miniLAB 1008™

USB Device
for
Data Acquisition

Software User's Guide

Document Revision 1, January, 2003

Lifetime warranty

Every hardware product manufactured by Measurement Computing Corp. is warranted against defects in materials or workmanship for the life of the product, to the original purchaser. Any products found to be defective will be repaired or replaced promptly.

30 Day Money Back Guarantee

Any Measurement Computing Corp. 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. Please call for an RMA number first. No credits or returns accepted without a copy of the original invoice. Some software products are subject to a repackaging fee.

These warranties are in lieu of all other warranties, expressed or implied, including any implied warranty of merchantability or fitness for a particular application. The remedies provided herein are the buyer's sole and exclusive remedies. Neither Measurement Computing Corp., 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 Corp. has been notified in advance of the possibility of such damages.

Notice

Measurement Computing Corporation does not authorize any Measurement Computing Corporation product for use in life support systems and/or devices without the written approval of the CEO of 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 Corp. 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.

Lifetime ***Harsh Environment Warranty™***

Any product manufactured by Measurement Computing Corp. that is damaged (even due to misuse) may be replaced for only 50% of the current list price. I/O boards face some tough operating conditions—some more severe than the boards are designed to withstand. When a board becomes damaged, just return the unit with an order for its replacement at only 50% of the current list price. We don't need to profit from your misfortune. By the way, we honor this warranty for any manufacturer's board that we have a replacement for.

Table of Contents

About this User's Guide

What you will learn from this user's guide

This user's guide explains how to install and configure the miniLAB 1008. In addition, the software Application Programming Interface (API) is explained.

This user's guide also refers you to related documents available on our web site, and to technical support resources that can also help you get the most out of these boards.

Conventions in this user's guide

For more information on …

Text presented in a box signifies additional information and helpful hints related to the subject matter you are reading.

Caution! Shaded caution statements present information to help you avoid injuring yourself and others, damaging your hardware, or losing your data.

<#:#> Angle brackets that enclose numbers separated by a colon signify a range of numbers, such those assigned to registers, bit settings, etc.

bold text **Bold text** is used for the names of objects on the screen, such as buttons, text boxes, and check boxes. For example:

1. Insert the disk or CD and click the **OK** button.

italic text *Italic text* is used for the names of manuals and help topic titles, and to emphasize a word or phrase. For example:

- The *Insta*Cal installation procedure is explained in the *Software Installation Manual*.
- *Never* touch the exposed pins or circuit connections on the board.

Where to find more information

The following electronic documents provide information that can help you get the most out of your miniLAB 1008.

- *miniLAB 1008 User's Guide* covers the hardware features of the miniLAB 1008 and is available on our web site at http://www.measurementcomputing.com/PDFmanuals/miniLAB-1008.pdf.
- *Specifications: miniLAB 1008* is available on our web site at http://www.measurementcomputing.com/pdfs/miniLAB-1008.pdf.
- MCC's *Guide to Signal Connections* is available on our web site at http://www.measurementcomputing.com/signals/signals.pdf.

Chapter 1

Installing the miniLAB 1008 USB software

Overview

The miniLAB 1008 is configured as a USB Human Interface Device (HID) class peripheral. As a USB class device, the miniLAB 1008 does not require a third-party device driver. The Microsoft USB HID class driver is used to enumerate and interface with the device. Therefore, you can install the software either before you connect the miniLAB 1008 to your computer or after you connect it to your computer. The hardware can be connected to any computer running Microsoft Windows 98SE, ME, 2000 or XP.

The Measurement Advantage Series installation CD contains the following files:

- miniLAB 1008 interface library (MccMinilab.dll)
- Measurement Computing USB configuration file (cbUSB.cfg)
- *miniLAB 1008 Software User's Guide* (this document)
- Read me file (ReadMe.txt)

Follow the steps below to install the USB software for the miniLAB 1008. Make sure your computer is running Microsoft Windows 98, ME, 2000 or XP.

1. Close all open applications.

2. Insert the installation CD into the CD ROM drive on the target system.

The autorun application on the CD should execute and display the **Measurement Computing USB CD** dialog window shown in Figure 1-1. If the autorun application does not start then the autorun application can be manually started by opening the CD and manually selecting the autorun.exe application.

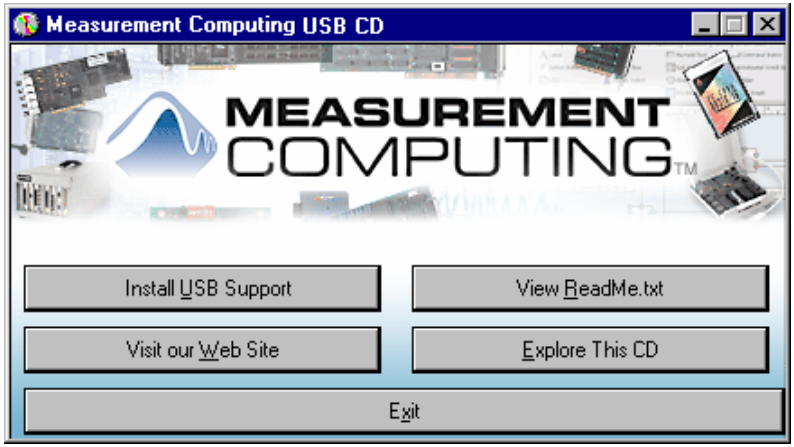

Figure 1-1. Measurement Computing USB CD Dialog Window

3. Click on the **Install USB Support** button to begin installing the software.

4. Follow the directions given on the subsequent screens to finish installing the software. If possible, use the default software settings, as this will make it easier to provide support if any is required. Reboot your computer when prompted.

After installing the software, connect the miniLAB 1008device. Refer to the chapter "Installing the miniLAB 1008," in the *miniLAB 1008 User's Guide* to learn how to connect this device. This document is available on our web site at http://www.measurementcomputing.com/PDFmanuals/miniLAB 1008.pdf.

Installing multiple devices

If your application needs more than one miniLAB 1008 on a single host system, then you need to follow certain steps in order for your application software to uniquely identify each device. The cbUSB.cfg configuration file manages the device mapping. Each miniLAB 1008 ships with the a Serial Number set to 0 in non-volatile memory. The default configuration file, cbUSB.cfg, contains a single entry that maps a single miniLAB 1008 device to DeviceNumber 0. The default configuration file is listed below. As long as only one device is connected, you do not need to change the configuration file, and the device is always programatically referenced as device 0.

```
[mccUSB]
FileType=Measurement Computing USB configuration file
FileFormat=ASCII
FileVersion=1
MaximumDevices=127
ConfiguredDevices=1

[Device.0]
DeviceType=USB-MINILAB1008
VendorID=0x09DB
DeviceID=0x0075
SerialNumber=0
DeviceNumber=0
Status=INSTALLED
```

However, if multiple devices are added to the host system then the following steps need to be taken.

1. Connect the first miniLAB 1008 device to the host system. This device will be referenced as device 0 with serial number 0 via the default configuration file.
2. Execute the application *ReNum.exe* to change the serial number of the current device. This application was installed in the same directory as the configuration file. The serial number can be re-assigned to any number between 1 and 255. Once this step is complete then exit the application.
3. Edit the cbUSB.cfg file and copy the [Device.0] section to create a new [Device.1] section. This is shown in the listing below. Now change the SerialNumber field for the [Device.0] section to reflect the newly assigned serial number.

```
[mccUSB]
FileType=Measurement Computing USB configuration file
FileFormat=ASCII
FileVersion=1
MaximumDevices=127
ConfiguredDevices=2

[Device.0]
DeviceType=USB-MINILAB1008
VendorID=0x09DB
DeviceID=0x0075
SerialNumber=1
DeviceNumber=0
Status=INSTALLED

[Device.1]
DeviceType=USB-MINILAB1008
VendorID=0x09DB
DeviceID=0x0075
SerialNumber=0
DeviceNumber=1
Status=INSTALLED
```

4. Disconnect the USB cable from the first unit, wait 5 to 10 seconds and re-insert the USB cable. This action will cause the first device to be re-enumerated with the new serial number.

5. Plug the second miniLAB 1008 device into the host computer or USB HUB. The second device will enumerate with serial number 0, since it has not been changed. The two devices can now be uniquely identified on the USB bus by serial number.

6. Record the serial number assigned to the device somewhere on the outside of the case. This serial number uniquely identifies the device if multiple devices are installed. The software API, discussed in detail below, relies on the DeviceNumber to communicate with each device. Each device number must be associated with a unique serial number via the configuration file.

The first time an application accesses the interface library, MccMinilab.dll, the configuration file is loaded and parsed. If the serial number of the specified device does not match the number recorded in the configuration file then the call will fail.

Default installation directories

During installation, several software components are copied onto your computer. The core software components and their default installation directories are listed in Table 1-1.

Table 1-1. Default Installation Directories

File	Default Installation Directory	Description
MccMinilab.dll	C:\MCCminiLAB	Interface library
CbUsb.cfg	C:\MCCminiLAB	Device configuration file
ReNum.exe	C:\MCCminiLAB	Multiple device configuration tool.

MccMinilab.dll Library Functions

This chapter covers in detail the API for writing user applications to communicate with and control the miniLAB 1008. For each function, the calling arguments and the return values will be presented. You can call the interface DLL—*MccMinilab.dll*—from any programming environment that provides support for accessing 32-bit DLL's. The most popular supported environments are Microsoft Visual C/C++ and Microsoft Visual Basic®.

For Visual Basic applications, you need to include the VB module `MccUsb.bas` (located in C:\MCCminiLAB\VB). This module contains the library function prototypes and unique definitions for interfacing to the miniLAB 1008 device.

For Visual C/C++ applications, you need to include the header file `MccUsb.h` and the library `MccMinilab.lib` (both located in C:\MCCminiLAB\C). Like the VB module, these files provide the interface definition for the library.

The library functions allow custom applications to control the miniLAB 1008. There are no separate initialization functions in the library. The first call to any of the following functions causes the configuration information to be loaded and a device list to be generated. Subsequent calls to any function in the library use the device list to map the BoardNum argument to a physical device. Table 2-1 lists the functions that are included in the library. A brief explanation of each function is presented in the table below. A more complete reference is provided in the following sections.

Table 2-1. miniLAB 1008 Software Functions

Function	Description
cbUSBDConfigPort	Sets the port direction for DIO bits.
cbUSBDConfigBit	Sets the direction for individual bits, only applicable for AUXPORT.
cbUSBDIn	Reads the current setting of a specified port.
cbUSBDOut	Writes a value to a specified port.
cbUSBDBitIn	Read the current bit setting for a specified port and bit position.
cbUSBDBitOut	Writes a single bit value to any write-enabled port.
cbUSBBlink	Triggers the USB LED to blink three times.
cbUSBAIn	Reads a specified analog input channel.
cbUSBAInScan	Scan a range of analog input channels.
cbUSBALoadQueue	Loads the channel/gain queue for analog input scanning.
cbUSBStopBackground	Stops a background analog input scan.
cbUSBSetTrig	Configures the external trigger input for analog input scans.
cbUSBGetStatus	Returns the status of a analog input scan operation.
cbUSBAOut	Writes data to a specified channel.
cbUSBCIn32	Read the 32-bit event counter.

cbUSBCInit	Performs initialization functions required to access the on-board counter.
cbUSBReset	Dynamically attempts to reset the USB device.
cbUSBGetErrMsg	Returns a descriptive error string for each error code.
cbUSBSetSerialNum	Used to write the device serial number.
cbUSBGetSerialNum	Used to read the device serial number.
cbUSBMemRead	Reads from the 'User Area' of the on-board FLASH memory.
cbUSBMemWrite	Writes to the 'User Area' of the on-board FLASH memory.
cbUSBWatchdog	Configures the on-board watchdog timer.
cbUSBToEngUnits	Converts raw binary data to engineering units.
cbUSBFromEngUnits	Converts data from engineering units to appropriate binary range for analog output.

BoardNum parameter

The first argument supplied to all of the library routines is the BoardNum. This parameter uniquely identifies a Measurement Advantage USB device when multiple Measurement Advantage USB devices are connected to the USB bus.

If more than one miniLAB 1008 device is installed on the host system, refer to the section "Installing Multiple Devices" on pg. 1-3 to learn how to uniquely identify each device in the system.

The following sections provide a complete overview of the software API that is available for programming the miniLAB 1008 device.

Configure port function

```
long cbUSBDConfigPort (int BoardNum, int PortNum, int
Direction)
```

Explanation

This function sets the direction of a DIO port to input or output. For 82C55 digital I/O devices, the port numbers are specified as DIO_PORTA, DIO_PORTB, DIO_PORTCL and DIO_PORTCH. Each of these ports can be configured for either input (CBDIN) or output (CBDOUT) operation. The four additional DIO bits that are available at the screw terminals, labeled DIO0-DIO3 can be individually configured for either input or output. DIO0-DIO3 are referenced through the PortNum argument DIO_AUXPORT. The DIO_AUXPORT I/O bits can be individually configured using the `cbUSBDConfigBit` function described below.

Arguments

- BoardNum — The target miniLAB device number.
- PortNum — The number of the port to configure. Valid port values for miniLAB devices are listed below:

`DIO_PORTA`	82C55 DIO Port A
`DIO_PORTB`	82C55 DIO Port B
`DIO_PORTCL`	82C55 DIO Port CL
`DIO_PORTCH`	82C55 DIO Port CH
`DIO_AUXPORT`	DIO0-DIO3

- Direction — Specifies DIO_DIR_OUT or DIO_DIR_IN.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >0 — Number code of error that occurred.

Configure bit function

```
long cbUSBDConfigBit (int BoardNum, int PortNum, int
BitNum, int Direction)
```

Explanation

This function sets the direction of individual DIO bits. This operation can only be used to configure DIO_AUXPORT digital I/O bits. The DIO_AUXPORT I/O bits refer to the four bits that are terminated at the screw terminals of the miniLAB 1008 and are labeled DIO0-DIO3.

Arguments

- BoardNum — The target miniLAB device number.
- PortNum — The number of the port to configure. Valid port values for miniLAB devices are listed below:

`DIO_AUXPORT`	DIO0-DIO3

- BitNum — The bit number to configure. Valid entries are indicated in the table below.

`0`	DIO0
`1`	DIO1
`2`	DIO2
`3`	DIO3

- Direction — Specifies DIO_DIR_OUT or DIO_DIR_IN direction.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer `>0` — Number code of error that occurred.

Read port function

```
long cbUSBDIn (int BoardNum, int PortNum, int *DataValue)
```

Explanation

This function reads the current state of the specified port. The specified port must first be configured as in input using the cbUSBDConfigPort() function. The returned DataValue will represent the current state of the DIO port.

Arguments

- BoardNum — The target miniLAB device number.
- PortNum — The number of the port to read. Valid port values for miniLAB devices are listed below:

DIO_PORTA	82C55 DIO Port A
DIO_PORTB	82C55 DIO Port B
DIO_PORTCL	82C55 DIO Port CL
DIO_PORTCH	82C55 DIO Port CH
DIO_AUXPORT	DIO0-DIO3

- DataValue — Returns the current value of the specified port when the function successfully executes.

Return Values

- 0 — CBUSB_SUCCESS. Function completed successfully.
- Integer >0 — Number code of error that occurred.

Write port function

```
long cbUSBDOut (int BoardNum, int PortNum, int DataValue);
```

Description

This function sets the specified DIO port to the state of DataValue. The specified port must first be configured as in output using the cbUSBDConfigPort() function.

Arguments

- BoardNum — The target miniLAB device number.
- PortNum — The number of the port to read. Valid port values for miniLAB devices are listed below:

`DIO_PORTA`	82C55 DIO Port A
`DIO_PORTB`	82C55 DIO Port B
`DIO_PORTCL`	82C55 DIO Port CL
`DIO_PORTCH`	82C55 DIO Port CH
`DIO_AUXPORT`	DIO0-DIO3

- DataValue — Integer variable that sets the state of the target PortNum.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer `>0` — Number code of error that occurred.

Read bit function

```
long cbUSBDBitIn (int BoardNum, int PortNum, int BitNum,
int *BitValue)
```

Explanation

This function reads a single bit value for any board port. It is similar to the previous read port function with the addition of a `BitNum` argument. The `BitNum` setting determines the individual input to read. The bit I/O functions are not very efficient since a complete USB transaction must be made to transfer one bit of data. The Port I/O functions should be utilized for the majority of digital I/O functionality.

Arguments

- BoardNum — The target miniLAB device number.
- PortNum — The number of the port to read. Valid port values for miniLAB devices are listed below:

`DIO_PORTA`	82C55 DIO Port A
`DIO_PORTB`	82C55 DIO Port B
`DIO_PORTCL`	82C55 DIO Port CL
`DIO_PORTCH`	82C55 DIO Port CH
`DIO_AUXPORT`	DIO0-DIO3

- BitNum — The bit number to read. For DIO_PORTA and DIO_PORTB this value can range from 0 to 7. For DIO_PORTCL, DIO_PORTCH, and DIO_AUXPORT, this value can range from 0 to 3.
- BitValue — Returns the setting of the specified bit when the function successfully executes. The bit value is either `0` or `1`.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Write bit function

```
long cbUSBDBitOut (int BoardNum, int PortNum, int BitNum,
int BitValue)
```

Explanation

This function writes a single bit value to any port that is write-enabled. It complements the read-bit function `cbUSBDBitIn`. The `BitNum` setting determines the input bit to write to. The miniLAB device executes a Read/Modify/Write operation to set the specified bit without effecting the state of the remaining bits in the port.

Arguments

- BoardNum — The target miniLAB device number.
- PortNum — The number of the port to write. Valid port values for miniLAB devices are listed below:

`DIO_PORTA`	82C55 DIO Port A
`DIO_PORTB`	82C55 DIO Port B
`DIO_PORTCL`	82C55 DIO Port CL
`DIO_PORTCH`	82C55 DIO Port CH
`DIO_AUXPORT`	DIO0-DIO3

- BitNum — The bit number to read. For DIO_PORTA and DIO_PORTB this value can range from 0 to 7. For DIO_PORTCL, DIO_PORTCH, and DIO_AUXPORT, this value can range from 0 to 3.
- BitValue — Sets the state of the specified bit when the function successfully executes. The bit value is either `0` or `1`.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Blink LED function

```
long cbUSBBlink (int BoardNum)
```

Explanation

This function is a simple communication test. When this function is called, the STATUS LED will blink three times.

Arguments

- BoardNum — The target miniLAB device number.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Read analog input function

```
long cbUSBAIn (int BoardNum, int Channel, int Gain,
unsigned short *Data
```

Explanation

This function reads the specified analog input channel and returns the data in offset binary format. If the Gain is selected as `GAIN2_DIFF` then a –10 volt input will return 0, a 0 volt input will return 2048, and a +10 volt input will return 4095. Select the input range with the `Gain` argument. Valid `Gain` values are shown in the table below. Each channel is configured for either single-ended (SE) or differential (DIFF) mode through the Gain parameter, as shown below.

Arguments

- BoardNum — The target miniLAB device number.
- Channel — Number of the input channel to read. Valid settings for differential (Diff) and single-ended (SE) analog inputs are listed below.

  `0-3` Diff
  `0-7` SE

- Gain — This specifies the gain you wish to apply to the conversion. Valid settings for single-ended mode are listed below.

GAIN1_SE	Bipolar 10 volt range, A/D gain = 1.

- Valid settings for differential mode Gain are listed below.

GAIN1_DIFF	Bipolar 20 volt range, A/D gain = 1.
GAIN2_DIFF	Bipolar 10 volt range, A/D gain = 2.
GAIN4_DIFF	Bipolar 5 volt range, A/D gain = 4.
GAIN5_DIFF	Bipolar 4 volt range, A/D gain = 5.
GAIN8_DIFF	Bipolar 2.5 volt range, A/D gain = 8.
GAIN10_DIFF	Bipolar 2 volt range, A/D gain = 10.
GAIN16_DIFF	Bipolar 1.25 volt range, A/D gain = 16.
GAIN20_DIFF	Bipolar 1 volt range, A/D gain = 20.

- Data (output) — 12-bit integer returned when the function successfully executes.

Return Values

- 0 — CBUSB_SUCCESS. Function completed successfully.
- Integer >0 — Number code of error that occurred.

Scan analog input function

```
long cbUSBAInScan (int BoardNum, int StartChan, int
EndChan, int Count, int *Rate, int Gain, unsigned short
*Data, int Options)
```

Explanation

This function scans a range of analog input channels. The cbUSBAInScan currently supports AD_SINGLE_MODE, AD_BURST_MODE and AD_CONT_MODE scanning modes. If the Options argument is left as 0 then the default scanning mode will be AD_SINGLE_MODE.

The AD_SINGLE_MODE mode is used to collect a finite amount of data in the user specified data buffer. In this mode, the Data buffer must be allocated to be at least Count samples long. The maximum sample rate for this mode is 1.2kS/s.

In AD_BURST_MODE mode, the device collects data from multiple channels and stores the data to on-board memory at the specified rate. AD_BURST_MODE mode offers the fastest data collection rate (up to 8ksps). However, the total sample count (Count argument) is limited to 4096 samples.

In AD_CONT_MODE mode, the function stores samples to the on-board memory and simultaneously updates the host. The maximum data collections rate for this mode is 1.2kS/s. The user supplied Data buffer will be used as a circular buffer for storing collected data. The function cbUSBGetStatus() can be used to determine the progress of the scan.

In addition to the scanning modes there are two other options that modify the scan behavior, they are AD_BACKGROUND and AD_EXTTRIG.

If AD_BACKGROUND is not selected, all of the requested data will be collected prior to returning from the call.

The AD_BACKGROUND mode allows the function call to return while data is collected in the background. The status of the background scan can be monitored by subsequent calls to cbUSBGetStatus().

The AD_EXTTRIG option may be used in conjunction with any of the scan modes to control the start of the acquisition through an external trigger. This option must be preceded by a call to cbUSBDSetTrig() to configure the digital trigger.

Arguments

- BoardNum — The target miniLAB device number.
- StartChan — Number of the first channel in the input scan range.
- EndChan — Number of the last channel in the input scan range.
- Count — Total number of samples to collect from all channels. The number of samples collected for each channel will be Count / (EndChan-StartChan +1). When using AD_BURST_MODE, the maximum number of samples is limited to 4096.
- Rate (output) — User specified data rate. This is the per channel scan rate. The aggregate rate will be this number times the number of channels. In AD_CONT_MODE mode the maximum aggregate rate is 1200. In AD_BURST_MODE mode the maximum aggregate rate is 8000. This value is not always obtainable and the actual rate will be returned to this argument.

- Gain — This specifies the gain you wish to apply to the conversion. Valid settings for single-ended mode are listed below.

`GAIN1_SE`	Bipolar 10 volt range, A/D gain = 1.

- Valid settings for differential mode Gain are listed below.

`GAIN1_DIFF`	Bipolar 20 volt range, A/D gain = 1.
`GAIN2_DIFF`	Bipolar 10 volt range, A/D gain = 2.
`GAIN4_DIFF`	Bipolar 5 volt range, A/D gain = 4.
`GAIN5_DIFF`	Bipolar 4 volt range, A/D gain = 5.
`GAIN8_DIFF`	Bipolar 2.5 volt range, A/D gain = 8.
`GAIN10_DIFF`	Bipolar 2 volt range, A/D gain = 10.
`GAIN16_DIFF`	Bipolar 1.25 volt range, A/D gain = 16.
`GAIN20_DIFF`	Bipolar 1 volt range, A/D gain = 20.

- Options — Specifies the channel scanning options. See the previous text for details. Some valid combinations are shown below.
  - AD_SINGLE_MODE
  - AD_CONT_MODE | AD_BACKGROUND
  - AD_BURST_MODE | AD_BACKGROUND | AD_EXTTRIG
- Data (output) — The collected data returned in an array. The array must be sized to at least Count. A fatal error will occur if this array is not dimensioned correctly.

Return Values

`0`	`CBUSB_SUCCESS`. Function completed successfully.
Integer >`0`	Number code of error that occurred.

Stop background scan function

```
long cbUSBStopBackground (int BoardNum)
```

Explanation

This function will stop any background task that is running on specified device. Typically this function will be used in conjunction with an analog input scan that was started in a continuous background mode. The analog input scan will continue until this function is called.

In software, first execute the `cbUSBAInScan` function with the Options argument set to AD_CONT_MODE. Once the required data has been collected then stop the scan with the `cbUSBStopBackground()` command.

Arguments

- BoardNum — The target miniLAB device number.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Channel gain queue configuration function

```
long cbUSBALoadQueue (int DeviceNum, short *ChanArray,
short *GainArray, int Count)
```

Explanation

The miniLAB 1008 can scan analog input channels with different gain settings. This function provides the mechanism for configuring each channel with a unique range. Each ChanArray entry must align with a corresponding GainArray entry. The maximum number of entries is 8. A combination of single-ended and differential channels may be included in the queue. Be careful to number the channels correctly when mixing single-ended and differential channels. Two sample configurations are shown below.

Table 2-2. Four differential channels with unique gains.

Index	ChanArray	GainArray
0	0	GAIN1_DIFF
1	1	GAIN2_DIFF
2	2	GAIN4_DIFF
3	3	GAIN5_DIFF

Table 2-3. Two differential channels with unique gains and four single-ended channels.

Index	ChanArray	GainArray
0	0	GAIN1_DIFF
1	1	GAIN2_DIFF
2	4	GAIN1_SE
3	5	GAIN1_SE
4	6	GAIN1_SE
5	7	GAIN1_SE

Arguments

- BoardNum — The target miniLAB device number.
- ChanArray — Array of channel entries, maximum of eight elements. Valid channel entries are shown in the table below.

0-3	Diff
0-7	SE

- GainArray — Array of gain entries, maximum of eight elements. Valid gain entries are shown in the table below.

GAIN1_SE	Bipolar 10 volt range, A/D gain = 1.
GAIN1_DIFF	Bipolar 20 volt range, A/D gain = 1.
GAIN2_DIFF	Bipolar 10 volt range, A/D gain = 2.
GAIN4_DIFF	Bipolar 5 volt range, A/D gain = 4.
GAIN5_DIFF	Bipolar 4 volt range, A/D gain = 5.
GAIN8_DIFF	Bipolar 2.5 volt range, A/D gain = 8.
GAIN10_DIFF	Bipolar 2 volt range, A/D gain = 10.
GAIN16_DIFF	Bipolar 1.25 volt range, A/D gain = 16.
GAIN20_DIFF	Bipolar 1 volt range, A/D gain = 20.

- Count — This argument represents the total number of entries in the queue. The maximum count is eight.

Return Values

- 0 — CBUSB_SUCCESS. Function completed successfully.
- Integer >0 — Number code of error that occurred.

Set analog trigger function

```
long cbUSBSetTrig (int BoardNum, int Type, int Channel)
```

Explanation

This function configures the external trigger for analog input. One of the four digital I/O connections, DIO0-DIO3, may be utilized as an external trigger input. The trigger may be configured to activate with either a logic HIGH or LOW input. Once the trigger is received the analog input will proceed as configured. The AD_EXTTRIG option must be used in the `cbUSBAInScan` call to utilize this feature.

In software, first execute the `cbUSBSetTrig()` function to configure the external trigger and then follow this with a call to `cbUSBAInScan()`.

Arguments

- BoardNum — The target miniLAB device number.
- Type — Configures external trigger input for HIGH or LOW assertion.

Type	Explanation
AD_TRIGHIGH	Trigger on logic level HIGH
AD_TRIGLOW	Trigger on logic level LOW

- Channel — Specifies the digital input that will be used for the external trigger. Valid entries for the argument are 0-3.

Channel	Digital input
0	DIO0
1	DIO1
2	DIO2
3	DIO3

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Write analog output function

```
long cbUSBAOut (int BoardNum, int Channel, int Gain,
unsigned short Data)
```

Explanation

This function sets the specified analog output channel to the target `Data` value. For the miniLAB 1008, the data has a maximum value of 1023 (10-bits). The analog output has a range of 0-5 volts. Note, however, that the 5 volt power for the device is supplied by the USB connection. In some instances this voltage may vary slightly from 5 volts. In this case the maximum output voltage will be under 5 volts.

Arguments

- BoardNum — The target miniLAB device number.
- Channel — Number of the output channel to write to, either 0 or 1.
- Gain — `GAIN1_DAC` is the only supported gain for the miniLAB 1008 analog output.
- Data (output) — 10-bit unsigned short value written to the channel when the function successfully executes.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer `>0` — Number code of error that occurred.

Read event counter function

```
long cbUSBCIn32 (int BoardNum, int CounterNum, unsigned
long *Data)
```

Explanation

This function reads the 32-bit event counter on the miniLAB 1008. This counter tallies the transitions of an external input attached to the CTR pin on the screw terminal.

Arguments

- BoardNum — The target miniLAB device number.
- CounterNum — This argument is supplied for future expansion. For the miniLAB 1008 series the only valid value is 0.
- Data (output) — 32-bit number returned when the function successfully executes.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Initialize counter function

```
long cbUSBCInit (int BoardNum)
```

Explanation

This function resets the count to zero and initializes the hardware.

Arguments

- BoardNum — The target miniLAB device number.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Reset USB device function

```
long cbUSBReset (int BoardNum)
```

Explanation

This function causes the miniLAB device to perform a soft reset. The device simulates a disconnect from the USB bus which in turn causes the host computer to re-enumerate the device. This call should be used as a last resort to restore communication to a device.

Arguments

- BoardNum — The target miniLAB device number.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Write serial number function

```
long cbUSBSetSerialNum (int BoardNum, char *SerialNum)
```

Explanation

This function assigns a serial number to the target miniLAB device. Each miniLAB device will initially be assigned 0 as the `SerialNum`. In systems where only one device is installed this will never need to be changed. However, in systems where multiple miniLAB devices are connected, each device will need to be assigned a unique `SerialNum`.

Arguments

- BoardNum — The target miniLAB device number.
- SerialNum — Serial number to assign to the board. The valid serial numbers are shown in the table below.

`0`	Default serial number
`1-255`	User assigned serial number

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Read serial number function

```
long cbUSBGetSerialNum (int BoardNum, char *SerialNum)
```

Explanation

This function reads the serial number from the target miniLAB device. The serial number is used to uniquely identify a device with the corresponding `BoardNum`.

Arguments

- BoardNum — The target miniLAB device number.
- SerialNum — The serial number read from the board.

`0`	Default serial number
`1-255`	User assigned serial number

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer `>0` — Number code of error that occurred.

Read on-board memory function

```
long cbUSBMemRead (int BoardNum, int Address, PVOID Data,
int Count)
```

Explanation

This function is provided to read the user area of the non-volatile memory on the miniLAB device. The non-volatile memory is used to store analog input data, calibration coefficients, and some system information. In addition, there are 1944 bytes of user addressable space that can be written to and/or read from.

Arguments

- BoardNum — The target miniLAB device number.
- Address — The start address to read from the non-volatile memory. Areas of memory from 0x0000 to 0x1FFF may be read, but generally, the area of interest would be the User Data area. Valid ranges for this address are 0x1800 to 0x1EFF. The USER_AREA_START definition can be used as the starting address.
- Data — This is a pointer to a user defined memory buffer to store the return data. This buffer must be at least `Count` bytes in length or a serious system error will occur.
- Count — The number of bytes of data to read. Data is transferred across the USB bus in 8-byte segments, so it is most efficient to specify `Count` in multiples of 8 bytes.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Write on-board memory function

```
long cbUSBMemWrite (int BoardNum, int Address, PVOID Data,
int Count)
```

Explanation

This function is provided to write to the non-volatile memory on the miniLAB device. The non-volatile memory is used to store analog input data, calibration coefficients, and some system information. In addition, there are 1944 bytes of user addressable space that can be written to and read from called the User Data area.

Arguments

- BoardNum — The target miniLAB device number.
- Address — The start address to write to the non-volatile memory. Areas of memory from 0x0000 to 0x1FFF may be written, but generally, the area of interest would be the User Data area. Valid ranges for this address are 0x1800 to 0x1FF0. The USER_AREA_START definition can be used as the starting address. Use caution when writing to non-volatile memory so that you don't accidentally overwrite data or calibration factors.
- Data — This is a pointer to a user defined memory buffer where the data to write out to the device is stored. This buffer must be at least `Count` bytes in length or a serious system error will occur.
- Count — The number of bytes of data to write. Data is transferred across the USB bus in 8-byte segments, so it is most efficient to specify `Count` in multiples of 8 bytes.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Watchdog timer function

```
long cbUSBWatchdog (int BoardNum, int Status, int Timeout,
int Action, int Channel, int State)
```

Explanation

The watchdog function is provided as a mechanism to recover from a loss of communication between the host computer and the miniLAB 1008. Typically, this function is only required for systems that are executing a continuous analog input scan in an unmanned environment. Once the watchdog function is executed, the miniLAB 1008 will continue to reset the on-board timer as long as there is communication with the host computer. If communication is lost with the host computer for the specified amount of time then the watchdog function will execute the specified action. Proper execution of this function assumes that the communication failure is the result of a host error. One thing that could cause host communication issues would be if a USB device is connected to the bus while the host is performing a continuous scan to the miniLAB 1008.

Arguments

- BoardNum — The target miniLAB device number.
- Status — Specifies if the call is enabling or disabling the watchdog timer. To enable the watchdog timer set this argument to WD_ENABLE. To disable the watchdog timer set this argument to WD_DISABLE.
- Timeout — This is the amount of idle time to allow before executing the specified action. The entry is specified in seconds and can be between 1 and 5000.
- Action — If the watchdog timer expires then this argument specifies what action to take.

Action	Explanation
WD_RESET	This will attempt to reset the device. The miniLAB 1008 will simulate a disconnect/reconnect on the USB bus.
WD_DIO	Using the Channel and State arguments this option will assert a DIO bit upon watchdog timeout.

-

- `Channel` — If Action is set to WD_DIO, then this argument will specify the digital I/O bit to assert on failure. The table below shows valid options.

Channel	Digital I/O
0	DIO0
1	DIO1
2	DIO2
3	DIO3

- State — If the Action is set to WD_DIO then this argument will specify the state that the DIO pin will be set to upon timeout. Valid entries are 0 for logic LOW and 1 for logic HIGH.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Data conversion functions

```
long cbUSBFromEngUnits (int BoardNum, int Range, float
EngUnits, unsigned short *DataVal)
```

Explanation

This function is provided to convert engineering units data into offset binary for output to the devices D/A channels. The function takes a floating point value and applies the specified `Range` to convert the input to binary. For the miniLAB 1008 the `Range` argument can only be `GAIN1_DAC`.

Arguments

- BoardNum — The target miniLAB device number.
- Range — The miniLAB 1008 only supports D/A outputs of 0 to 5 volts so this parameter must be GAIN1_DAC.
- EngUnits — This represents the data that is to be converted to binary for output to the D/A converter.
- DataVal — The result of the conversion will be returned in this parameter.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer `>0` — Number code of error that occurred.

```
long cbUSBToEngUnits (int BoardNum, int Range, unsigned int
DataVal, float *EngUnits)
```

Explanation

This function is provided to convert offset binary data, returned from the miniLAB 1008 analog input channels, input engineering units. The function takes a binary value and applies the specified `Range` to convert the input to a floating point voltage.

Arguments

- BoardNum — The target miniLAB device number.
- Gain — This specifies the gain you wish to apply to the conversion. Valid settings for single-ended mode are listed below.

`GAIN1_SE`	Bipolar 10 volt range, A/D gain = 1.

Valid settings for differential mode Gain are listed below.

`GAIN1_DIFF`	Bipolar 20 volt range, A/D gain = 1.
`GAIN2_DIFF`	Bipolar 10 volt range, A/D gain = 2.
`GAIN4_DIFF`	Bipolar 5 volt range, A/D gain = 4.
`GAIN5_DIFF`	Bipolar 4 volt range, A/D gain = 5.
`GAIN8_DIFF`	Bipolar 2.5 volt range, A/D gain = 8.
`GAIN10_DIFF`	Bipolar 2 volt range, A/D gain = 10.
`GAIN16_DIFF`	Bipolar 1.25 volt range, A/D gain = 16.
`GAIN20_DIFF`	Bipolar 1 volt range, A/D gain = 20.

- DataVal — This represents the data returned from the A/D converter that is to be converted to a floating point voltage.
- EngUnits — The result of the conversion will be returned in this parameter.

Return Values

- `0` — `CBUSB_SUCCESS`. Function completed successfully.
- Integer >`0` — Number code of error that occurred.

Get error message function

```
long cbUSBGetErrMsg (long ErrCode, char *ErrMsg)
```

Explanation

The `cbUSBGetErrMsg` function converts the return value from any of the library functions into an error message. Please refer to Table 2-4 for specific error codes. In addition, any system errors that occur as the result of a library function call will return the associated Win32 error message.

Caution! The application must allocate the ErrMsg string to be at least CBUSB_ERRSTRLEN long. Currently, CBUSB_ERRSTRLEN is defined to be 80. Failure to allocate this string correctly will result in a severe system error.

Arguments

- ErrCode — Number code of the error that occurred
- ErrMsg (output) — String explaining the error that occurred. This string must be at least `CBUSB_ERRSTRLEN` characters long (currently 80).

Return Values

- `0` — `CBUSB_SUCCESS`. Call completed successfully.

Error Codes

Table 2-4. miniLAB 1008 Software Error Codes

Code	Description
CBUSB_SUCCESS	Success
CBUSB_FILE_OPEN_FAILED	Error opening configuration file
CBUSB_REGISTRY_OPEN_ERROR	Error opening registry key
CBUSB_REGISTRY_QUERY_ERROR	Error querying registry key
CBUSB_DEVICE_OPEN_FAILED	Error opening USB device
CBUSB_NO_DETECT	No USB device detected
CBUSB_INVALID_DEVICE_NUM	Invalid device number
CBUSB_INVALID_PORT_NUM	Invalid port number
CBUSB_INVALID_BIT_NUM	Invalid bit number
CBUSB_NO_DEVICES_IN_CFG	No devices in CbUsb.cfg file
CBUSB_AOUT_ERR	Analog output error
CBUSB_AIN_REQUEST_ERR	Analog input request error
CBUSB_AIN_READ_ERR	Analog input read error
CBUSB_CINIT_ERR	Event counter initialization error
CBUSB_CIN32_REQUEST_ERR	Event counter request error
CBUSB_CIN32_READ_ERR	Event counter read error,
CBUSB_DCONFIG_ERR	Digital I/O configuration error,
CBUSB_DIN_REQUEST_ERR	Digital I/O request error,
CBUSB_DIN_READ_ERR	Digital I/O read error,
CBUSB_DOUT_ERR	Digital output error,
CBUSB_AINSCAN_REQUEST_ERR	Analog input scan request error,
CBUSB_AINSCAN_READ_ERR	Analog input scan read error,
CBUSB_MAXRATE_ERR	Maximum sample rate exceeded,
CBUSB_MINRATE_ERR	Specified sample rate is below minimum,
CBUSB_MAXCOUNT_ERR	Specified count exceeds limits,
CBUSB_AINSTOP_ERR	Error stopping analog input scan,
CBUSB_AIGETDATA_ERR	Error retrieving analog input data,
CBUSB_MAXCONT_RATE_ERR	Specified sample rate exceeds maximum
CBUSB_DCONFIGBIT_ERR	Error configuring digital I/O bit direction,
CBUSB_GETSNUM_ERR	Request for device ID number failed,
CBUSB_SETSNUM_ERR	Request to set device ID number failed,
CBUSB_LOADQ_ERR	Error loading the channel/gain queue,
CBUSB_BAD_CHANNEL	Invalid channel specified,

Code	Description
CBUSB_BAD_RANGE	Invalid range specified
CBUSB_INVALID_DATA	Data specified is out of range
CBUSB_MAXQ_CHANS	Exceeded maximum count for Channel/Gain
CBUSB_DATA_RANGE_ERR	Data out of range
CBUSB_OVERRUN_ERR	Data overrun error
CBUSB_SCAN_IN_PROGRESS	Analog input scan already in progress
CBUSB_THREAD_FAIL	Error creating read thread
CBUSB_BUFSIZ_ERR	Input buffer must be an even multiple of 64
CBUSB_RESET_ERR	Error resetting device
CBUSB_PACKET_SYNC_ERR	Data synchronization error
CBUSB_INVALID_ADDRESS	Invalid memory address
CBUSB_MEMREAD_ERR	Error reading device memory
CBUSB_MEMWRITE_ERR	Error writing device memory
CBUSB_CAL_READ_ERR	Error reading calibration data from device
CBUSB_NOSUPPORT	This function is not supported by the selected device.
CBUSB_SETTRIG_ERR	Communication error attempting to set external trigger.
CBUSB_INVALID_TRIG_TYPE	Invalid trigger type, TRIGLOW or TRIGHIGH.
CBUSB_OPTIONS_ERR	AD_FOREGROUND operation can not be specified with AD_CONT_MODE.
CBUSB_INVALID_TIMEOUT	Watchdog timeout can not be 0.
CBUSB_PORT_DIR_ERR	Port direction setting is inconsistent with operation.
CBUSB_BLINK_ERR	Error executing the BLINK command.
CBUSB_CHAN_CFG_ERR	Invalid channel configuration, max differential channels is 4.
CBUSB_WATCHDOG_ACTIVE	Disable watchdog prior to attempting counter access.

Measurement Computing Corporation
16 Commerce Boulevard,
Middleboro, Massachusetts 02346
(508) 946-5100
Fax: (508) 946-9500
E-mail: info@measurementcomputing.com
www.measurementcomputing.com