoScan	The total number of samples output upon the scan completion or end.
OnDataAvailable	The number of samples acquired since the start of the scan.
OnExternalInterrupt	The number of interrupts generated since enabling the OnExternalInterrupt event.
OnPretrigger	The number of pretrigger samples available at the time of pretrigger. Value is invalid for some boards when a <a href="#">TOOFEW</a> error occurs. See board details.
OnScanError	The <a href="#">Error code</a> of the scan error.

## FlashLED() method

Causes the LED on a USB device to flash.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function FlashLED\(\) As MccDag.ErrorInfo
```

C# .NET

```
public FlashLED\(\)
```

## Note

After calling FlashLED(), wait a few seconds before calling additional methods, or execution of the next method may fail.

## FromEngUnits() method

Converts a single precision voltage (or current) value in engineering units to an integer count value. This function is typically used to obtain a data value from a voltage value for output to a D/A with methods such as [AOut\(\)](#).

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function FromEngUnits(ByVal range As MccDaq.Range, ByVal engUnits As Single, ByRef dataVal As Short) As MccDaq.ErrorInfo
```

```
Public Function FromEngUnits(ByVal range As MccDaq.Range, ByVal engUnits As Single, ByRef dataVal As System.UInt16) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo FromEngUnits(MccDaq.Range range, float engUnits, out ushort dataVal)
```

```
public MccDaq.ErrorInfo FromEngUnits(MccDaq.Range range, float engUnits, out short dataVal)
```

## Parameters

*range*

The voltage (or current) range to use for the conversion to counts. When using this method to obtain a value to send to a D/A board, keep in mind that some D/A boards have programmable voltage ranges, and others set the voltage range via switches on the board. In either case, the desired range must be passed to this method. Refer to board-specific information for a list of [valid range settings](#).

*engUnits*

The single precision voltage (or current) value to use for the conversion to counts. Set the value to be within the range specified by the range parameter.

*dataVal*

Returns an integer count to this variable that is equivalent to the engUnits parameter using the resolution of the D/A on the board (if any).

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- dataVal – the integer count equivalent to engUnits is returned here.

## Notes

- This method is not supported for hardware with resolution greater than 16 bits.

The default resolution of this method is 12 bits, so if the device has neither analog input nor analog output, the result is a 12 bit conversion.

If the device has both analog input and analog output, the resolution and transfer function of the D/A converter on the device is used.

## GetBoardName() method

Returns the name of a specified board.

Member of the [MccService class](#).

## Function Prototype

VB .NET

```
Public Shared Function GetBoardName(ByVal boardNumber As Integer, ByRef boardName As String) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetBoardName(int boardNumber, ref string boardName)
```

## Parameters

*boardNumber*

Refers either to the board number associated with a board when it was installed, or GETFIRST or GETNEXT.

*boardName*

A string variable that contains the board name. Refer to the [Measurement Computing Device IDs](#) in the Universal Library User's Guide.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- boardName – return string containing the board name.

## Notes

There are two ways to use this method:

- Pass a board number as the boardNumber argument. The string that is returned describes the board type of the installed board.
- Set boardNumber to GETFIRST or GETNEXT to get a list of all board types that are supported by the library.

Set boardNumber to GETFIRST to get the first board type in the list of supported boards. Subsequent calls with Board=GETNEXT returns each of the other board types supported by the library. When you reach the end of the list, boardName is set to an empty string. The **ulgt04** example program in the installation directory demonstrates how to use this method.

## GetStatus() method

Returns the status about the background operation currently running.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function GetStatus(ByRef status As Short, ByRef curCount As Integer, ByRef curIndex As Integer,
ByVal functionType As MccDag.FunctionType) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo GetStatus(out short status, out int curCount, out int curIndex,
MccDag.FunctionType functionType)
```

## Parameters

*status*

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

*curCount*

The curCount parameter specifies how many points have been input or output since the Background process started. Use it to gauge how far along the operation is towards completion. Generally, curCount returns the total number of samples transferred between the DAQ board and the Windows data buffer at the time GetStatus() was called.

When you set both the Continuous and Background options, curCount's behavior depends on the board model. Refer to the board-specific information in the *Universal Library User's Guide* for the behavior of your board.

With recent MCC DAQ designs, the curCount parameter continually increases in increments of the packet size as Windows' circular data buffer recycles, until it reaches 231. Since the count parameter is a signed integer, at 2,147,483,647 + 1, the Count parameter rolls back to a negative number (-2,147,483,647). The count parameter resumes incrementing, eventually reaching 0 and increasing back up to 2,147,483,647.

The curIndex parameter is usually more useful than the curCount parameter in managing data collected when you set both the Continuous and Background options.

*curIndex*

The curIndex parameter is an index into the Windows data buffer. This index points to the start of the last completed channel scan that was transferred between the DAQ board and the Windows data buffer. If a scan is running but no points in the buffer have been transferred, curIndex equals -1 in most cases

For Continuous operations, curIndex rolls over when the Windows data buffer is full. This rollover indicates that "new" data is now overwriting "old" data. Your goal is to process the old data before it gets overwritten. You can keep ahead of the data flow by copying the old data out of the buffer before new data overwrites it.

The curIndex parameter can help you access the most recently transferred data. Your application does not have to process the data exactly when it becomes available in the buffer – in fact, you should avoid doing so unless absolutely necessary. The curIndex parameter generally increments by the packet size, but in some cases the curIndex parameter can vary within the same scan. One instance of a variable increment is when the packet size is not evenly divisible by the number of channels.

You should determine the best size of the "chunks" of data that your application can most efficiently process, and then periodically check on the curIndex parameter value to determine when that amount of additional data has been transferred.

Refer to the *Universal Library User's Guide* for information on your board, particularly when using Pre-Trigger.

*functionType*

Specifies which scan to retrieve status information about. Set it to one of the constants in the [functionType parameter values](#) section below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- Status – Returns the status of the operation:
  - 0 – a background process is not currently executing.
  - 1 – a background process is currently executing.
- curCount – The current number of samples collected.
- curIndex – The current sample index.

## [functionType](#) parameter values

AiFunction	Specifies analog input scans started with <a href="#">AInScan()</a> or <a href="#">APretrig()</a> .
------------	---

AoFunction	Specifies analog output scans started with <a href="#">AOutScan()</a> .
DiFunction	Specifies digital input scans started with <a href="#">DInScan()</a> .
DoFunction	Specifies digital output scans started with <a href="#">DOutScan()</a> .
CtrFunction	Specifies counter background operations started with <a href="#">CStoreOnInt()</a> or <a href="#">CInScan()</a> .
DaqiFunction	Specifies a synchronous input scan started with <a href="#">DagInScan()</a> .
DaqoFunction	Specifies a synchronous output scan started with <a href="#">DagOutScan()</a> .

## GetTCValues() method

Converts raw thermocouple data from a Windows global memory buffer collected using the [DagInScan\(\)](#) method to a one-dimensional or two dimensional array of data on a temperature scale (Celsius, Fahrenheit or Kelvin).

Member of the [MccBoard class](#).

### Function Prototype

VB .NET

```
Public Function GetTCValues(ByVal chanArray As Short(), ByVal chanTypeArray As MccDaq.ChannelType,  
ByVal chanCount As Integer, ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal count As  
Integer, ByVal scale As MccDaq.TempScale, ByVal tempValArray As Single()) As MccDaq.ErrorInfo  
  
Public Function GetTCValues(ByVal chanArray As Short(),ByVal chanTypeArray As MccDaq.ChannelType(),  
ByVal chanCount As Integer, ByVal memHandle As IntPtr, ByVal firstPoint As Integer, ByVal count As  
Integer, ByVal scale As MccDaq.TempScale, ByRef tempValArray As Double(,)) As MccDaq.ErrorInfo
```

C# .NET

```
public MccDaq.ErrorInfo GetTCValues(short[] chanArray, MccDaq.ChannelType[] chanTypeArray, int  
chanCount, IntPtr memHandle, int firstPoint, int count, TempScale scale, float[] tempValArray)  
  
public MccDaq.ErrorInfo GetTCValues(short[] chanArray, MccDaq.ChannelType[] chanTypeArray, int  
chanCount, IntPtr memHandle, int firstPoint, int count, TempScale scale, double[,] tempValArray)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

VB .NET

Copies data to a one-dimensional array of single values:

```
Public Function GetTCValues(ByVal chanArray As Short(), ByVal chanTypeArray As MccDaq.ChannelType,  
ByVal chanCount As Integer, ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal count As  
Integer, ByVal scale As MccDaq.TempScale, ByRef tempValArray As Single) As MccDaq.ErrorInfo
```

Copies data to a two-dimensional array of double values:

```
Public Function GetTCValues(ByVal chanArray As Short(),ByVal chanTypeArray As MccDaq.ChannelType(),  
ByVal chanCount As Integer, ByVal memHandle As Integer, ByVal firstPoint As Integer, ByVal count As  
Integer, ByVal scale As MccDaq.TempScale, ByRef tempValArray As Double(,)) As MccDaq.ErrorInfo
```

C# .NET

Copies data to a one-dimensional array of single values:

```
public MccDaq.ErrorInfo GetTCValues(short[] chanArray, MccDaq.ChannelType chanTypeArray, int chanCount,  
int memHandle, int firstPoint, int count, MccDaq.TempScale scale, out float tempValArray)
```

Copies data to a two-dimensional array of double values:

```
public MccDaq.ErrorInfo GetTCValues(short[] chanArray, MccDaq.ChannelType() chanTypeArray, int  
chanCount, int memHandle, int firstPoint, int count, MccDaq.TempScale scale, out double[,]  
tempValArray)
```

### Parameters

*chanArray*

Array containing channel values. Valid channel values are analog and temperature input channels and digital ports. chanArray must match the channel array used with the [DagInScan\(\)](#) method.

[chanTypeArray](#)

Array containing channel types. Each element of this array defines the type of the corresponding element in the chanArray. chanTypeArray must match the channel type settings used with the [DagInScan\(\)](#) method.

*chanCount*

Number of elements in chanArray.

*memHandle*

The memory handle that was returned (by [WinBufAlloc\(\)](#), [WinBufAlloc32\(\)](#), or [WinBufAlloc64\(\)](#)) when the buffer was allocated. The buffer should contain the data that you want to convert.

*firstPoint*

The index into the raw data memory buffer that holds the first sample of the first channel to be converted. The index into the



raw memory is (firstPoint x chanCount) so that converted data always starts with the first channel specified in the scan. For example, if firstPoint is 14 and the number of channels is 8, the index of the first converted sample is 112.

#### *count*

The number of samples per channel to convert to engineering units. count should not exceed Windows buffer size / chanCount – firstPoint.

#### *scale*

Specifies the temperature scale that the input will be converted to. Choices are MccDaq.TempScale.Celsius, MccDaq.TempScale.Fahrenheit, or MccDaq.TempScale.Kelvin.

#### *tempValArray*

The array to hold the converted data. This array must be allocated by the user, and must be large enough to hold count samples x the number of temperature channels.

## **Returns**

- An [ErrorInfo object](#) that indicates the status of the operation.
- tempValArray – Converted data.

## HideLoginDialog() method

Prevents the default login dialog from being shown when a protected function is called while not logged in.

Member of the [MccBoard class](#).

## Function Prototypes

VB .NET

```
Public Function HideLoginDialog(ByVal hide As Boolean) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo HideLoginDialog(System.Boolean hide)
```

## Parameters

*hide*

If true, the default dialog will not be shown when a protected function is called while the user is not logged in.

## Returns

- [Error code](#) or 0 if no errors.

## Notes

- Overrides InstaCal's **Show Login Dialog** prompt setting.

## InByte() method

Reads a byte from a hardware register on a board.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function InByte(ByVal portNum As Integer) As Integer
```

C# .NET

```
public int InByte(int portNum)
```

## Parameters

*portNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this parameter to the offset for the desired register. This method takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

## Returns

- The current value of the specified register

## Notes

- InByte() is used to read 8 bit ports. [InWord\(\)](#) is used to read 16-bit ports.
- This method was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.

## InWord() method

Reads a word from a hardware register on a board.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function InWord(ByVal portNum As Integer) As Integer
```

C# .NET

```
public int InWord(int portNum)
```

## Parameters

*portNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this parameter to the offset for the desired register. This method takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

## Returns

- The current value of the specified register

## Notes

- [InByte\(\)](#) is used to read 8 bit ports. InWord() is used to read 16-bit ports.
- This method was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.

## OutByte() method

Writes a byte to a hardware register on a board.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function OutByte(ByVal portNum As Integer, ByVal portVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo OutByte(int portNum, int portVal)
```

## Arguments

*portNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this parameter to the offset for the desired register. This method takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

*portVal*

Value that is written to the register.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- OutByte() is used to write to 8 bit ports. [OutWord\(\)](#) is used to write to 16-bit ports.
- This method was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.

## OutWord() method

Writes a word to a hardware register on a board.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function OutWord(ByVal portNum As Integer, ByVal portVal As Integer) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo OutWord(int portNum, int portVal)
```

## Parameters

*portNum*

Register within the board. Boards are set to a particular base address. The registers on the boards are at addresses that are offsets from the base address of the board (BaseAdr + 0, BaseAdr + 2, etc).

Set this parameter to the offset for the desired register. This method takes care of adding the base address to the offset, so that the board's address can be changed without changing the code.

*PortVal*

Value that is written to the register.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- [OutByte\(\)](#) is used to write to 8 bit ports. [OutWord\(\)](#) is used to write to 16-bit ports.
- This method was designed for use with ISA bus boards. Use with PCI bus boards is not recommended.

## RS485() method

Sets the direction of RS-485 communications port buffers.

Member of the [MccBoard class](#).

## Function Prototype

VB .NET

```
Public Function RS485(ByVal transmit As MccDag.OptionState, ByVal receive As MccDag.OptionState) As MccDag.ErrorInfo
```

C# .NET

```
public MccDag.ErrorInfo RS485(MccDag.OptionState transmit, MccDag.OptionState receive)
```

## Parameters

*transmit*

Set to Enabled or Disabled. The transmit RS-485 line driver is turned on. Data written to the RS-485 UART chip is transmitted to the cable connected to that port.

*receive*

Set to Enabled or Disabled. The receive RS-485 buffer is turned on. Data present on the cable connected to the RS-485 port is received by the UART chip.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- You can simultaneously enable or disable the transmit and receive buffers. If both are enabled, data written to the port is also received by the port. For a complete discussion of RS485 network construction and communication, refer to the CIO-COM485 or PCM-COM485 hardware manual.

# StopBackground() method

Stops one or more subsystem background operations that are in progress for the specified board. Use this method to stop any method that is running in the background. This includes any method that was started with the Background option, as well as [CStoreOnInt\(\)](#) (which always runs in the background).

Execute StopBackground() after normal termination of all background methods to clear variables and flags.

Member of the [MccBoard class](#).

## Function Prototype

```
VB .NET
Public Function StopBackground(ByVal funcType As MccDag.FunctionType) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo StopBackground(MccDag.FunctionType funcType)
```

## Parameters

functionType  
Specifies which background operation to stop. Set it to one of the constants in the [functionType parameter values](#) below.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

### [functionType](#) parameter values

AiFunction	Specifies analog input scans started with <a href="#">AInScan()</a> or <a href="#">APretrig()</a> .
AoFunction	Specifies analog output scans started with <a href="#">AOutScan()</a> .
DiFunction	Specifies digital input scans started with <a href="#">DInScan()</a> .
DoFunction	Specifies digital output scans started with <a href="#">DOutScan()</a> .
CtrFunction	Specifies counter background operations started with <a href="#">CStoreOnInt()</a> or <a href="#">CInScan()</a> .
DaqiFunction	Specifies a synchronous input scan started with <a href="#">DagInScan()</a> .
DaqoFunction	Specifies a synchronous output scan started with <a href="#">DagOutScan()</a> .



# TEDSRead() method

Reads data from a TEDS sensor into an array.

Member of the [MccBoard class](#).

## Function Prototype

```
VB .NET
Public Function TEDSRead(ByVal chan As Integer, dataBuffer As Byte(), count As Integer, ByVal options
As MccDag.TEDSReadOptions) As MccDag.ErrorInfo

C# .NET
public MccDag.ErrorInfo TEDSRead(int chan, Byte[] dataBuffer, ref int count, MccDag.TEDSReadOptions
options)
```

## Parameters

- chan*  
A/D channel number.
- dataBuffer*  
Pointer to the data array.
- count*  
Number of data points to read.
- options*  
Reserved for future use.

## Returns

- [Error code](#) or 0 if no errors.

### options parameter values

Default	Reserved for future use.
---------	--------------------------

## ToEngUnits() method

Converts an integer count value to an equivalent single precision voltage (or current) value. This method is typically used to obtain a voltage value from data received from an A/D with methods such as [AIn\(\)](#).

Member of the [MccBoard class](#).

### Function Prototype

VB .NET

```
Public Function ToEngUnits(ByVal range As MccDag.Range, ByVal dataVal As Short, ByRef engUnits As Single) As MccDag.ErrorInfo

Public Function ToEngUnits(ByVal range As MccDag.Range, ByVal dataVal As System.UInt16, ByRef engUnits As Single) As MccDag.ErrorInfo
```

C# .NET

```
Public MccDag.ErrorInfo ToEngUnits(MccDag.Range range, ushort dataVal, out float engUnits)

Public MccDag.ErrorInfo ToEngUnits(MccDag.Range range, short dataVal, out float engUnits)
```

### Parameters

*range*

Voltage (or current) range to use for the conversion to engineering units. When using this method to obtain engineering units from a value received from an A/D board, keep in mind that some A/D boards have programmable voltage ranges, and others set the voltage range via switches on the board. In either case, the desired range must be passed to this method. Refer to board-specific information for a list of [valid range settings](#).

*dataVal*

An integer count value (typically, one returned from an A/D board).

*engUnits*

The single precision voltage (or current) value that is equivalent to dataVal is returned to this variable. The value will be within the range specified by the range parameter.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- engUnits – the engineering units value equivalent to dataVal is returned to this variable.

### Notes

- This method is not supported for hardware with resolution greater than 16 bits.

The default resolution of this method is 12 bits, so if the device has neither analog input nor analog output, the result will be a 12 bit conversion.

If the device has both analog input and analog output, the resolution and transfer function of the D/A converter on the device is used.

## ToEngUnits32() method

Converts an integer count value to an equivalent double precision voltage (or current) value. This method is typically used to obtain a voltage value from data received from an A/D with methods such as [AIn32\(\)](#).

Member of the [MccBoard class](#).

### Function Prototype

VB .NET

```
Public Function ToEngUnits32(ByVal range As MccDaq.Range, ByVal dataVal As Integer, ByRef engUnits As Double) As MccDaq.ErrorInfo

Public Function ToEngUnits32(ByVal range As MccDaq.Range, ByVal dataVal As UInteger, ByRef engUnits As Double) As MccDaq.ErrorInfo
```

C# .NET

```
Public MccDaq.ErrorInfo ToEngUnits32(MccDaq.Range range, uint dataVal, out double engUnits)

public MccDaq.ErrorInfo ToEngUnits32(MccDaq.Range range, int dataVal, out double engUnits)
```

### Parameters

*range*

Voltage (or current) range to use for the conversion to engineering units. When using this method to obtain engineering units from a value received from an A/D board, keep in mind that some A/D boards have programmable voltage ranges, and others set the voltage range via switches on the board. In either case, the desired range must be passed to this method. Refer to board-specific information for a list of [valid range settings](#).

*dataVal*

An integer count value (typically, one returned from an A/D board) to convert to engineering units.

*engUnits*

The double precision voltage (or current) value that is equivalent to *dataVal* is returned to this variable. The value will be within the range specified by the *range* parameter using the resolution of the A/D on the board.

### Returns

- An [ErrorInfo object](#) that indicates the status of the operation.
- engUnits – the engineering units value equivalent to dataVal is returned to this variable.

### Notes

- This method is typically used to obtain a voltage (or current) value from data received from an A/D with methods such as [AIn32\(\)](#).
- This method should be used for devices with a resolution of 20-bits or more.

The default resolution of this method is 32-bits, so if the device has neither analog input nor analog output, the result will be a 32-bit conversion.

If the device has both analog input and analog output, the resolution and transfer function of the D/A converter on the device is used.

## WinBufToEngArray() method

Transfers integer values from a Windows buffer to a 2D array as engineering unit values.

The conversion from integer values to engineering unit values uses the A/D resolution of the board associated with the MccBoard object.

This method is usually used to obtain values compatible with those required by Measurement Studio waveform display controls from a Windows buffer containing data from a method such as [AInScan\(\)](#) or [DagInScan\(\)](#).

The converted values are transferred to the 2D array based on the gain, firstPoint, count, and numChannels parameters.

Member of the [MccBoard](#) class.

### Function Prototype

#### VB .NET

```
Public Function WinBufToEngArray(ByVal gain As MccDag.Range, ByVal memHandle As IntPtr, ByVal engUnits As Double(), ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

```
Public Function WinBufToEngArray(ByVal gainArray As MccDag.Range(), ByVal gainCount As Integer, ByVal memHandle As IntPtr, ByVal engUnits As Double(), ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

#### C# .NET

```
public MccDag.ErrorInfo WinBufToEngArray(MccDag.Range Gain, IntPtr MemHandle, double[,] EngUnits, int FirstPoint, int Count, int NumChannels)
```

```
public MccDag.ErrorInfo WinBufToEngArray(MccDag.Range[] GainArray, int GainCount, IntPtr MemHandle, double[,] EngUnits, int FirstPoint, int Count, int NumChannels)
```

### Deprecated methods

The following methods are deprecated, and should only be used for legacy applications. The methods above are preferred, and must be used for 64-bit application development.

#### VB .NET

```
Public Function WinBufToEngArray(ByVal gain As MccDag.Range, ByVal memHandle As Integer, ByVal engUnits As Double(), ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

```
Public Function WinBufToEngArray(ByVal gainArray As MccDag.Range(), ByVal gainCount As Integer, ByVal memHandle As Integer, ByVal engUnits As Double(), ByVal firstPoint As Integer, ByVal count As Integer, ByVal numChannels As Integer) As MccDag.ErrorInfo
```

#### C# .NET

```
public MccDag.ErrorInfo WinBufToEngArray(MccDag.Range gain, int memHandle, double[,] engUnits, int firstPoint, int count, int numChannels)
```

```
public MccDag.ErrorInfo WinBufToEngArray(MccDag.Range gainArray, int gainCount, int memHandle, double[,] engUnits, int firstPoint, int count, int numChannels)
```

### Parameters

#### *gain*

The [range](#) to use for converting scan data. This value should be the same as the range specified for [AInScan\(\)](#) or [DagInScan\(\)](#).

#### *gainArray*

The array containing the A/D range values used during the analog input scan.

If a gain queue was not used for the scan, this array should only contain 1 element whose value matches the gain used during the scan. If a gain queue was used during the scan, this array should match the gainArray value used in [ALoadQueue\(\)](#) or [DagInScan\(\)](#).

If the corresponding range in the gainArray is set to *NotUsed* (MccDag.Range.NotUsed), raw data is returned in engineering unit values.

#### *gainCount*

The number of array elements in gainArray. Set gainCount to 1 when no gain queue was used for the scan. If a gain queue was used for the scan, this number should match the number of gain queue pairs defined in [ALoadQueue\(\)](#) or [DagInScan\(\)](#).

#### *memHandle*

The handle to the memory buffer holding the raw data to be converted to engineering units. This value should be large

enough to hold (count x numChannels) samples.

#### *engUnits*

The array to hold the converted data. This array must be allocated by the user, and must be large enough to hold count samples. The first dimension should be the number of channels. The second dimension should equal the number of points/channel.

#### *firstPoint*

The index into the raw data memory buffer that holds the first sample of the first channel to be converted. The index into the raw memory is (firstPoint × numChannels) so that converted data always starts with the first channel specified in the scan. For example, if firstPoint is 14 and the number of channels is 8, the index of the first converted sample is 112.

#### *count*

The number of samples per channel to convert to engineering units. count should not exceed Windows buffer size / (numChannels – firstPoint).

#### *numChannels*

The number of channels of data stored in the existing array to be transferred.

## Returns

- An [ErrorInfo object](#) that indicates the status of the operation.

## Notes

- If gainCount is greater than one, the conversions cycle through the array until count samples have been converted. When only one gain is specified, that gain is applied to all conversions. Data is returned in engineering unit values as a two-dimensional array.
- If the corresponding range in the gainArray is set to *NotUsed*, raw data is returned in engineering unit values.

## Universal Library example programs sorted by program name

The table below lists Universal Library example programs sorted by the program name. It includes the featured function calls, notes, and other functions included in the example program. All example programs include the [cbDeclareRevision\(\)](#) and [cbErrHandling\(\)](#) functions.

**Note:** The CWIN sample program directory contains programs A101, A102 and A103 only.

Example UL program	Featured function	Notes	Other functions included
CInScan01	<a href="#">cbCInScan()</a>	Scans a range of counter input channels, and writes the data to an array.  Board 0 must support counter scans.	<a href="#">cbWinBuffAlloc32()</a> <a href="#">cbWinBufToArray32()</a> <a href="#">cbWinBufFree()</a>
CInScan02	<a href="#">cbCInScan()</a> <a href="#">cbCCConfigScan()</a>	Scans a counter input channel in decrement mode, and writes the data to an array.  Board 0 must support counter scans.	<a href="#">cbWinBuffAlloc32()</a> <a href="#">cbWinBufToArray32()</a> <a href="#">cbWinBufFree()</a>
CInScan03	<a href="#">cbCInScan()</a> <a href="#">cbCCConfigScan()</a>	Scans a counter input in encoder mode, and writes the sample data to an array. This example displays counts from encoder as phase A, phase B, and totalizes the index on Z.  Board 0 must support counter scans in encoder mode.	<a href="#">cbWinBuffAlloc32()</a> <a href="#">cbWinBufToArray32()</a> <a href="#">cbWinBufFree()</a>
DaqInScan01	<a href="#">cbDaqInScan()</a>	Synchronously scans analog input channels, digital input ports, and counter input channels in the foreground.  Board 0 must support synchronous input.	<a href="#">cbDConfigPort()</a>  <a href="#">cbCCConfigScan()</a>
DaqInScan02	<a href="#">cbDaqInScan()</a>	Synchronously scans analog input channels, digital input ports, and counter input channels in the background.  Board 0 must support synchronous input.	<a href="#">cbDConfigPort()</a> <a href="#">cbCCConfigScan()</a> <a href="#">cbStopBackground()</a> <a href="#">cbGetStatus()</a>
DaqInScan03	<a href="#">cbDaqInScan()</a> <a href="#">cbGetTCValues()</a>	Synchronously scans analog input channels, digital input ports, and thermocouple input channels in the foreground.  Board 0 must support synchronous input.	<a href="#">cbDConfigPort()</a> <a href="#">cbCCConfigScan()</a>
DaqOutScan01	<a href="#">cbDaqOutScan()</a>	Synchronously writes to an analog output channel and a digital output port in the background.  Board 0 must support synchronous output.	<a href="#">cbDConfigPort()</a>
DaqSetSetpoint01	<a href="#">cbDaqSetSetpoints()</a>	Configures setpoints, adds the setpoint status to the scan list, and performs asynchronous reads of the setpoint status.  Board 0 must support cbDaqInScan().	<a href="#">cbDaqInScan()</a> <a href="#">cbDConfigPort()</a> <a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a>
DaqSetTrigger01	<a href="#">cbDaqSetTrigger()</a>	Configures start and stop triggers. These triggers are used to initiate and terminate A/D conversion using cbDaqInScan() with the EXTTRIGGER option selected.  Board 0 must support synchronous output.	<a href="#">cbDConfigPort()</a> <a href="#">cbStopBackground()</a> <a href="#">cbGetStatus()</a>
PulseOutStart01	<a href="#">cbPulseOutStart()</a> <a href="#">cbPulseOutStop()</a>	Sends a frequency output to an output timer channel. Board 0 must have a timer output.  You enter a frequency and a duty	

		cycle within the timer's range.	
TimerOutStart01	<a href="#">cbTimerOutStart()</a> <a href="#">cbTimerOutStop()</a>	Sends a frequency output to an output timer channel. Board 0 must have a timer output.  You enter a frequency within the timer's range.	
ULAI01	<a href="#">cbAIn()</a>		<a href="#">cbToEngUnits()</a>
ULAI02	<a href="#">cbAInScan()</a>	BACKGROUND mode	<a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI03	<a href="#">cbAInScan()</a>	BACKGROUND mode	<a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI04	<a href="#">cbAConvertData()</a>		<a href="#">cbAInScan()</a> <a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI05	<a href="#">cbAInScan()</a>	with manual data conversion	<a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI06	<a href="#">cbAInScan()</a>	CONTINUOUS, BACKGROUND mode	<a href="#">cbAConvertData()</a> <a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI07	<a href="#">cbATrig()</a>		<a href="#">cbFromEngUnits()</a>
ULAI08	<a href="#">cbAPretrig()</a>		<a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI09	<a href="#">cbAConvertPretrigData()</a>	BACKGROUND mode	<a href="#">cbAPretrig()</a> <a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI10	<a href="#">cbALoadQueue()</a>		<a href="#">cbAInScan()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI11	<a href="#">cbToEngUnits()</a>		<a href="#">cbAIn()</a>
ULAI12	<a href="#">cbAInScan()</a>	EXTCLOCK mode	<a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI13	<a href="#">cbAInScan()</a>	Various sampling mode options	<a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI14	<a href="#">cbSetTrigger()</a>	EXTTRIGGER mode	<a href="#">cbAInScan()</a>

			<a href="#">cbFromEngUnits()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAI15	<a href="#">cbAInScan()</a>	SCALEDATA mode Scans a range of A/D input channels, and stores the scaled data in an array.	<a href="#">cbScaledWinBufAlloc()</a> <a href="#">cbScaledWinBufToArray()</a> <a href="#">cbWinBufFree()</a>
ULAI001	<a href="#">cbAInScan()</a> <a href="#">cbAOutScan()</a>	concurrent analog input and output scans	<a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinArrayToBuf()</a> <a href="#">cbWinBufAlloc()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufToArray()</a>
ULAO01	<a href="#">cbAOut()</a>		<a href="#">cbFromEngUnits()</a>
ULAO02	<a href="#">cbAOutScan()</a>		<a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULAO03	<a href="#">cbAOut()</a> <a href="#">cbSetConfig()</a>	Demonstrates the difference between BIDACUPDATEMODE settings of UPDATEIMMEDIATE and UPDATEONCOMMAND.  Board 0 must support BIDACUPDATEMODE settings, such as the <a href="#">PCI-DAC6702</a> and <a href="#">PCI-DAC6703</a> .	<a href="#">cbFromEngUnits()</a>
ULAO04	<a href="#">cbAOutScan()</a>	SCALEDATA mode Synchronously writes to analog channels in the background.	
ULCT01	<a href="#">cbC8254Config()</a>		<a href="#">cbCLoad()</a> <a href="#">cbCIn()</a>
ULCT02	<a href="#">cbC9513Init()</a> <a href="#">cbC9513Config()</a>		<a href="#">cbCLoad()</a> <a href="#">cbCIn()</a>
ULCT03	<a href="#">cbCStoreOnInt()</a>		<a href="#">cbC9513Init()</a> <a href="#">cbC9513Config()</a> <a href="#">cbCLoad()</a> <a href="#">cbCIn()</a>
ULCT04	<a href="#">cbCFreqIn()</a>		<a href="#">cbC9513Init()</a>
ULCT05	<a href="#">cbC8536Init()</a> <a href="#">cbC8536Config()</a>		<a href="#">cbCLoad()</a> <a href="#">cbCIn()</a>
ULCT06	<a href="#">cbC7266Config()</a>		<a href="#">cbCLoad32()</a> <a href="#">cbCIn32()</a> <a href="#">cbCStatus()</a>
ULCT07	<a href="#">cbCLoad32()</a> <a href="#">cbCIn32()</a>	Board 0 must have an event counter, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .	
ULDI01	<a href="#">cbDIn()</a>		<a href="#">cbDConfigPort()</a>
ULDI02	<a href="#">cbDBitIn()</a>		<a href="#">cbDConfigPort()</a>
ULDI03	<a href="#">cbDInScan()</a>		<a href="#">cbDConfigPort()</a> <a href="#">cbGetStatus()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinBufToArray()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufAlloc()</a>
ULDI04	<a href="#">cbDIn()</a>	using the AUXPORT	
ULDI05	<a href="#">cbDBitIn()</a>	using the AUXPORT	
ULDI06	<a href="#">cbDConfigBit()</a>		<a href="#">cbDBitIn()</a>



ULDO01	<a href="#">cbDOut()</a>		<a href="#">cbDConfigPort()</a>
ULDO02	<a href="#">cbDBitOut()</a>		<a href="#">cbDOut()</a> <a href="#">cbDConfigPort()</a>
ULDO04	<a href="#">cbDOut()</a>	using the AUXPORT	
ULDO05	<a href="#">cbDBitOut()</a>	using the AUXPORT	<a href="#">cbDOut()</a>
ULEV01*	<a href="#">cbEnableEvent()</a> <a href="#">cbDisableEvent()</a>	using ON_EXTERNAL_INTERRUPT	<a href="#">cbDConfigPort()</a> <a href="#">cbDIn()</a>
ULEV02*	<a href="#">cbEnableEvent()</a> <a href="#">cbDisableEvent()</a>	using ON_DATA_AVAILABLE and ON_END_OF_AI_SCAN	<a href="#">cbAInScan()</a> <a href="#">cbStopBackground()</a> <a href="#">cbToEngUnits()</a> <a href="#">cbWinBufAlloc()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufToArray()</a>
ULEV03*	<a href="#">cbEnableEvent()</a> <a href="#">cbDisableEvent()</a>	using ON_PRETRIG and ON_END_OF_AI_SCAN	<a href="#">cbAPretrig()</a> <a href="#">cbAConvertPretrigData()</a> <a href="#">cbDConfigPort()</a> <a href="#">cbDOut()</a> <a href="#">cbStopBackground()</a> <a href="#">cbToEngUnits()</a> <a href="#">cbWinBufAlloc()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufToArray()</a>
ULEV04*	<a href="#">cbEnableEvent()</a> <a href="#">cbDisableEvent()</a>	using ON_END_OF_AO_SCAN	<a href="#">cbAOutScan()</a> <a href="#">cbDConfigPort()</a> <a href="#">cbDOut()</a> <a href="#">cbFromEngUnits()</a> <a href="#">cbStopBackground()</a> <a href="#">cbWinArrayToBuf()</a> <a href="#">cbWinBufAlloc()</a> <a href="#">cbWinBufFree()</a>
ULFI01	<a href="#">cbFileAInScan()</a>		<a href="#">cbFileGetInfo()</a>
ULFI02	<a href="#">cbFileRead()</a>		<a href="#">cbFileAInScan()</a> <a href="#">cbFileGetInfo()</a>
ULFI03	<a href="#">cbFilePretrig()</a>		<a href="#">cbFileGetInfo()</a> <a href="#">cbFileRead()</a>
ULFL01	<a href="#">cbFlashLED()</a>	Board 0 must have an external LED, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .	
ULGT01	<a href="#">cbGetErrMsg()</a>		<a href="#">cbAIn()</a>
ULGT03	<a href="#">cbGetConfig()</a>		<a href="#">cbGetBoardName()</a>
ULGT04	<a href="#">cbGetBoardName()</a>		<a href="#">cbGetConfig()</a>
ULLOG01	<a href="#">cbLogGetFileName()</a>	Retrieves the name of a binary log file.	
ULLOG02	<a href="#">cbLogGetFileInfo()</a> <a href="#">cbLogGetSampleInfo()</a> <a href="#">cbLogGetAIChannelCount()</a> <a href="#">cbLogGetCJCInfo()</a> <a href="#">cbLogGetDIOInfo()</a>	Retrieves information about the analog data, CJC data, and digital port data contained in a binary log file.	<a href="#">cbLogGetFileName()</a>
ULLOG03	<a href="#">cbLogReadAIChannels()</a> <a href="#">cbLogReadCJCChannels()</a> <a href="#">cbLogReadDIOChannels()</a> <a href="#">cbLogReadTimeTags()</a>	Retrieves the analog input data, CJC temperature data, digital I/O port data, date values, and time values logged in a binary file, and writes the data to separate arrays.	<a href="#">cbLogGetFileName()</a> <a href="#">cbLogGetSampleInfo()</a> <a href="#">cbLogGetAIChannelCount()</a> <a href="#">cbLogGetCJCInfo()</a> <a href="#">cbLogGetDIOInfo()</a> <a href="#">cbLogSetPreferences()</a>

ULLOG04	<a href="#">cbLogConvertFile()</a>	Converts a binary log file to a comma-separated values (.csv) text file or another text file format that you specify.	<a href="#">cbLogGetSampleInfo()</a>
ULMBDI01	<a href="#">cbDIn()</a>	Reads a digital input port on MetraBus card	
ULMBDI02	<a href="#">cbDBitIn()</a>	Reads the status of single digital input bit from MetraBus	
ULMBDO01	<a href="#">cbDOut()</a>	Writes a byte to digital output ports on MetraBus card	
ULMBDO02	<a href="#">cbDBitOut()</a>	Sets the state of a single digital output bit for MetraBus	
ULMM01	<a href="#">cbMemReadPretrig()</a>		<a href="#">cbAPretrig()</a>
ULMM02	<a href="#">cbMemRead()</a> <a href="#">cbMemWrite()</a>		
ULMM03	<a href="#">cbAInScan()</a>	with EXTMEMORY option	<a href="#">cbMemReset()</a> <a href="#">cbMemRead()</a>
ULTI01	<a href="#">cbTIn()</a>		<a href="#">cbGetConfig()</a>
ULTI02	<a href="#">cbTInScan()</a>		<a href="#">cbGetConfig()</a>
VIn01	<a href="#">cbVIn()</a>	Reads an A/D input channel.	
VOut01	<a href="#">cbVOut()</a>	Writes to a D/A output channel.	
* Example programs <i>ULEV01</i> , <i>ULEV02</i> , <i>ULEV03</i> and <i>ULEV04</i> are not available for the C Console.			

## Universal Library example programs sorted by function call

UL function name	Example program name	Special features/notes
<a href="#">cbAConvertData()</a>	ULAI04 ULAI06	
<a href="#">cbAConvertPreTrigData()</a>	ULAI09 ULEV03*	
<a href="#">cbACalibrateData()</a>	None	No example programs at this time
<a href="#">cbAIn()</a>	ULAI01 ULAI11 ULGT01	
<a href="#">cbAInScan()</a>	ULAI02 ULAI03 ULAI04 ULAI05 ULAI06 ULAI10 ULAI12 ULAI13 ULAI14 ULAI15 ULAI001 ULEV02* ULMM03	<ul style="list-style-type: none"> <li>■ FOREGROUND mode</li> <li>■ BACKGROUND mode with manual data conversion</li> <li>■ CONTINUOUS BACKGROUND mode</li> <li>■ EXTLOCK mode</li> <li>■ SCALEDATA mode (ULAI15)</li> <li>■ Various sampling mode options</li> </ul>
<a href="#">cbALoadQueue()</a>	ULAI10	
<a href="#">cbAOut()</a>	ULAO01 ULAO03	ULAO03 demonstrates difference between BIDACUPDATEMODE settings of UPDATEIMMEDIATE and UPDATEONCOMMAND. Board 0 must support BIDACUPDATEMODE settings, such as the <a href="#">PCI-DAC6702</a> and <a href="#">PCI-DAC6703</a> .
<a href="#">cbAOutScan()</a>	ULAO02 ULAO04 ULAI001 ULEV04*	<ul style="list-style-type: none"> <li>■ concurrent cbAInScan() and cbAOutScan()</li> <li>■ SCALEDATA mode (ULAO04)</li> </ul>
<a href="#">cbAPretrig()</a>	ULAI08 ULAI09 ULEV03* ULFI03 ULMM01	
<a href="#">cbATrig()</a>	ULAI07 ULMM01	
<a href="#">cbC7266Config()</a>	ULCT06	
<a href="#">cbC8254Config()</a>	ULCT01	
<a href="#">cbC8536Config()</a>	ULCT05	
<a href="#">cbC8536Init()</a>	ULCT05	
<a href="#">cbC9513Config()</a>	ULCT02 ULCT03	
<a href="#">cbC9513Init()</a>	ULCT02 ULCT03 ULCT04	
<a href="#">cbCConfigScan()</a>	CInScan02 CInScan03	Demonstrates how to scan a counter input channel in decrement mode, and then write the data to an array. Board 0 must support counter scans. For CInScan03, board 0 must support counter scans in encoder mode.
<a href="#">cbCFreqIn()</a>	ULCT04	
<a href="#">cbCIn()</a>	ULCT01 ULCT02 ULCT05	
<a href="#">cbCIn32()</a>	ULCT06 ULCT07	For ULCT07, board 0 must have an event counter, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .
<a href="#">cbCInScan()</a>	CInScan01 CInScan02 CInScan03	Demonstrates how to scan a range of counter channels and then write the data to an array. The board must support counter scans. For CInScan03, board 0 must support counter scans in encoder mode.
<a href="#">cbCLoad()</a>	ULCT01 ULCT02 ULCT03	

	ULCT05	
<a href="#">cbCLoad32()</a>	ULCT06 ULCT07	For ULCT07, board 0 must have an event counter, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .
<a href="#">cbCStoreOnInt()</a>	ULCT03	
<a href="#">cbCStatus()</a>	ULCT06	
<a href="#">cbDaqInScan()</a>	DaqInScan01 DaqInScan02 DaqInScan03 DaqOutScan01 DaqSetTrigger01	Demonstrates how to synchronously scan analog, counter, and thermocouple input channels, and digital input ports. Board 0 must support synchronous output.
<a href="#">cbDaqOutScan()</a>	DaqOutScan01	Demonstrates how to synchronously write to an analog output channel and digital output port in the background. Board 0 must support synchronous output.
<a href="#">cbDaqSetSetpoints()</a>	DaqSetSetpoints01	Demonstrates how to configure and use setpoints, including how to add the setpoint status to the scan list and perform asynchronous reads of the setpoint status.  Board 0 must support <a href="#">cbDaqInScan()</a> .
<a href="#">cbDaqSetTrigger()</a>	DaqSetTrigger01	Demonstrates how to set up start and stop trigger events. Board 0 must support synchronous output.
<a href="#">cbDBitIn()</a>	ULDI02 ULDI05 ULDI06 ULMBDI02	
<a href="#">cbDBitOut()</a>	ULDO02 ULDO05 ULMBDO02	
<a href="#">cbDConfigBit()</a>	ULDI06	
<a href="#">cbDConfigPort()</a>	ULDI01 ULDI02 ULDI03 ULDO01 ULDO02 ULDO05 ULEV01* ULEV03* ULEV04*	
<a href="#">cbDeclareRevision()</a>	All examples.	All example programs use this function
<a href="#">cbDIn()</a>	ULDI01 ULDI03 ULDI04 ULEV01* ULMBDI01	
<a href="#">cbDInScan()</a>	ULDI03	
<a href="#">cbDOut()</a>	ULDO01 ULDO02 ULDO04 ULDO05 ULEV03* ULEV04* ULMBDO01 ULMBDO02	
<a href="#">cbDOutScan()</a>	None	No example programs at this time
<a href="#">cbDisableEvent()</a> <a href="#">cbEnableEvent()</a>	ULEV01* ULEV02* ULEV03* ULEV04*	<ul style="list-style-type: none"> <li>■ ON_EXTERNAL_INTERRUPT</li> <li>■ ON_DATA_AVAILABLE</li> <li>■ ON_PRETRIGGER</li> <li>■ ON_END_OF_AO_SCAN</li> <li>■ ON_SCAN_ERROR</li> <li>■ ON_END_OF_AI_SCAN</li> </ul>
<a href="#">cbErrHandling()</a>	All examples	All example programs use this function
<a href="#">cbFileAInScan()</a>	ULFI01 ULFI02	Demonstrates how to scan one or more A/D channels and store the samples in a disk file.
<a href="#">cbFilePretrig()</a>	ULFI03	Demonstrates how to stream data continuously to a streamer file until a trigger is received, then continue data streaming until the total number of samples minus the number of pretrigger samples is reached.
<a href="#">cbFileRead()</a>	ULFI02 ULFI03	

<a href="#">cbFlashLED()</a>	ULFL01	Flashes the onboard LED for visual identification (board 0 must have an external LED, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> ).
<a href="#">cbFromEngUnits()</a>	ULAO01 ULAO03 ULAI07 ULAI14 ULEV04*	
<a href="#">cbGetBoardName()</a>	ULGT03 ULGT04	
<a href="#">cbGetConfig()</a>	ULGT03 ULGT04 ULTI01 ULTI02	
<a href="#">cbGetErrMsg()</a>	ULGT01	
<a href="#">cbGetRevision()</a>	None	No example programs at this time
<a href="#">cbGetStatus()</a>	ULAI03 ULAI04 ULAI05 ULAI06 ULAI09 ULAI001 ULCT03 ULDIO3	
<a href="#">cbGetTCValues()</a>	DaqInScan03	Demonstrates how to retrieve analog, thermocouple, and digital data from a synchronous scan operation. Board 0 must support synchronous input.
<a href="#">cbInByte()</a>	None	No example programs at this time
<a href="#">cbInWord()</a>	None	No example programs at this time
<a href="#">cbLogConvertFile()</a>	ULLOG04	Demonstrates how to convert a binary log file to a .CSV file.
<a href="#">cbLogGetAIChannelCount()</a>	ULLOG02 ULLOG03 ULLOG04	Demonstrates how to retrieve the number of analog channels contained in a binary log file.
<a href="#">cbLogGetAIInfo()</a>	ULLOG03	Demonstrates how to retrieve information about the analog input data contained in a binary log file and then write the data to an array.
<a href="#">cbLogGetCJCInfo()</a> <a href="#">cbLogGetDIOInfo()</a>	ULLOG02 ULLOG03	Demonstrates how to retrieve information about the CJC temperature data and digital I/O channel data contained in a binary log file.
<a href="#">cbLogGetFileInfo()</a>	ULLOG02	Demonstrates how to retrieve the version level and byte size of a binary log file.
<a href="#">cbLogGetFileName()</a>	ULLOG01 ULLOG02 ULLOG03	Demonstrates how to retrieve the name of a binary log file.
<a href="#">cbLogGetPreferences()</a>	None	No example programs at this time
<a href="#">cbLogGetSampleInfo()</a>	ULLOG02 ULLOG03 ULLOG04	Demonstrates how to retrieve the sample interval, sample count, and the date and time of the first data point logged in a binary file.
<a href="#">cbLogReadAIChannels()</a> <a href="#">cbLogReadCJCChannels()</a> <a href="#">cbLogReadDIOChannels()</a> <a href="#">cbLogReadTimeTags()</a>	ULLOG03	Demonstrates how to retrieve analog input data, CJC data, DIO channel data, and date/time values contained in a binary log file, and store the data in separate arrays.
<a href="#">cbLogSetPreferences()</a>	ULLOG03	Demonstrates how to stores preference settings for time stamp data, analog data, and CJC temperature data.
<a href="#">cbMemRead()</a>	ULMM01 ULMM02 ULMM03	
<a href="#">cbMemReadPretrig()</a>	ULMM01	
<a href="#">cbMemReset()</a>	ULMM03	
<a href="#">cbMemSetDTMode()</a>	None	No example programs at this time
<a href="#">cbMemWrite()</a>	ULMM02	
<a href="#">cbOutByte()</a>	None	No example programs at this time
<a href="#">cbOutWord()</a>	None	No example programs at this time
<a href="#">cbRS485()</a>	None	No example programs at this time
<a href="#">cbPulseOutStart()</a> <a href="#">cbPulseOutStop()</a>	PulseOutStart01	Demonstrates how to start and stop a timer square wave output. Board 0 must have a timer output. You enter a frequency and a duty cycle within the timer's range.
<a href="#">cbScaledWinArrayToBuf()</a>	None	No example programs at this time.

<a href="#">cbScaledWinBufAlloc()</a> <a href="#">cbScaledWinBufToArray()</a>	ULAI15	Demonstrates how to store scaled data in an array.
<a href="#">cbSetConfig()</a>	ULAO03	Demonstrates the difference between BIDACUPDATEMODE settings of UPDATEIMMEDIATE and UPDATEONCOMMAND. Board 0 must support BIDACUPDATEMODE settings, such as the <a href="#">PCI-DAC6702</a> and <a href="#">PCI-DAC6703</a> .
<a href="#">cbSetTrigger()</a>	ULAI14	
<a href="#">cbStopBackground()</a>	ULAI03 ULAI04 ULAI05 ULAI06 ULAI09 ULAI001 ULCT03 ULDIO3 ULEV02* ULEV03* ULEV04*	concurrent cbAInScan() and cbAOutScan()
<a href="#">cbTIn()</a>	ULTI01	
<a href="#">cbTInScan()</a>	ULTI02	
<a href="#">cbTimerOutStart()</a> <a href="#">cbTimerOutStop()</a>	TimerOutStart01	Demonstrates how to start and stop a timer square wave output. Board 0 must have a timer output. You enter a frequency within the timer's range.
<a href="#">cbToEngUnits()</a>	ULAI01 ULAI07 ULAI11 ULEV02* ULEV03*	
<a href="#">cbVIn()</a>	VIn01	Demonstrates how to read an A/D input channel.
<a href="#">cbVOut()</a>	VOut01	Demonstrates how to write to a D/A output channel.
<a href="#">cbWinBufAlloc()</a> <a href="#">cbWinBufFree()</a> <a href="#">cbWinBufToArray()</a>	ULAI02 ULAI03 ULAI04 ULAI05 ULAI06 ULAI08 ULAI09 ULAI10 ULAI12 ULAI13 ULAI14 ULAI15 ULAI01 ULAO02 ULCT03 ULDIO3 ULEV02* ULEV03* ULEV04*	ULEV04 features cbWinBufAlloc() and cbWinBufFree() only
<a href="#">cbWinArrayToBuf()</a>	ULAI001 ULAO02 ULEV04*	
<a href="#">cbWinBufAlloc32()</a>	CInScan01 CInScan02	Demonstrates how to allocate a Windows global memory buffer for use with 32-bit scan functions.
<a href="#">cbWinBufToArray32()</a>	CInScan01 CInScan02	Demonstrates how to copy 32-bit data from a Windows memory buffer into an array.
* Example programs ULEV01, ULEV02, ULEV03 and ULEV04 are not available for the C Console.		

## Universal Library for .NET example programs sorted by program name

The table below lists Universal Library for .NET example programs sorted by the program name. It includes the featured method calls, notes, and other methods included in the example program. All example programs include the [DeclareRevision\(\)](#) and [ErrHandling\(\)](#) methods.

Example UL for .NET program	Featured method	Notes	Other methods included
CInScan01	<a href="#">CInScan()</a>	Scans a range of counter input channels, and writes the data to an array. Board 0 must support counter scans.	<a href="#">WinBuffAlloc32()</a> <a href="#">WinBufToArray32()</a> <a href="#">WinBufFree()</a>
CInScan02	<a href="#">CInScan()</a> <a href="#">CConfigScan()</a>	Scans a counter input channel in decrement mode, and writes the data to an array. Board 0 must support counter scans.	<a href="#">WinBuffAlloc32()</a> <a href="#">WinBufToArray32()</a> <a href="#">WinBufFree()</a>
CInScan03	<a href="#">CInScan()</a> <a href="#">CConfigScan()</a>	Scans a counter input in encoder mode, and writes the sample data to an array. This example displays counts from encoder as phase A, phase B, and totalizes the index on Z. Board 0 must support counter scans in encoder mode.	<a href="#">WinBuffAlloc32()</a> <a href="#">WinBufToArray32()</a> <a href="#">WinBufFree()</a>
DaqInScan01	<a href="#">DaqInScan()</a>	Synchronously scans analog input channels, digital input ports and counter input channels in the foreground. Board 0 must support synchronous input.	<a href="#">DConfigPort()</a> <a href="#">CConfigScan()</a>
DaqInScan02	<a href="#">DaqInScan()</a>	Synchronously scans analog input channels, digital input ports, and counter input channels in the background. Board 0 must support synchronous input.	<a href="#">DConfigPort()</a> <a href="#">CConfigScan()</a> <a href="#">GetStatus()</a> <a href="#">StopBackground()</a>
DaqInScan03	<a href="#">DaqInScan()</a> <a href="#">GetTCValues()</a>	Synchronously scans analog input channels, digital input ports and thermocouple input channels in the foreground. Board 0 must support synchronous input.	<a href="#">DConfigPort()</a>
DaqOutScan01	<a href="#">DaqOutScan()</a>	Synchronously writes to an analog output channel and a digital output port in the background. Board 0 must support synchronous output.	<a href="#">DConfigPort()</a>
DaqSetSetpoints01	<a href="#">DaqSetSetpoints()</a>	Configures setpoints, adds the setpoint status to the scan list, and performs asynchronous reads of the setpoint status. Board 0 must support DaqInScan().	<a href="#">DaqInScan()</a> <a href="#">DConfigPort()</a> <a href="#">GetStatus()</a> <a href="#">StopBackground()</a>
DaqSetTrigger01	<a href="#">DaqSetTrigger()</a>	Configures start and stop triggers. These triggers are used to initiate and terminate A/D conversion using <a href="#">DaqInScan()</a> with the ExtTrigger option selected. Board 0 must support synchronous output.	<a href="#">DConfigPort()</a> <a href="#">GetStatus()</a> <a href="#">StopBackground()</a>
PulseOutStart01	<a href="#">PulseOutStart()</a> <a href="#">PulseOutStop()</a>	Sends a frequency output to an output timer channel. Board 0 must have a timer output. You enter a frequency and a duty cycle within the timer's range.	
TimerOutStart01	<a href="#">TimerOutStart()</a> <a href="#">TimerOutStop()</a>	Sends a frequency output to an output timer channel. Board 0 must have a timer output. You enter a frequency within the	

		timer's range.	
ULAI01	<a href="#">AIn()</a>		<a href="#">ToEngUnits()</a>
ULAI02	<a href="#">AInScan()</a>	Default mode	<a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI03	<a href="#">AInScan()</a>	Background mode	<a href="#">GetStatus()</a> <a href="#">StopBackground()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI04	<a href="#">AConvertData()</a>		<a href="#">AInScan()</a> <a href="#">GetStatus()</a> <a href="#">StopBackground()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI05	<a href="#">AInScan()</a>	With manual data conversion	<a href="#">GetStatus()</a> <a href="#">StopBackground()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI06	<a href="#">AInScan()</a>	Continuous Background mode	<a href="#">AConvertData()</a> <a href="#">GetStatus()</a> <a href="#">StopBackground()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI07	<a href="#">ATrig()</a>		<a href="#">FromEngUnits()</a>
ULAI08	<a href="#">APretrig()</a>		<a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI09	<a href="#">AConvertPretrigData()</a>	Background	<a href="#">APretrig()</a> <a href="#">GetStatus()</a> <a href="#">StopBackground()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI10	<a href="#">ALoadQueue()</a>		<a href="#">AInScan()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI11	<a href="#">ToEngUnits()</a>		<a href="#">AIn()</a>
ULAI12	<a href="#">AInScan()</a>	ExtClock mode	<a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI13	<a href="#">AInScan()</a>	Various sampling mode options	<a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAI14	<a href="#">SetTrigger()</a>	ExtTrigger mode	<a href="#">AInScan()</a> <a href="#">FromEngUnits()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>



ULAI15	<a href="#">AInScan()</a>	ScaleData mode Scans a range of A/D input channels, and stores the scaled data in an array.	<a href="#">ScaledWinBufAlloc()</a> <a href="#">ScaledWinBufToArray()</a> <a href="#">WinBufFree()</a>
ULAI001	<a href="#">AInScan()</a> <a href="#">AOutScan()</a>	Concurrent analog input and output scans	<a href="#">GetStatus()</a> <a href="#">StopBackground()</a> <a href="#">WinArrayToBuf()</a> <a href="#">WinBufAlloc()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufToArray()</a>
ULAO01	<a href="#">AOut()</a>		<a href="#">FromEngUnits()</a> <a href="#">AOut()</a>
ULAO02	<a href="#">AOutScan()</a>		<a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULAO03	<a href="#">AOut()</a> <a href="#">DacUpdate()</a> <a href="#">SetDACUpdateMode()</a>	Demonstrates difference between <a href="#">BoardConfig.DACUpdate.Immediate</a> and <a href="#">BoardConfig.DACUpdate.OnCommand</a> D/A update modes. Board 0 must support DAC update mode settings, such as the <a href="#">PCI-DAC6700s</a> .	<a href="#">FromEngUnits()</a>
ULAO04	<a href="#">AOutScan()</a>	ScaleData mode Synchronously writes to analog channels in the background.	
ULCT01	<a href="#">C8254Config()</a>		<a href="#">CLoad()</a> <a href="#">CIn()</a>
ULCT02	<a href="#">C9513Init()</a> <a href="#">C9513Config()</a>		<a href="#">CLoad()</a> <a href="#">CIn()</a>
ULCT03	<a href="#">CStoreOnInt()</a>		<a href="#">C9513Init()</a> <a href="#">C9513Config()</a> <a href="#">CLoad()</a> <a href="#">CIn()</a>
ULCT04	<a href="#">CFreqIn()</a>		<a href="#">C9513Init()</a>
ULCT05	<a href="#">C8536Init()</a> <a href="#">C8536Config()</a>		<a href="#">CLoad()</a> <a href="#">CIn()</a>
ULCT06	<a href="#">C7266Config()</a>		<a href="#">CLoad32()</a> <a href="#">CIn32()</a> <a href="#">CStatus()</a>
ULCT07	<a href="#">CLoad32()</a> <a href="#">CIn32()</a>	Board 0 must have an event counter, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .	
ULDI01	<a href="#">DIn()</a>		<a href="#">DConfigPort()</a>
ULDI02	<a href="#">DBitIn()</a>		<a href="#">DConfigPort()</a>
ULDI03	<a href="#">DInScan()</a>		<a href="#">DConfigPort()</a> <a href="#">GetStatus()</a> <a href="#">StopBackground()</a> <a href="#">WinBufToArray()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufAlloc()</a>
ULDI04	<a href="#">DIn()</a>	Using the AuxPort	
ULDI05	<a href="#">DBitIn()</a>	Using the AuxPort	
ULDI06	<a href="#">DConfigBit()</a>		<a href="#">DBitIn()</a>
ULDO01	<a href="#">DOut()</a>		<a href="#">DConfigPort()</a>
ULDO02	<a href="#">DBitOut()</a>		<a href="#">DOut()</a> <a href="#">DConfigPort()</a>
ULDO04	<a href="#">DOut()</a>	Using the AuxPort	

ULDO05	<a href="#">DBitOut()</a>	Using the AuxPort	<a href="#">DOut()</a>
ULEV01	<a href="#">EnableEvent()</a> <a href="#">DisableEvent()</a>	Using OnExternalInterrupt	<a href="#">DConfigPort()</a> <a href="#">DIn()</a>
ULEV02	<a href="#">EnableEvent()</a> <a href="#">DisableEvent()</a>	Using OnDataAvailable and OnEndOfAiScan	<a href="#">AInScan()</a> <a href="#">StopBackground()</a> <a href="#">ToEngUnits()</a> <a href="#">WinBufAlloc()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufToArray()</a>
ULEV03	<a href="#">EnableEvent()</a> <a href="#">DisableEvent()</a>	Using OnPretrig and OnEndOfAiScan	<a href="#">APretrig()</a> <a href="#">AConvertPretrigData()</a> <a href="#">DConfigPort()</a> <a href="#">DOut()</a> <a href="#">StopBackground()</a> <a href="#">ToEngUnits()</a> <a href="#">WinBufAlloc()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufToArray()</a>
ULEV04	<a href="#">EnableEvent()</a> <a href="#">DisableEvent()</a>	Using OnEndOfAoScan	<a href="#">AOutScan()</a> <a href="#">DConfigPort()</a> <a href="#">DOut()</a> <a href="#">FromEngUnits()</a> <a href="#">StopBackground()</a> <a href="#">WinArrayToBuf()</a> <a href="#">WinBufAlloc()</a> <a href="#">WinBufFree()</a>
ULFI01	<a href="#">FileAInScan()</a>		<a href="#">FileGetInfo()</a>
ULFI02	<a href="#">FileRead()</a>		<a href="#">FileAInScan()</a> <a href="#">FileGetInfo()</a>
ULFI03	<a href="#">FilePretrig()</a>		<a href="#">FileGetInfo()</a> <a href="#">FileRead()</a>
ULFL01	<a href="#">cbFlashLED()</a>	Board 0 must have an external LED, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .	
ULGT01	<a href="#">GetErrMsg()</a>		<a href="#">AIn()</a>
ULGT03	MccBoard class properties: <a href="#">BoardConfig</a> , <a href="#">DioConfig</a> , <a href="#">ExpansionConfig</a>	Use MccBoard class properties to get configuration information for a board.	<a href="#">GetBoardName()</a>
ULGT04	<a href="#">GetBoardName()</a>		<a href="#">MccDag.MccBoard.BoardName</a> property <a href="#">MccDag.GlobalConfig.NumBoards</a> property
ULLOG01	<a href="#">GetFileName()</a>	Retrieves the name of a binary log file.	
ULLOG02	<a href="#">GetFileInfo()</a> <a href="#">GetSampleInfo()</a> <a href="#">GetAIChannelCount()</a> <a href="#">GetCJCInfo()</a> <a href="#">GetDIOInfo()</a>	Retrieves analog data, CJC data, and digital port data contained in a binary log file.	<a href="#">GetFileName()</a>
ULLOG03	<a href="#">ReadAIChannels()</a> <a href="#">ReadCJCChannels()</a> <a href="#">ReadDIOChannels()</a> <a href="#">ReadTimeTags()</a>	Retrieves the analog input data, CJC temperature data, digital I/O port data, date values, and time values logged in a binary file, and writes the data to separate arrays.	<a href="#">GetFileName()</a> <a href="#">GetSampleInfo()</a> <a href="#">GetAIChannelCount()</a> <a href="#">GetCJCInfo()</a> <a href="#">GetDIOInfo()</a>

			<a href="#">SetPreferences()</a>
ULLOG04	<a href="#">ConvertFile()</a>	Converts a binary log file to a comma-separated values (.csv) text file or another text file format that you specify.	<a href="#">GetSampleInfo()</a>
ULMM01	<a href="#">MemReadPretrig()</a>		<a href="#">APretrig()</a>
ULMM02	<a href="#">MemRead()</a> <a href="#">MemWrite()</a>		
ULMM03	<a href="#">AInScan()</a>	With ExtMemory option	<a href="#">MemReset()</a> <a href="#">MemRead()</a>
ULTI01	<a href="#">TIn()</a>		
ULTI02	<a href="#">TInScan()</a>		
VIn01	<a href="#">VIn()</a>	Reads an A/D input channel.	
VOut01	<a href="#">VOut()</a>	Writes to a D/A output channel.	

## Universal Library for .NET example programs sorted by method call

UL for .NET method name	Example program name	Special features/notes
<a href="#">ACalibrateData()</a>	None	No example programs at this time
<a href="#">AConvertData()</a>	ULAI04 ULAI06	
<a href="#">AConvertPreTrigData()</a>	ULAI09 ULEV03	
<a href="#">AIn()</a>	ULAI01 ULAI11 ULGT01	
<a href="#">AInScan()</a>	ULAI02 ULAI03 ULAI04 ULAI05 ULAI06 ULAI10 ULAI12 ULAI13 ULAI14 ULAI15 ULEV02 ULMM03	<ul style="list-style-type: none"> <li>■ Default mode</li> <li>■ Background mode with manual data conversion</li> <li>■ Continuous Background mode</li> <li>■ ExtClock mode</li> <li>■ ScaleData mode (ULAI15)</li> <li>■ Various sampling mode options</li> </ul>
<a href="#">ALoadQueue()</a>	ULAI10	
<a href="#">AOut()</a>	ULAO01 ULAO03	Demonstrates difference between <a href="#">BoardConfig.DACUpdate.Immediate</a> and <a href="#">BoardConfig.DACUpdate.OnCommand</a> D/A update modes. Board 0 must support DAC update mode settings, such as the <a href="#">PCI-DAC6702</a> and <a href="#">PCI-DAC6703</a> .
<a href="#">AOutScan()</a>	ULAO02 ULAO04 ULAI001 ULEV04	<ul style="list-style-type: none"> <li>■ concurrent AInScan() and AOutScan()</li> <li>■ ScaleData mode (ULAO04)</li> </ul>
<a href="#">APretrig()</a>	ULAI08 ULAI09 ULEV03 ULFI03 ULMM01	
<a href="#">ATrig()</a>	ULAI07 ULMM01	
<a href="#">C7266Config()</a>	ULCT06	
<a href="#">C8254Config()</a>	ULCT01	
<a href="#">C8536Config()</a>	ULCT05	
<a href="#">C8536Init()</a>	ULCT05	
<a href="#">C9513Config()</a>	ULCT02 ULCT03	
<a href="#">C9513Init()</a>	ULCT02 ULCT03 ULCT04	
<a href="#">CConfigScan()</a>	CInScan02 CInScan03	Demonstrates how to scan a counter input channel in decrement mode, and then write the data to an array. Board 0 must support counter scans. For CInScan03, board 0 must support counter scans in encoder mode.
<a href="#">CFreqIn()</a>	ULCT04	
<a href="#">CIn()</a>	ULCT01 ULCT02 ULCT05	
<a href="#">CIn32()</a>	ULCT06 ULCT07	For ULCT07, board 0 must have an event counter, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .
<a href="#">CInScan()</a>	CInScan01 CInScan02 CInScan03	Demonstrates how to scan one or more counter input channels and then write the data to an array. Board 0 must support counter scans. For CInScan03, board 0 must support counter scans in encoder mode.
<a href="#">CLoad()</a>	ULCT01 ULCT02 ULCT03 ULCT05	

<a href="#">CLoad32()</a>	ULCT06 ULCT07	For ULCT07, board 0 must have an event counter, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> .
<a href="#">ConvertFile()</a>	ULLOG04	Demonstrates how to convert a binary log file to a .CSV or .TXT format.
<a href="#">CStatus()</a>	ULCT06	
<a href="#">CStoreOnInt()</a>	ULCT03	
<a href="#">DacUpdate()</a>	ALAO03	Demonstrates the difference between <a href="#">BoardConfig.DACUpdate.Immediate</a> and <a href="#">BoardConfig.DACUpdate.OnCommand</a> D/A update modes. Board 0 must support DAC update mode settings, such as the <a href="#">PCI-DAC6702</a> and <a href="#">PCI-DAC6703</a> .
<a href="#">DaqInScan()</a>	DaqInScan01 DaqInScan02 DaqInScan03	Demonstrates how to synchronously scan analog, counter, and thermocouple input channels, and digital input ports. Board 0 must support synchronous input.
<a href="#">DaqOutScan()</a>	DaqOutScan01	Demonstrates how to synchronously write to an analog output channel and digital output port in the background. Board 0 must support synchronous output.
<a href="#">DaqSetSetpoints()</a>	DaqSetSetpoints01	Demonstrates how to configure and use setpoints, including how to add the setpoint status to the scan list and perform asynchronous reads of the setpoint status. Board 0 must support DaqInScan().
<a href="#">DaqSetTrigger()</a>	DaqSetTrigger01	Demonstrates how to set up start and stop trigger events, and display input channel data.
<a href="#">DBitIn()</a>	ULDI02 ULDI05 ULDI06	
<a href="#">DBitOut()</a>	ULDO02 ULDO05	
<a href="#">DConfigBit()</a>	ULDI06	
<a href="#">DConfigPort()</a>	ULDI01 ULDI02 ULDI03 ULDO01 ULDO02 ULDO05 ULEV01 ULEV03 ULEV04	
<a href="#">DeclareRevision()</a>	All examples	All example programs use this method
<a href="#">DIn()</a>	ULDI01 ULDI03 ULDI04 ULEV01	
<a href="#">DInScan()</a>	ULDI03	
<a href="#">DOut()</a>	ULDO01 ULDO02 ULDO04 ULDO05 ULEV03 ULEV04	
<a href="#">DOutScan()</a>	None	No example programs at this time
<a href="#">EnableEvent()</a> <a href="#">DisableEvent()</a>	ULEV01 ULEV02 ULEV03 ULEV04	<ul style="list-style-type: none"> <li>■ OnExternalInterrupt</li> <li>■ OnDataAvailable</li> <li>■ OnPretrigger</li> <li>■ OnEndOfAoScan</li> <li>■ OnScanError</li> <li>■ OnEndOfAiScan</li> </ul>
<a href="#">ErrHandling()</a>	All examples	All example programs use this method
<a href="#">FileAInScan()</a>	ULFI01 ULFI02	Demonstrates how to scan one or more A/D channels and store the samples in a disk file.
<a href="#">FilePretrig()</a>	ULFI01 ULFI02 ULFI03	Demonstrates how to stream data continuously to a streamer file until a trigger is received, then continue data streaming until the total number of samples minus the number of pretrigger samples is reached.
<a href="#">FileRead()</a>	ULFI02 ULFI03	
<a href="#">FlashLED()</a>	ULFL01	Flashes the onboard LED for visual identification (board 0 must have an external LED, such as the <a href="#">miniLAB 1008</a> or <a href="#">USB-1208LS</a> ).

<a href="#">FromEngUnits()</a>	U LAO01 U LAO03 U LAI07 U LAI14 U LEV04	
<a href="#">GetAIInfo()</a>	ULLOG02	Demonstrates how to retrieve analog input data contained in a binary log file.
<a href="#">GetBoardName()</a>	ULGT03 ULGT04	
<a href="#">GetCJCInfo()</a>	ULLOG02 ULLOG03	Demonstrates how to retrieve CJC temperature data contained in a binary log file.
<a href="#">GetDACStartup()</a>	None	No example programs at this time
<a href="#">GetDACUpdateMode()</a>	None	No example programs at this time
<a href="#">GetDIOInfo()</a>	ULLOG02 ULLOG03	Demonstrates how to retrieve digital I/O port data contained in a binary log file.
<a href="#">GetErrMsg()</a>	ULGT01	
<a href="#">GetFileInfo()</a>	ULLOG02	Demonstrates how to retrieve the version level and byte size of a binary log file.
<a href="#">GetFileName()</a>	ULLOG01 ULLOG02 ULLOG03 ULLOG04	Demonstrates how to retrieve the name of a binary log file.
<a href="#">GetPreferences()</a>	None	No example programs at this time
<a href="#">GetRevision()</a>	None	No example programs at this time
<a href="#">GetSampleInfo()</a>	ULLOG02 ULLOG03	Demonstrates how to retrieve the sample interval, sample count, and the date and time of the first data point logged in a binary file.
<a href="#">GetStatus()</a>	U LAI03 U LAI04 U LAI05 U LAI06 U LAI09 U LAIO01 U LCT03 U LDI03	
<a href="#">GetTCValues()</a>	DaqInScan03	Demonstrates how to retrieve analog, thermocouple, and digital data from a synchronous scan operation. Board 0 must support synchronous input.
<a href="#">InByte()</a>	None	No example programs at this time
<a href="#">InWord()</a>	None	No example programs at this time
MccDaq.MccBoard class properties: <a href="#">BoardConfig</a> , <a href="#">DioConfig</a> , and <a href="#">ExpansionConfig</a>	ULGT03 ULGT04	Use the MccBoard class properties to get configuration information for a board.
<a href="#">MemRead()</a>	ULMM01 ULMM02 ULMM03	
<a href="#">MemReadPretrig()</a>	ULMM01	
<a href="#">MemReset()</a>	ULMM03	
<a href="#">MemSetDTMode()</a>	None	No example programs at this time
<a href="#">MemWrite()</a>	ULMM02	
<a href="#">OutByte()</a>	None	No example programs at this time
<a href="#">OutWord()</a>	None	No example programs at this time
<a href="#">ReadAICchannels()</a> <a href="#">ReadCJCChannels()</a> <a href="#">ReadDIOChannels()</a> <a href="#">ReadTimeTags()</a>	ULLOG03	Demonstrates how to retrieve analog input data, CJC data, DIO port data, and date/time values contained in a binary log file, and then store the data in separate arrays.
<a href="#">RS485()</a>	None	No example programs at this time
<a href="#">PulseOutStart()</a> <a href="#">PulseOutStop()</a>	PulseOutStart01	Demonstrates how to send a frequency output to a timer output channel. Board 0 must have a timer output.  You enter a frequency and a duty cycle within the timer's range.
<a href="#">ScaledWinArrayToBuf()</a>	None	No example programs at this time
<a href="#">ScaledWinBufAlloc()</a> <a href="#">ScaledWinBufToArray()</a>	U LAI15	Demonstrates how to store scaled data in an array.
<a href="#">SetDACStartup()</a>	None	No example programs at this time
<a href="#">SetDACUpdateMode()</a>	U LAO03	Demonstrates difference between <a href="#">BoardConfig.DACUpdate.Immediate</a> and

		BoardConfig.DACUpdate.OnCommand D/A update modes. Board 0 must support DAC update mode settings, such as the <a href="#">PCI-DAC6702</a> and <a href="#">PCI-DAC6703</a> .
<a href="#">SetPreferences()</a>	ULLOG03	Demonstrates how to store preference settings for time stamp data, analog data, and CJC temperature data.
<a href="#">SetTrigger()</a>	ULAI14	
<a href="#">StopBackground()</a>	ULAI03 ULAI04 ULAI05 ULAI06 ULAI09 ULAI001 ULCT03 ULDI03 ULEV02 ULEV03 ULEV04	Concurrent AInScan() and AOutScan()
<a href="#">TIn()</a>	ULTI01	
<a href="#">TInScan()</a>	ULTI02	
<a href="#">TimerOutStart()</a> <a href="#">TimerOutStop()</a>	TimerOutStart01	Demonstrates how to send a frequency output to a timer output channel. Board 0 must have a timer output. You enter a frequency within the timer's range.
<a href="#">ToEngUnits()</a>	ULAI01 ULAI07 ULAI11 ULEV02 ULEV03	
<a href="#">VIn()</a>	VIn01	Demonstrates how to read an A/D input channel.
<a href="#">VOut()</a>	VOut01	Demonstrates how to write to a D/A output channel.
<a href="#">WinArrayToBuf()</a>	ULAI001 ULAO02 ULEV04	
<a href="#">WinBufAlloc()</a> <a href="#">WinBufFree()</a> <a href="#">WinBufToArray()</a>	ULAI02 ULAI03 ULAI04 ULAI05 ULAI06 ULAI08 ULAI09 ULAI010 ULAI012 ULAI013 ULAI014 ULAI01 ULAO02 ULCT03 ULDI03 ULEV02 ULEV03 ULEV04 (WinBufAlloc() and WinBufFree() only)	
<a href="#">WinArrayToBuf()</a>	ULAI01 ULAI02 ULAI04	
<a href="#">WinArrayAlloc32()</a>	CInScan01 CInScan02	Demonstrates how to allocate a Windows global memory buffer for use with 32-bit scan functions.
<a href="#">WinBufToArray32()</a>	CInScan01 CInScan02	Demonstrates how to copy 32-bit data from a Windows global memory buffer into an array.

## Error Codes

Error codes that are returned when running Universal Library or Universal Library for .NET are listed below. Universal Library .NET errors can be referenced from the [MccDag.ErrorInfo.Message](#) property.

Each entry in the list has four parts:

- Error code number
- Symbolic name
- Error message
- Explanation of the error

Both the Universal Library function and its Universal Library .NET equivalent method are referred to when appropriate. Error code and error messages are identical for both programming libraries. The only difference in the error names used by each library is the case the Universal Library error names are all uppercase (for example *NOERRORS*), while the Universal Library for .NET error names are mixed case (for example *NoErrors*).

0	<b>NOERRORS</b>	No error has occurred
	The function executed successfully.	
1	<b>BADBOARD</b>	Invalid board number
	The BoardNum argument 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	<b>DEADDIGITALDEV</b>	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 board was installed.	
3	<b>DEADCOUNTERDEV</b>	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	<b>DEADDADEV</b>	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	<b>DEADADDEV</b>	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	<b>NOTDIGITALCONF</b>	Selected board does not have digital I/O.
	A digital I/O function or method 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	<b>NOTCOUNTERCONF</b>	Selected board does not have a counter.
	A counter function or method 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	<b>NOTDACONF</b>	Selected board does not have a D/A.
	An analog output function or method 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 the board number refers to.	
9	<b>NOTADCONF</b>	Selected board does not have an A/D.
	An analog input function or method 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	<b>NOTMUXCONF</b>	Selected board does not have thermocouple inputs.
	A thermocouple input function or method was called with a board number that does not support thermocouple inputs, or is not connected to an EXP board. Run the configuration program to view/change the board configuration.	
11	<b>BADPORTNUM</b>	Invalid digital port number.
	The Port number that was specified for a digital I/O function does not exist on the specified board.	
12	<b>BADCOUNTERDEVNUM</b>	Invalid counter device.
	The Counter Number that was specified for a counter function does not exist on the board that was specified.	
13	<b>BADDADEVNUM</b>	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	<b>BADSAMPLEMODE</b>	Invalid sample mode.
	A sample mode that is not supported on this board (SINGLEIO, DMAIO or BLOCKIO) was specified in the Options argument. Try running the function without setting any of the Sample Mode options.	
15	<b>BADINT</b>	Board configured for invalid interrupt level.
	No interrupt was selected in InstaCal and one is required, or the board is set for "compatible mode" and the interrupt level selected is not supported in this mode. Interrupts above 7 are not valid in compatible mode. Either change the switch setting on the board to "enhanced mode", or change the interrupt level with the configuration program to	



	something less than 8.	
16	<b>BADADCHAN</b>	Invalid A/D channel number.
	An invalid channel argument was passed to an analog input function or method. 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	<b>BADCOUNT</b>	Invalid count.
	An invalid Count argument was specified to a function or method. If this error occurs during cbAInScan()/AInScan(), increasing the Count should correct the problem. For boards using DMAIO, adjust the data buffer and Count above (HighChan-LowChan+1)*Rate/100 for CONTINUOUS mode scans. However, those boards using BLOCKIO, require a user buffer and Count large enough to hold at least one half FIFO worth of samples (typically, 512 samples) for CONTINUOUS mode scans.	
18	<b>BADCNTRCONFIG</b>	Invalid counter configuration specified.
	An invalid Config argument was passed to cbC8254Config/C8254Config. The only legal values are HIGHONLASTCOUNT, ONESHOT, RATEGENERATOR, SQUAREWAVE, SOFTWARESTROBE and HARDWARESTROBE.	
19	<b>BADDAVAL</b>	Invalid D/A value.
	An invalid D/A value was passed as an argument/parameter to an analog output function or method. The only legal values are 0 to 4,095 for 12-bit boards or 0 to 65,535 for 16-bit boards (see the " <a href="#">Visual Basic signed integers</a> " discussion at the beginning of the "Counter Boards" section in the <i>Universal Library User's Guide</i> ).	
20	<b>BADDACHAN</b>	Invalid D/A channel number.
	An invalid D/A channel was passed as an argument/parameter to an analog output function or method. 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	<b>ALREADYACTIVE</b>	Background operation already in progress.
	An attempt was made to start a second background process on the same board before the first one had completed. Background processes are started whenever the BACKGROUND option is used by cbCStoreOnInt()/CStoreOnInt(). To stop a background operation, call cbStopBackground()/StopBackground(). To wait for a background process to complete. To wait for a background process to complete call cbGetStatus()/GetStatus() and wait for status=IDLE.	
23	<b>PAGEOVERRUN</b>	DMA transfer crossed page boundary, may have gaps in data.
	When a DMA transfer crosses a 64K memory page boundary on boards without FIFO buffers, there may be a small gap (missing samples) in the data. For applications requiring high speed transfers of greater than 32K samples, please select a board with a FIFO buffer. For boards without, check the data for gaps and do not specify rates over that at which gapless data may be taken. This is system-specific, so you must determine the rate by experimentation.	
24	<b>BADRATE</b>	Invalid sampling rate.
	Invalid sampling rate argument was specified. The rate was either zero, a negative number or it was higher than the selected board supports. Refer to board-specific information for board maximum rates.	
25	<b>COMPATMODE</b>	Board switches set for Compatible mode.
	An operation was attempted that is not possible when the board's switch is "set for 'compatible' operation. The Most likely causes are due to 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	<b>TRIGSTATE</b>	Incorrect initial trigger state - trigger must start at TTL low.
	Boards that use "polled gate" triggering require that the trigger be "off" when a pre-trigger functions is first called. It then waits for the trigger signal. Make sure that the Trigger Input line (usually D0) is held at TTL low before calling the pre-trigger function.	
27	<b>ADSTATUSHUNG</b>	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	<b>TOOFEW</b>	Trigger occurred before the requested number of samples were collected.
	A pre-trigger function or method was called and the trigger signal occurred before the requested number of samples could be collected. This is only a warning message. The function or method continued anyway. The data that was returned to the array will contain fewer than the expected number of points. The function or method 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	<b>OVERRUN</b>	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 or methods, or by using SINGLIO mode. Possible solutions include lowering the sampling rate, defragmenting the "streamer" file, switching to a RAM disk, or lowering the count.	
30	<b>BADRANGE</b>	Invalid voltage or current range.
	Invalid Range argument was specified to an analog input or output function or method. The board does not support the gain you specified. Refer to board-specific information for a list of allowable ranges.	
31	<b>NOPROGGAIN</b>	This A/D board does not have programmable gain.
	Invalid Range argument was passed to an analog input function or method. The selected board does not support programmable gains so the only valid Range argument is 0. (This argument is ignored for these board types in later versions of the library.)	
32	<b>BADFILENAME</b>	Specified file name is not valid.

	The FileName argument/parameter that was passed to a file function is not valid. It is either an empty string or a NULL pointer.	
33	<b>DISKISFULL</b>	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()/FileAInScan() or cbFilePretrig()/FilePretrig() it indicates another problem. The disk space for these commands should have been previously allocated with the 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	<b>COMPATWARN</b>	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 sampling 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	<b>BADPOINTER</b>	Pointer is not valid.
	An invalid (NULL) pointer was passed as an argument/parameter to a function or method.	
37	<b>RATEWARNING</b>	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	<b>CONVERTDMA</b>	CONVERTDATA cannot be used with DMAIO 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()/AConvertData() to convert the data after it is collected. Don't use the BACKGROUND option. Use the BLOCKIO option if your A/D board supports it. Use the SINGLEIO option if your computer is fast enough to support the selected sampling rate.	
39	<b>DTCONNECTERR</b>	Board does not support the DTCONNECT option.
	The DTCONNECT option was passed to an analog input function or method. The selected board does not support that option.	
40	<b>FORECONTINUOUS</b>	CONTINUOUS can only be run with BACKGROUND.
	The CONTINUOUS option was passed to a function or method 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	<b>BADBOARDTYPE</b>	This function or method cannot be used with this board.
	An attempt was made to call a function or method for a board that does not support that function or method.	
42	<b>WRONGDIGCONFIG</b>	Digital port not configured correctly for requested operation.
	Some of the digital bits or ports (FIRSTPORTA - EIGHTHPORTCH) must be configured as inputs OR outputs but not both. An attempt was made to use a digital input function or method on a port or bit that was configured as an output or vice versa. Use cbDConfigPort()/DConfigPort() or cbDConfigBit()/DConfigBit() to switch a port's (or bits) direction. If the board you are using contains configurable port types and you do not call cbDConfigPort()/DConfigPort() or cbDConfigBit()/DConfigBit() in your program, then all of the configurable ports will be in an unknown state (input or output).	
43	<b>NOTCONFIGURABLE</b>	This digital port is not configurable (it's an In/Out port).
	cbDConfigPort()/DConfigPort() or cbDConfigBit()/DConfigBit() was called for a port that is not configurable. Check the PortNum argument passed to cbDConfigPort() and make sure that it is in the range FIRSTPORTA - EIGHTHPORTCH. If PortNum is AUXPORT, make sure your hardware supports configuration of this port type. If not then there is no need to call this function or method.	
44	<b>BADPORTCONFIG</b>	Invalid digital port configuration.
	The Direction argument passed to cbDConfigPort()/DConfigPort() or cbDConfigBit()/DConfigBit() is invalid. It must be set to either DIGITALIN or DIGITALOUT.	
45	<b>BADFIRSTPOINT</b>	First point number is not valid.
	The FirstPoint argument to cbFileRead()/FileRead() is invalid. It is either a negative number or it is larger then the number of points in the file.	
46	<b>ENDOFFILE</b>	Attempted to read past the end of the file.
	cbFileRead()/FileRead() attempted to read beyond the end of the file. Check the file length with cbFileGetInfo()/FileGetInfo() and make sure that the FirstPoint and Count arguments to cbFileRead()/FileRead() are correct for that file length.	
47	<b>NOT8254CTR</b>	This board does not have an 8254 counter.
	cbC8254Config()/C8254Config() was called for a board that has a counter but not an 8254 counter. This function or method can only be used with an 8254 counter.	
48	<b>NOT9513CTR</b>	This board does not have a 9513 counter.
	cbC9513Config()/C9513Config() was called for a board that has a counter but not a 9513 counter. This function or method can only be used with a 9513 counter.	
49	<b>BADTRIGTYPE</b>	Invalid TrigType.
	cbATrig()/ATrig() was called with an invalid TrigType argument. It must be set to either TRIGABOVE or TRIGBELOW.	
50	<b>BADTRIGVALUE</b>	Invalid TrigValue.
	cbATrig()/ATrig() was called with an invalid TrigValue argument/parameter. It must be in the range 0 - 4,095 for 12-bit	

	boards or 0 to 65,535 for 16-bit boards (see the " <a href="#">Visual Basic signed integers</a> " discussion at the beginning of the "Counter Boards" section in the <i>Universal Library User's Guide</i> .)	
52	<b>BADOPTION</b>	Invalid option specified for this function or method.
	The Option argument contains an option that is not valid for this function or method.	
53	<b>BADPRETRIGCOUNT</b>	Invalid PreTrigCount specified.
	Either cbAPretrig()/APretrig() or cbFilePretrig()/FilePretrig() was called with an invalid PretrigCount argument. The pre-trigger count must not be <0, and must be less than TotalCount - 512. It also must be less than 32k for cbAPretrig()/APretrig(), and less than 16k for cbFilePretrig()/FilePretrig().	
55	<b>BADDIVIDER</b>	Invalid FOutDivider value.
	The FOutDivider argument to cbC9513Init()/C9513Init() is not valid. It must be in the range 0 - 15.	
56	<b>BADSOURCE</b>	Invalid FOutSource value.
	The FOutSource argument to cbC9513Init() (C9513Init()) is not valid. It must be one of the following values: CTRINPUT1, CTRINPUT2, CTRINPUT3, CTRINPUT4, CTRINPUT5, GATE1, GATE2, GATE3, GATE4, GATE5, FREQ1, FREQ2, FREQ3, FREQ4, FREQ5 (for example 0 to 15).	
57	<b>BADCOMPARE</b>	Invalid compare value.
	One or both of the compare arguments to cbC9513Init()/C9513Init() are not valid. They must be set to (CB)ENABLED or (CB)DISABLED (1 or 0).	
58	<b>BADTIMEOFDAY</b>	Invalid TimeOfDay value.
	The TimeOfDay argument to cbC9513Init()/C9513Init() is not valid. It must be set to either ENABLED or DISABLED (1 or 0).	
59	<b>BADGATEINTERVAL</b>	Invalid Gate Interval value.
	The GateInterval argument to cbCFreqIn()/CFreqIn() is not valid. It must be greater than 0.	
60	<b>BADGATECNTRL</b>	Invalid Gate Control value.
	The GateControl argument to cbC9513Config()/C9513Config() is not valid. It must be in the range 0 -7.	
61	<b>BADCOUNTEREDGE</b>	Invalid Counter Edge value.
	The CounterEdge argument to cbC9513Config()/C9513Config() is not valid. It must be set to either POSITIVEEDGE or NEGATIVEEDGE.	
62	<b>BADSPCLGATE</b>	Invalid SpecialGate value.
	The SpecialGate argument to cbC9513Config()/C9513Config() is not valid. It must be set to either (CB)ENABLED or (CB)DISABLED (1 or 0).	
63	<b>BADRELOAD</b>	Invalid Reload value.
	The Reload argument to cbC9513Config()/C9513Config() is not valid. It must be set to either LOADREG or LOADANDHOLDREG.	
64	<b>BADRECYCLEFLAG</b>	Invalid Recycle Mode value.
	The RecycleMode argument to cbC9513Config()/C9513Config() is not valid. It must be set to either (CB)ENABLED or (CB)DISABLED (1 or 0).	
65	<b>BADBCDFLAG</b>	Invalid BCD Mode value.
	The BCDMode argument/parameter to cbC9513Config()/C9513Config() is not valid. It must be set to either (CB)ENABLED or (CB)DISABLED (1 or 0).	
66	<b>BADDIRECTION</b>	Invalid Count Direction value.
	The CountDirection argument to cbC9513Config() (C9513Config()) is not valid. It must be set to either COUNTUP or COUNTDOWN.	
67	<b>BADOUTCONTROL</b>	Invalid Output Control value.
	The OutputControl argument to cbC9513Config() (C9513Config()) is not valid. It must be set to either ALWAYSLOW, HIGHPULSEONTC, TOGGLEONTC, DISCONNECTED or LOWPULSEONTC.	
68	<b>BADBITNUMBER</b>	Invalid BitNum specified.
	The BitNum argument to cbDBitIn() or cbDBitOut() (DBitIn() or DBitOut()) 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. Refer to board-specific information in the <i>Universal Library User's Guide</i> or in your hardware manual.	
69	<b>NONEENABLED</b>	None of the counter channels were enabled.
	None of the counter channels were marked as (CB)ENABLED in the CntrControl array that was passed to cbCStoreOnInt()/CStoreOnInt(). At least one of the counter channels must be enabled.	
70	<b>BADCTRCONTROL</b>	An element of Cntr Control array not set to ENABLED or DISABLED
	One of the elements of the CntrControl array that was passed to cbCStoreOnInt()/CStoreOnInt() was set to something other than (CB)ENABLED or (CB)DISABLED. The array must have at least ten elements, and the first ten elements must be set to either (CB)ENABLED or (CB)DISABLED.	
71	<b>BADEXPCHAN</b>	Invalid EXP channel specified.
	An invalid channel was passed to one of the thermocouple input commands. The channel number when using an EXP board must be ≥16. The maximum allowable channel number depends on which EXP board is being used (and how many of them). Refer to the board manual to find the number of channels.	
72	<b>WRONGADRANGE</b>	Board set to wrong A/D range for reading thermocouples.

	A thermocouple input function or method was called to read an EXP board input. The EXP board is connected to an A/D board with hardware selected gain that is set to the wrong range. When using EXP boards with thermocouples, the A/D must be set to the -5 to +5 volt range when available. When using RTD sensors, the range is 0 to 10V when available.	
73	<b>OUTOFRANGE</b>	Temperature input is out of range.
	A thermocouple input function or method returned an invalid temperature. This usually indicates an open connection in the thermocouple or its connection to the mux board.	
74	<b>BADTEMPSCALE</b>	Invalid temperature scale specified
	<p>The Scale argument to a thermocouple input function or method is not valid. It must be set to either CELSIUS, FAHRENHEIT, KELVIN, VOLTS or NOSCALE.</p> <p>Set to VOLTS to read the voltage input of a thermocouple. Refer to board-specific information in the <i>Universal Library User's Guide</i> to determine if the hardware supports this option.</p> <p>Set to NOSCALE to retrieve raw data from a device. Specifying NOSCALE returns calibrated data, however a cold junction compensation (CJC) correction factor is not applied to the returned values. Refer to board-specific information in the <i>Universal Library User's Guide</i> to determine if the hardware supports this option.</p>	
76	<b>NOQUEUE</b>	Specified board does not have channel/gain queue.
	The function or method that was called requires that the board has a channel/gain queue. The specified board does not have a queue.	
77	<b>CONTINUOSCOUN</b>	Count must be > the packet size to use continuous mode.
	The Count argument is not valid for continuous mode. Using BLOCKIO mode, the Count argument must be large enough to cause at least one interrupt. This is usually half the size of the boards FIFO (typical sizes are 256, 512, and 1,024). Refer to the board-specific information in the <i>Universal Library User's Guide</i> .	
78	<b>UNDERRUN</b>	D/A FIFO went empty during output.
	The specified D/A output rate could not be sustained. Try increasing the size of the data buffer or reducing the update rate to eliminate the error.	
79	<b>BADMEMMODE</b>	Invalid memory mode specified.
	The memory mode that was selected with cbMemSetDTMode()/MemSetDTMode() is not one of the valid modes.	
80	<b>FREQOVERRUN</b>	Measured frequency too high for selected gating interval.
	The GateInterval argument used with cbCFreqIn()/CFreqIn() is too large to measure the frequency of the signal connected to the counter. The counter is overflowing. Decrease the gating interval to eliminate the error.	
81	<b>NOCJCCHAN</b>	A CJC Channel must be configured to make temperature measurements.
	When the board was installed with the InstaCal installation program, no Cold Junction Compression (CJC) channel was selected. To use the temperature measurement functions or methods with thermocouples, you must first select a CJC channel on the A/D board and then rerun the InstaCal installation program.	
82	<b>BADCHIPNUM</b>	Invalid ChipNum specified.
	An invalid ChipNum argument was used with cbC9513Init()/C9513Init(). If the board is CTR05, set ChipNum to 0. If the board is a CTR10, set ChipNum to either 0 or 1.	
83	<b>DIGNOTENABLED</b>	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 InstaCal installation program and set the digital I/O to ENABLED.	
84	<b>CONVERT16BITS</b>	CONVERTDATA 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 CONVERTDATA option with cbAInScan()/AInScan() or call cbAConvertData()/AConvertData(), this error is returned. (This has been updated in later versions of the library so that it is ignored for boards for which it is inappropriate.)	
85	<b>NOMEMBOARD</b>	The EXTMEMORY option requires that a MEGA-FIFO be attached.
	Attempt to use a cbMem_() function or Mem_() method without a MEGA-FIFO board installed. Install MEGA-FIFO with InstaCal.	
86	<b>DTACTIVE</b>	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	<b>NOTMEMCONF</b>	Specified board is not a memory board.
	The specified board is not a memory board. This function or method only works with memory boards.	
88	<b>ODDCHAN</b>	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()/AInScan(). On those boards the low channel must be an even number.	
89	<b>CTRNOINIT</b>	Counter was not configured or initialized.
	You attempted to use cbCLoad() or cbCIn() (CLoad() or CIn()) before initializing and configuring the counter.	
90	<b>NOT8536CTR</b>	This board does not have an 8536 counter chip.
	An attempt has been made to use 8536 initialization or configuration on board without an 8536 chip.	

91	<b>FREERUNNING</b>	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 cbAInScan()/AInScan(). 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	<b>INTERRUPTED</b>	Operation interrupted with Ctrl-C key.
	A foreground operation was stopped before completion because either the Ctrl-C or Ctrl-Break keys were pressed.	
93	<b>NOSELECTORS</b>	No selector could be allocated.
	A Windows selector required by the library could not be allocated. Close any open Windows applications that are not required to be running, and try again.	
94	<b>NOBURSTMODE</b>	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	<b>NOTWINDOWSFUNC</b>	This function is not available in Windows library.
	The library function you called is not supported in the current revision of Universal Library for Windows Languages. It may be supported in the future. Contact technical support.	
96	<b>NOTSIMULCONF</b>	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()/AOutScan(). The jumpers on the D/A board must be set for simultaneous update before it will work.	
97	<b>EVENODDMISMATCH</b>	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 the channel numbers must be in even queue positions and odd channel numbers must be in odd queue positions.	
98	<b>M1RATEWARNING</b>	Sampling speed to system memory 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 is the case, data will overrun and sample data will be garbled. This warning is initiated whenever you request a sample rate over 625 kHz, AND the sample set is larger than the FIFO buffer on the board, AND an external memory board, such as a MEGA-FIFO is not being used. Your system may be able to handle the rate requested but only experimentation will bear this out. Your system may be capable of the full 1 MHz rate directly to system memory.	
99	<b>NOTRS485</b>	Selected board is not a RS-485 board.
	An attempt was made to call cbRS485()/RS485() with a board that is not RS485 compatible.	
100	<b>NOTDOSFUNCTION</b>	This function is not available in DOS.
	The function that was called is not available in the DOS version of the Universal Library.	
101	<b>RANGEMISMATCH</b>	Bipolar and unipolar ranges cannot be used together in A/D queue.
	The channel/gain queue should only be loaded (via cbALoadQueue()/ALoadQueue()) with all unipolar or bipolar ranges.	
102	<b>CLOCKTOOSLOW</b>	Sampling rate is too high for clock speed; change clock jumper on the board.
	The sampling rate that you requested is too fast. The A/D board pacer might be capable of running at a higher rate. Check the board for an XTAL jumper and, if it is not set for the highest rate, place the jumper in the position for the highest rate. After the jumper is set, re-run InstaCal.	
103	<b>BADCALFACTORS</b>	Calibration factors are invalid - Disabling software calibration.
	The selected board uses software calibration and the stored calibration factors are invalid. Run InstaCal and calibrate the board before using it.	
104	<b>BADCONFIGTYPE</b>	Invalid configuration information type specified.
	An invalid ConfigType argument was passed to either cbGetConfig() or cbSetConfig().	
105	<b>BADCONFIGITEM</b>	Invalid configuration item specified.
	An invalid ConfigItem argument was passed to either cbGetConfig() or cbSetConfig().	
106	<b>NOPCMCIABOARD</b>	Cannot access the PCMCIA board.
	Cannot access the specified PCMCIA board. Make sure that the PCMCIA Card & Socket Services are installed correctly and that the board was installed in the system correctly via InstaCal.	
107	<b>NOBACKGROUND</b>	Board does not support background operation.
	The BACKGROUND option was used and the specified board does not support background operation.	
108	<b>STRINGTOOSHORT</b>	The string argument is too short for the string being returned.
	The string passed to a library function or method is too small to contain the string that is being returned. Increase the size of the string to the minimum size specified for the function or method that you are using.	
109	<b>CONVERTTEXTMEM</b>	CONVERTDATA not allowed with EXTMEMORY option.
	You requested both the CONVERTDATA and EXTMEMORY option. These options cannot be used together. Collect the data without the CONVERTDATA option. After the data has been collected, read it back from the memory card (cbMemRead()/MemRead() or cbMemReadPretrig()/MemReadPretrig()), and use cbAConvertData()/AConvertData() to convert the data.	
110	<b>BADEUADD</b>	Program error bad values used in cbFromEngUnits or cbToEngUnits().
	Invalid floating point data was used in cbFromEngUnits()/FromEngUnits() or cbToEngUnits()/ToEngUnits(). Check the	

	arguments passed to the relevant function or method.	
111	<b>DAS16JRRATEWARNING</b>	Rates greater than 125 kHz must use on board 10 MHz clock.
	If a rate greater than 125 kHz is selected and the on board jumper is set for 1 MHz when using the CIO-DAS16/Jr, this warning is generated. Place the jumper on the 10MHz position and update your InstaCal settings.	
112	<b>DAS08TOOLOW_RATE</b>	The desired sample rate is below hardware minimum.
	Increase the value of the Rate argument in cbAInScan()/AInScan(). The lowest pacer frequency is the clock frequency (usually 8 MHz ÷ 2) ÷ by 65,535 for the CIO-, PC104 and PCM- DAS08.	
114	<b>AMBIGSENSORONGP</b>	More than one temperature sensor type defined for EXP-GP.
	Thermocouple and RTD types are both defined for an EXP-GP. cbTin()(Tin() and cbTinScan()/TinScan()) require that only one be defined to operate. Use InstaCal to set one of the sensor types to "Not Installed".	
115	<b>NOSENSORATYPEONGP</b>	No temperature sensor type defined for EXP-GP.
	Neither Thermocouple nor RTD types are defined for an EXP-GP. cbTin()(Tin() and cbTinScan()/TinScan()) require that one and only one be defined to operate. Use InstaCal to set one of the sensor types to a predefined type.	
116	<b>NOCONVERSIONNEEDED</b>	Selected 12-bit board already returns converted data.
	Some 12-bit boards do not need to have their data converted after a call to cbAInScan()/AInScan() with the NOCONVERTDATA option. These boards return no channel tags and therefore return data in its proper format. Calling cbAConvertData()/AConvertData() with data generated from these boards will generate this warning.	
117	<b>NOEXTCONTINUOUS</b>	CONTINUOUS mode cannot be used with EXTMEMORY.
	CONTINUOUS mode is ignored when used with the EXTMEMORY option.	
118	<b>INVALIDPRETRIGCONVERT</b>	cbAConvertPretrigData called after cbAPretrig failed.
	The data you are attempting to convert with cbAConvertPretrigData()/ AConvertPretrigData() can not be converted because cbAPretrig()/APretrig() did not return a complete data set, probably due to an early trigger.	
119	<b>BADCTRREG</b>	Bad counter argument passed to cbCLoad()
	The RegNum argument passed to cbCLoad() (CLoad()) is not a valid register.	
120	<b>BADTRIGTHRESHOLD</b>	Low trigger threshold is greater than high threshold.
	The LowThreshold arguments to cbSetTrigger()/SetTrigger() must be < the HighThreshold.	
121	<b>BADPCMSLOTREF</b>	NO PCM Card was found in the specified slot.
	This is usually caused by swapping PCMCIA cards and not re-running InstaCal. Run InstaCal.	
122	<b>AMBIGPCMSLOTREF</b>	Two identical PCM cards found. Please specify exact slot in InstaCal.
	This error occurs in DOS mode only when InstaCal is configured for a PCMCIA card in "any slot". To correct the problem, run InstaCal, go to the Install menu and pop up the board's menu. Highlight PCMCIA slot and choose either "0" or "1".	
123	<b>BADSENSORATYPE</b>	Invalid sensor type selected in InstaCal.
	The specified sensor type is not included in the allowed list of thermocouple/RTD types. Set the sensor type to a predefined type using InstaCal.	
126	<b>CFGFILENOTFOUND</b>	Cannot find CB.CFG file.
	The CB.CFG file could not be found. This file should be located in the same directory in which you installed the software.	
127	<b>NOVDDINSTALLED</b>	The CBUL.386 virtual device driver is not installed.
	The Windows device driver CBUL.386 is not installed on your system. Normally, it will be automatically installed when you run the standard installation program. The following line should be in your \windows\system.ini file in the [386Enh] section:	
128	<b>NOWINDOWSMEMORY</b>	Requested amount of Windows page-locked memory is not available.
	The Windows device driver could not allocate the required amount of physical memory. This error should not normally occur unless you are collecting very large amounts of data or your system is very memory constrained. If you are collecting a very large block of memory, try collecting a smaller amount. If this is not an option, than consider using cbFileAInScan()/FileAInScan() instead of cbAInScan()/AInScan(). Also, if you are running other programs, try shutting them down.	
129	<b>OUTOFDOSMEMORY</b>	Not enough DOS memory available.
	Try closing down any unneeded programs that are running.	
130	<b>OBSOLETEOPTION</b>	Obsolete option specified for cbSetConfig/cbGetConfig.
	The specified configuration item is no longer supported in the 32 bit version of the Universal Library.	
131	<b>NOPCMREGKEY</b>	No registry entry for this PCMCIA card.
	When running under Windows/NT, there must be an entry in the system registry for each PCMCIA card that you will be using with the system. This is ordinarily taken care of automatically by the Universal Library installation program. If this error occurs, contact technical support for assistance.	
132	<b>NOCBUL32SYS</b>	CBUL32.SYS device driver is not installed.
	The Windows device driver CBUL.SYS is not installed on your system. Normally, it will be automatically installed when you run the MCC standard installation program. Contact technical support for assistance.	
133	<b>NODMAMEMORY</b>	No DMA memory available to device driver.
	The Windows device driver could not allocate the minimum required amount of memory for DMA. If you are sampling at slower speeds, you can specify SINGLEIO in the Options argument to cbAInScan()/AInScan(). This will prevent the library from attempting to use DMA. In general though, this error should not ordinarily occur. Contact technical support	

	for assistance.	
134	<b>IRQNOTAVAILABLE</b>	IRQ not available.
	The Interrupt Level that was specified for the board (in InstaCal) conflicts with another board in your computer. Try switching to a different interrupt level.	
135	<b>NOT7266CTR</b>	This board does not have an LS7266 counter.
	This function or method can only be used with a board that contains an LS7266 chip. These chips are used on various quadrature encoder input boards.	
136	<b>BADQUADRATURE</b>	Invalid Quadrature argument passed to cbC7266Config().
	The Quadrature argument must be set to either NO_QUAD, X1_QUAD, X2_QUAD, or X4_QUAD.	
137	<b>BADCOUNTMODE</b>	Invalid counting mode specified.
138	<b>BADENCODING</b>	Invalid DataEncoding argument passed to cbC7266Config().
	The DataEncoding argument must be set to either BCD_ENCODING or BINARY_ENCODING.	
139	<b>BADINDEXMODE</b>	Invalid IndexMode argument passed to cbC7266Config()
	The IndexMode argument must be set to either INDEX_DISABLED, LOAD_CTR, LOAD_OUT_LATCH, or RESET_CTR.	
140	<b>BADINVERTINDEX</b>	Invalid InvertIndex argument passed to cbC7266Config()
	The InvertIndex argument must be set to either (CB)ENABLED or (CB)DISABLED.	
141	<b>BADFLAGPINS</b>	Invalid FlagPins argument passed to cbC7266Config()
	The FlagPins argument must be set to either CARRY_BORROW, COMPARE_BORROW, CARRYBORROW_UPDOWN, or INDEX_ERROR.	
142	<b>NOCTRSTATUS</b>	This board does not support cbCStatus()
	This board does not return any status information.	
143	<b>NOGATEALLOWED</b>	Gating cannot be used when indexing is enabled.
	Gating and indexing can not be used simultaneously. If Gating is set to (CB)ENABLED, then IndexMode must be set to INDEX_DISABLED.	
144	<b>NOINDEXALLOWED</b>	Indexing not allowed in non-quadrature mode
	Indexing is not supported when the Quadrature argument is set to NO_QUAD.	
145	<b>OPENCONNECTION</b>	Temperature input has open connection
146	<b>BMCONTINUOUSCOUNT</b>	Count must be integer multiple of packet size for Continuous mode.
147	<b>BADCALLBACKFUNC</b>	Invalid pointer to callback function or delegate passed as argument.
148	<b>MBUSINUSE</b>	Metrabus in use
149	<b>MBUSNOCTLR</b>	Metrabus I/O card has no configured controller card
150	<b>BADEVENTTYPE</b>	Invalid event type specified for this board.
	Although this board does support cbEnableEvent()/EnableEvent(), it does not support one or more of the event types specified.	
151	<b>ALREADYENABLED</b>	Event handler is already enabled for this event type.
	There is already an event handler bound to one or more of the events specified. To attach the new handler to the event type, first disable and disconnect the current handler using cbDisableEvent()/DisableEvent().	
152	<b>BADEVENTSIZE</b>	Invalid event count has been specified
	The ON_DATA_AVAILABLE event requires an event count greater than (0).	
153	<b>CANTINSTALLEVENT</b>	Unable to install event handler
	An internal error occurred while trying to setup the event handling.	
154	<b>BADBUFFERSIZE</b>	Buffer is too small for operation
	The memory allocated by cbWinBufAlloc()/WinBufAlloc() is too small to hold all the data specified in the operation.	
155	<b>BADAIMODE</b>	Invalid analog input mode
	Invalid analog input mode (RSE, NRSE, DIFF).	
156	<b>BADSIGNAL</b>	Invalid signal type specified
	The specified signal type does not exist, or is not valid for signal direction specified.	
157	<b>BADCONNECTION</b>	message
	The specified connection does not exist, or is not valid for the signal type and direction specified.	
158	<b>BADINDEX</b>	Invalid index specified.
	For Index >0, indicates that the specified index is beyond the end of the internal list of output connections assigned to the specified signal type.	

159	<b>NOCONNECTION</b>	Invalid connection
	No connection is assigned to the specified signal.	
160	<b>BADBURSTIOCOUNT</b>	Count cannot be greater than the FIFO size for BURSTIO mode. Furthermore, Count must be integer multiple of number of channels in scan.
	When using BURSTIO mode, the count entered cannot be larger than the FIFO size.	
161	<b>DEADDEV</b>	Device has stopped responding. Please check connections.
	Check cable connections to USB device and to your computer's USB port.	
163	<b>INVALIDACCESS</b>	Required access or privilege not acquired for specified operation. Please check for other users of device and restart application.
	You are currently not the device owner and therefore cannot change the state or configuration of the Ethernet device with functions such as cbAOut()/AOut(), cbDBitOut()/DBitOut(), cbAInScan()/AInScan(), cbFlashLED()/FlashLED(), and others. However, you can still read the state or configuration of the Ethernet device with functions such as cbAIn()/AIn(), cbDBitIn()/DBitIn(), and so on.	
164	<b>UNAVAILABLE</b>	Device unavailable at time of request. Please repeat operation.
	You requested an operation that conflicts with an operation in progress on the device. This error usually occurs in multithreaded applications or if you are running multiple applications that access the device. Both types of operations are not supported.	
165	<b>NOTREADY</b>	Device is not ready to send data. Please repeat operation.
	You requested an operation that conflicts with an operation in progress on the device. This error can occur during device initialization.	
169	<b>BITUSEDFORALARM</b>	The specified bit is used for alarm.
	You attempted to set the state of a digital output bit that is configured as an alarm input.	
170	<b>PORTUSEDFORALARM</b>	One or more bits on the specified port are used for alarm.
	You attempted to write to a digital output port that contains a bit configured as an alarm input.	
171	<b>PACEROVERRUN</b>	Pacer overrun; external clock rate too fast.
	You set the external clock rate to a value that is higher than the rate supported by the board.	
172	<b>BADCHANTYPE</b>	Invalid channel type specified.
	You set the channel type to a type that is not supported by the board.	
173	<b>BADTRIGSENSE</b>	Invalid trigger sensitivity specified.
	You set the trigger sensitivity to a value that is not supported by the board.	
174	<b>BADTRIGCHAN</b>	Invalid trigger channel specified.
	You set the trigger channel to a value that is not supported by the board.	
175	<b>BADTRIGLEVEL</b>	Invalid trigger level specified.
	You set the trigger level to a value that is not supported by the board.	
176	<b>NOPRETRIGMODE</b>	Pretrigger mode is not supported for the specified trigger type.
	You selected a trigger source that does not support pre-trigger data acquisitions.	
177	<b>BADDEBOUNCETIME</b>	Invalid debounce timing specified.
	You set the debounce time to a value that is not supported by the board.	
178	<b>BADDEBOUNCETRIGMODE</b>	Invalid debounce trigger mode specified.
	You set the debounce trigger mode to a value that is not supported by the board.	
179	<b>BADMAPPEDCOUNTER</b>	Invalid mapped channel specified.
	You mapped to a counter input channel that is not supported by the board.	
180	<b>BADCOUNTERMODE</b>	Invalid counter mode specified.
	This function cannot be used with the current mode of the specified counter.	
181	<b>BADTCCHANMODE</b>	Single-ended mode cannot be used for temperature input.
	You specified single-ended mode for use with a temperature input.	
182	<b>BADFREQUENCY</b>	Invalid frequency specified.
	You specified a frequency value that is not supported by the board.	
183	<b>BADEVENTPARAM</b>	Invalid event parameter specified.
	You specified an event parameter that is not supported by the board.	
184	<b>NONETIFC</b>	No interface devices were found with the required PAN and channel.
	No interface devices were detected whose PAN ID and RF channel number match those of a remote device.	
185	<b>DEADNETIFC</b>	The interface device(s) with the required PAN and channel has failed. Please check the connection.
	The interface device whose PAN ID and RF channel number match a remote device is not responding. Check the USB connection to the computer.	
186	<b>NOREMOTEACK</b>	The remote device is not responding to commands and queries. Please check the device.



	The wireless remote device is not responding. Check that the device is powered, that its PAN ID and RF channel match the interface device, and that the LEDs are functioning.	
187	<b>INPUTTIMEOUT</b>	The device acknowledged the operation, but has not completed before the timeout.
	The operation was acknowledged but has timed out before it was completed.	
188	<b>MISMATCHSETPOINTCOUNT</b>	Number of setpoints is not equal to number of channels with a setpoint flag set.
	Set the number of setpoints equal to the number of channels with a setpoint flag set.	
189	<b>INVALIDSETPOINTLEVEL</b>	Setpoint level is outside channel range.
	You specified a setpoint level that is outside of the range supported by the board.	
190	<b>INVALIDSETPOINTOUTPUTTYPE</b>	Setpoint Output Type is invalid.
	You specified a setpoint output type that is not supported by the board.	
191	<b>INVALIDSETPOINTOUTPUTVALUE</b>	Setpoint Output Value is outside channel range.
	You specified a setpoint output value that is outside of the range supported by the board.	
192	<b>INVALIDSETPOINTLIMITS</b>	Setpoint Comparison Limit B greater than Limit A.
	Set the setpoint comparison value for limit A to be larger than the value set for limit B.	
193	<b>STRINGTOOLONG</b>	The string length entered is too long for this operation.
	Enter a string up to the maximum number of characters specified for the function or method that you are using.	
194	<b>INVALIDLOGIN</b>	An invalid user name or password has been entered.
	Check that the password and user name entered were correct. If either has been lost, use the device reset button to reset the device to default values.	
195	<b>SESSIONINUSE</b>	Device session is already in use.
	Another user is currently logged in to a device session. Only one device session can be opened at a time.	
196	<b>NOEXTPOWER</b>	External power is not connected.
	External power is required. Connect the device to an external power supply.	
197	<b>BADDUTYCYCLE</b>	Invalid duty cycle specified.
	You attempted to set the duty cycle to a value not supported by the hardware.	
200-299	<b>200-299</b>	Internal 16-bit error
	Internal error occurred in the library. Refer to the specific errors below:	
201	<b>CANT_LOCK_DMA_BUF</b>	DMA buffer could not be locked.
	There is not enough physical memory to lock down enough DMA memory for this operation. Try closing out other applications, or installing additional RAM.	
202	<b>DMA_IN_USE</b>	DMA already controlled by another driver.
	The DMA controller is currently being used by another device, such as another DMA board or the floppy drive.	
203	<b>BAD_MEM_HANDLE</b>	Invalid Windows memory handle.
	The memory handle supplied is invalid. Memory handles supplied to library functions and methods should be allocated using cbWinBufAlloc()/WinBufAlloc(), and should not be de-allocated until BACKGROUND operations using this buffer are complete or cancelled with cbStopBackground()/StopBackground().	
300-399	<b>300-399</b>	Internal 32-bit error. See specific errors below.
304	<b>CFG_FILE_READ_FAILURE</b>	Error reading from configuration file.
	The program was unable to read the configuration file CB.CFG. Confirm that CB.CFG was not deleted, moved, or renamed since the software installation.	
305	<b>CFG_FILE_WRITE_FAILURE</b>	Error writing to configuration file.
	The program was unable to write to the configuration file CB.CFG. Confirm that CB.CFG is present and that its attributes are not set for Read-only. Also, check that not more than one application is trying to access this file.	
308	<b>CFGFILE_CANT_OPEN</b>	Cannot open configuration file.
	The program was unable to open the configuration file CB.CFG. Confirm that CB.CFG was not deleted, moved, or renamed since the software installation.	
325	<b>BAD_RTD_CONVERSION</b>	Overflow of RTD conversion.
	Either cbTin()/Tin() or cbTinScan()/TinScan() returned an invalid temperature conversion. Confirm that the configuration matches the RTD type, and physical EXP board settings; pay particular attention to gain settings and RTD base resistance. Also, check that the RTD leads are securely attached to the EXP terminals. Finally, confirm that the board is measuring reasonable voltages via cbAIn()/AIn().	
326	<b>NO_PCI_BIOS</b>	PCI BIOS not present on the PC.
	Could not locate the BIOS for the PCI bus. Consult PC supplier for proper installation of the PCI BIOS.	
327	<b>BAD_PCI_INDEX</b>	Specified PCI board not detected.
	The specified PCI board was not detected. Check that the PCI board is securely installed into the PCI slot. Also, run	

	InstaCal to locate/set valid base address and configuration.	
328	<b>NO_PCI_BOARD</b>	Specified PCI board not detected.
	The specified PCI board was not detected. Check that the PCI board is securely installed into the PCI slot. Also, run InstaCal to locate/set valid base address and configuration.	
334	<b>CANT_INSTALL_INT</b>	Cannot install interrupt handler. IRQ already in use.
	The device driver could not enable requested interrupt. Check that the selected IRQ is not already in use by another device. This error can also occur if a FOREGROUND scan was aborted; in such cases, rebooting the PC will correct the problem.	
339	<b>CANT_MAP_PCM_CIS</b>	Unable to access Card Information Structure.
	A resource conflict between the specified PCMCIA or PC-Card device and another device prevents the system from allocating sufficient resources to map the onboard CIS.	
344	<b>NOMOREFILES</b>	No more files in the directory.
	The end of the log file was reached before the file header was read.	
345	<b>BADFILENUMBER</b>	No file exists for the specified file number.
	The specified binary file number does not exist.	
347	<b>LOSSOFDATA</b>	The file may not contain all of the data from the logging session because the logging session was not terminated properly.
	The log file may be incomplete if the logging session is not properly terminated. Always end a logging session by pressing the data logging button until the LED turns off. Possible data loss may occur if the end of the log file is reached before the file header is read.	
348	<b>INVALIDBINARYFILE</b>	The file is not a valid MCC binary file.
	The binary file was not logged from an MCC USB device with data logging capability, or the binary file was logged during a data logging session that was not properly terminated and is missing information.	
349	<b>INVALIDDELIMITER</b>	Invalid delimiter specified for CSV file extension.
	When converting a binary log file to a comma-separated values text file (.CSV), the delimiter character must be set to a comma.	
400-499	<b>PCMCIA error</b>	Card & Socket Service error. Contact the manufacturer.
500-599	<b>Internal DOS error</b>	Contact the manufacturer.
600-699	<b>Internal Windows error</b>	Refer to specific errors below.
603	<b>WIN_CANNOT_ENABLE_INT</b>	Cannot enable interrupt. IRQ already in use.
	The device driver could not enable requested interrupt. Check that the selected IRQ is not already in use by another device. This error can also occur if a FOREGROUND scan was aborted; in such cases, rebooting the PC will correct the problem.	
605	<b>WIN_CANNOT_DISABLE_INT</b>	Cannot disable interrupts.
	The device driver was unable to disable the IRQ. This can occur when interrupts are generated too fast for the PC to complete servicing. For example, sampling at high frequencies (above ~2 kHz) with scan mode set for SINGLEIO can lead to this error. Frequently, an OVERRUN error accompanies this condition.	
606	<b>WIN_CANT_PAGE_LOCK_BUFFER</b>	Insufficient memory to page lock data buffer.
	There is not enough physical memory to lock down the entire data buffer. Try closing out other applications, selecting smaller data buffers, or installing additional RAM.	
630	<b>NO_PCM_CARD</b>	PCM card not detected.
	The specified PCMCIA card was not detected. Confirm that the PCM card is securely plugged into PCMCIA slot. If the board continues to return this error, run InstaCal to reset the configuration.	
801	<b>INVALIDGAINARRAYLENGTH</b>	The number of elements in the gain array must equal the number of channels in the scan.
	This error is generated when WinBufToEngArray() is called with the number of elements in gainArray is not equal to the number of channels specified. Make sure that the number of elements in the array is the same as the number of channels in the scan.	
802	<b>INVALIDDIMENSIONLENGTH</b>	The length of dimension 0 in the data array must equal the number of channels in the scan.
	This error is generated when WinBufToEngArray() is called with the length of dimension 0 of EngUnits not equal to the number of channels specified. Make sure that the length of dimension 0 in the array is the same as the number of channels in the scan.	
1000	<b>NOTEDSSENSOR</b>	No TEDS sensor was detected on the specified channel.
	Connect a TEDS sensor to the specified channel.	
1001	<b>INVALIDTEDSSENSOR</b>	Connected TEDS sensor to the specified channel is not supported.

	Connect a TEDS sensor that is supported by the hardware to the specified channel.	
1002	<b>CALIBRATIONFAILED</b>	Calibration failed.
	The attempt to calibrate the device has failed.	
1003	<b>BITUSEDFORTERMINALCOUNTSTATUS</b>	The specified bit is used for terminal count status.
	The terminal count status must be disabled for a digital bit before it can be used for timer output or DIO operations.	
1004	<b>PORTUSEDFORTERMINALCOUNTSTATUS</b>	One or more bits on the specified port are used for terminal count status.
	The terminal count status must be disabled for all digital bits before the port can be used for digital operations.	
1005	<b>BADEXCITATION</b>	Invalid excitation specified
	Refer to board-specific information for valid values.	
1006	<b>BADBRIDGETYPE</b>	Invalid bridge type specified
	Refer to board-specific information for valid values.	
1007	<b>BADLOADVAL</b>	Invalid load value specified
	Refer to board-specific information for valid values.	
1008	<b>BADTICKSIZE</b>	Invalid tick size specified
	Refer to board-specific information for valid values.	

**INVALIDGAINARRAYLENGTH and INVALIDDIMENSION0LENGTH errors only occur in the .NET class library.** The Universal Library will not print or stop if these errors occur, regardless of the error handling configuration specified by the call to MccService.ErrHandling. These errors must be checked by examining the ErrorInfo object returned from MccBoard.WinBufToEngArray.

## PDF Document

This help file is also available in PDF on our web site at [www.mccdaq.com/pdfs/manuals/Universal-Library-Help.pdf](http://www.mccdaq.com/pdfs/manuals/Universal-Library-Help.pdf).

Adobe® Reader® is required to view this document. Click on the link below to go to the Adobe Reader home page where you can download a free copy of Adobe Reader.

