ACCES I/O Products USB Library

Generated by Doxygen 1.8.6

Tue Nov 29 2016 10:19:14

Contents

Chapter 1

Main Page

ACCES I/O Products, Inc.

1.1 AIOUSB driver library

This project contains USB drivers and APIs for ACCES I/O Product's line of USB based data acquisition modules. This driver represents a large API collection for communicating with one or more of ACCES I/O Product's line of USB based data acquisition products. All of the core functionality that exists and is supported by the Windows software is implemented in this library for non-Windows based operating systems. This code base compiles using either GCC and Clang compilers to both shared and static libraries that be can used in applications that need to perform highspeed USB data acquisition.

The entire set of drivers are rely on functionality provided by the `libusb-1.0` library. Please see the `prequisites` section to find out about required software for building the driver.

Currently, this project provides full support to the following platforms:

- Linux
- Mac OS X
- Free / Net BSD
- Rasberry Pi
- Beagle Board
- POSIX compliant operating systems that can successfully compile and use libusb.
- Windows with Cygwin

1.1.1 Prerequisites

The functionality in this driver depends on the following installed packages.

1. `libusb-1.0`
2. `cmake`
3. `swig`
4. (Optionally for python bindings) `python-dev`

Ubuntu / Debian

```
sudo apt-get install libusb-1.0 libusb-1.0-0-dev cmake swig python-dev
```

Fedora / Red Hat

```
sudo yum install libusb-1.0 cmake swig
```

Open SUSE

```
sudo zypper install libusb-1.0 cmake swig
```

Mac OS X

Homebrew

```
brew install libusb  cmake
```

Darwin Ports

```
sudo port install libusb cmake
```

Rasberry Pi

```
sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev  cmake
sudo apt-get install python-dev # If you want Python bindings
```

Beagle Board

```
sudo apt-get install libusb-1.0-0 libusb-1.0-0-dev cmake
sudo apt-get install python-dev # If you want Python bindings
```

1.2 Building on Linux/ MacOS / BSD / *NIX systems

Building ACCES I/O Products' Driver library amounts to compiling C source files to produce C and C++ based shared (.so) or static (.a) libraries. The build process relies on either GNU make or Cmake. The first method of building (see non-cmake users is a little more involved but will give you the ability to build wrapper language packs. Currently ,the simplified cmake system is easier to build and install the general libraries but we have been unable to use it to deploy the Swig based wrappers as we would have liked. The other option is the CMake build.

1.2.1 GNU make build

You will need to do the following

```
cd AIOUSB
source sourceme.sh
cd ${AIO_LIB_DIR} && make && cd -
cd ${AIO_CLASSLIB_DIR} && make && cd -
cd samples/USB_SAMPLE_OF_CHOICE
make sample AIOUSBLIBDIR=${AIO_LIB_DIR} AIOUSBCLASSLIBDIR=${AIO_CLASSLIB_DIR}
      DEBUG=1
```

1.2.2 Build with CMake

```
cd AIOUSB
mkdir build
cd build
cmake ..
make
sudo make install
```

1.2.3 Installation

Linux Installation

1. Install fxload either using the appropriate installation tool for your platform or by installing from https://github.com/accesio/fxload. Copy fxload to a standard location in your $PATH.
2. sudo cp AIOUSB/Firmware/*.hex /usr/share/usb/
3. sudo cp AIOUSB/Firmware/10-acces*.rules /etc/udev/rules.d

Mac Installation (work in progress!!)

1. Build and Install fxload from https://github.com/accesio/fxload and copy fxload to a standard location in your $PATH.
2. Determine the raw USB Device ID for your card by looking for the Vendor ID 1605 in your System Profiler. Set the variable PRODUCTID to be this value.
3. Manually upload your corresponding firmware to your device by running the following:

fxload -t fx2lp -I AIOUSB/Firmware/CORRESPONDING_HEXFILE.hex -D 1605:${PRODUCTID}

Windows Installation

1. Un-install all of the AIOUSB drivers that have been installed and are associated with your device. The procedure to do this is as follows:
   (a) Plug in your card
   (b) Go to device manager , search for data acqusition products and remove the AIOUSB / ACCES I/O driver associated with your card.

(c) Click the Scan For Hardware Changes toolbar icon, or the equivalent menu item.

(d) If Windows detects and reinstalls the device, go back to step C. You may have to repeat this loop many times, depending on how (and how many times) you've installed our drivers. If the device shows up as an "unknown" of some kind, proceed to step F.

2. Download the `WinUSB 8.10 drivers` and extract them to a saved directory.
3. In Device Manager right click on the original USB Data acquisition device that should now have no driver associated with it and it should be listed as a generic USB device. Right click on the device and click "Update Driver" and instruct Windows to look for the drivers in the "Saved_Directory" from step 2.
4. After it has installed, under Device Manager the device should now be listed as a Data Acquisition product. In addition, Check the name the device ended up with; it should have a (WinUSB) tag, like "ACCES USB-IIRO-16 (WinUSB)". If it has a (CyUSB) tag instead, something went wrong, please tell us.
5. Make sure that you already have a full Cygwin installation along with the following packages:
   - python
   - python-development
   - cmake
   - gnumake
   - libusb-1.0
   - gcc
   - g++
   - swig (if you want to build wrapper language support)
6. Follow the instructions listed at either `Cmake build` or `GNU make`

Extra Language Support

In addition, to providing fully functional C Shared and Static libraries, this project also provides wrapper language support for the following languages:

- Java
- Perl
- Python
- Ruby
- PHP
- Octave
- R

1.2.4 How to build Wrapper languages

CMake

This is the easiest way to build the wrapper languages. Perform the following

```
cmake  -DCMAKE_INSTALL_PREFIX=/some/path/Dir  -DBUILD_PERL=ON -DBUILD_JAVA=ON ..
```

This will build the languages for Perl and Java. The remaining languages that can be built are Python (-DBUILD_PYTHON=ON) , Ruby (-DBUILD_RUBY=ON), PHP (-DBUILD_PHP=ON) and R (-DBUILD_R=ON) while Octave is currently not ready yet. The installation of these wrapper scripts will default be written to the CMAKE_INSTALL_PREFIX. To better customize the installation, you should use

```
ccmake -DCMAKE_INSTALL_PREFIX=/some/path/Dir ..
```

or if you have installed cmake-gui, then

```
cmake-gui -DCMAKE_INSTALL_PREFIX=/some/path/Dir ..
```

Regular Make system for building wrapper language support

Perform this step *AFTER* you have already followed the instructions for building the aiousb libraries.

Perl

```
cd AIOUSB/lib/wrappers
make -f GNUMakefile inplace_perl
cd perl
sudo make install
```

Java

You must make sure that you have the Java Development Kit installed (JDK).

```
export CPATH=$CPATH:$JAVA_HOME/include # example /usr/lib/jvm/java-7-openjdk-i386/include
cd AIOUSB/lib/wrappers
make -f GNUMakefile inplace_java
sudo cp java/{AIOUSB.jar,libaiousb.jar} $JAR_FOLDER
```

Python

```
pyver=$(python  -c 'import platform; print platform.python_version()')
cd AIOUSB/lib/wrappers
make -f GNUMakefile inplace_python
sudo cp python/build/lib.linux-$(uname -m)-${pyver}/* /usr/lib/python${pyver}/
```

Ruby

```
cd AIOUSB/lib/wrappers
make -f GNUMakefile inplace_ruby
```

Octave

```
cd AIOUSB/lib/wrappers
make -f GNUMakefile inplace_octave
```

R

```
cd AIOUSB/lib/wrappers
make -f GNUMakefile inplace_R
```

Users who wish to build web applications around the ACCES I/O Product line might consider one of these for faster development cycles. Suggestions for additional languages and features are well received and can be made to suggestions *AT* accesio *DOT* com

Sincerely,

The ACCES I/O Development team.

Chapter 2

Compiling and Installation

2.1 Compiling from source on Linux/MacOS X / UNIX

There are two ways that you can build your software on Linux/MacOS X : one way involves using the CMake tool and the other relies solely on GNU make.

- Build with CMake
- GNU make build

2.2 Compiling from source on Windows

TBD

Todo Complete the Windows port of the AIOUSB libraries

2.3 Installing the Compiled libraries

Assuming you're starting from the root directory of the distribution, installing AIOUSB consists of performing these few simple steps (logged in as 'root', of course).

```
mkdir /usr/local/include/aiousb
cp -p lib/aiousb.h /usr/local/include/aiousb
cp -p lib/libaiousb*.a /usr/local/lib
export CPATH=/usr/local/include/libusb-1.0/:/usr/local/include/aiousb
```

You can, of course, place the AIOUSB files (aiousb.h, libaiousb∗.a) elsewhere, such as in the local directory of the application program that will use AIOUSB. The above steps are suggested in order to conform to the usual Linux conventions and make the AIOUSB files available to all users and all application programs.

If you do locate the libraries and header files somewhere else, there is an environment variable you can set which is used by the sample program make files:

```
export AIOUSBLIBDIR="/path/to/the/AIOUSB_libraries/"
```

There are several variations of the AIOUSB C library. They all begin with "libaiousb" and have an extension of ".a". There are currently no shared versions of the AIOUSB C library. The library files which contain the string "cpp" contain object modules that are compiled for use with C++ (using "g++"). They are not C++ class libraries (see below for information about the genuine C++ class library). The library files which contain the string "dbg" are compiled for source level debugging with the "-ggdb" compiler option. At present, there are four variations of the AIOUSB C library: C or C++, and release or debug.

To take full advantage of the AIOUSB documentation the following additional tools should be installed to create both web documentation , pdf output and manual pages

- GNU make
- A LaTeX distribution: for instance `TeX Live` This is needed for generating LaTeX, Postscript, and PDF output.
- `the Graph visualization toolkit version 1.8.10 or higher` Needed for the include dependency graphs, the graphical inheritance graphs, and the collaboration graphs. If you compile graphviz yourself, make sure you do include freetype support (which requires the freetype library and header files), otherwise the graphs will not render proper text labels.
- For formulas in the HTML output (when MathJax is not used) or in case you do not wish to use 'pdflatex, the ghostscript interpreter is needed. You can find it at `www.ghostscript.com`.

```
cd AIOUSB-$VERSION
mkdir build
```

Chapter 3

AIOUSB C libary

3.1 Overview

The AIOUSB C language library implements the core functionality of the entire suite of libraries, supporting the capabilities of all of ACCES' USB products. (The only deficiency is that the D/A streaming functions in the USB-DA12-8A product are not currently implemented, although they will be eventually.) This library is written in C and compiled for both C (gcc) and C++ (g++) and may be used as-is with C and C++ programs. This library utilizes libusb (preferred version 1.0.6) for all USB communications. Installing

Assuming you're starting from the root directory of the distribution, installing AIOUSB consists of performing these few simple steps (logged in as 'root', of course).

```
mkdir /usr/local/include/aiousb
cp -p lib/aiousb.h /usr/local/include/aiousb
cp -p lib/libaiousb*.a /usr/local/lib
export CPATH=/usr/local/include/libusb-1.0/:/usr/local/include/aiousb
```

3.2 Other stuff

You can, of course, place the AIOUSB files (aiousb.h, libaiousb∗.a) elsewhere, such as in the local directory of the application program that will use AIOUSB. The above steps are suggested in order to conform to the usual Linux conventions and make the AIOUSB files available to all users and all application programs.

If you do locate the libraries and header files somewhere else, there is an environment variable you can set which is used by the sample program make files:

export AIOUSBLIBDIR="path to the AIOUSB libraries"

There are several variations of the AIOUSB C library. They all begin with "libaiousb" and have an extension of ".a". There are currently no shared versions of the AIOUSB C library. The library files which contain the string "cpp" contain object modules that are compiled for use with C++ (using "g++"). They are not C++ class libraries (see below for information about the genuine C++ class library). The library files which contain the string "dbg" are compiled for source level debugging with the "-ggdb" compiler option. At present, there are four variations of the AIOUSB C library: C or C++, and release or debug. Documentation

Complete documentation for the AIOUSB C library may be found in AIOUSB API Reference. This document is based on the USB Software Reference Manual, but adds considerably more detail and includes documentation for the many new functions added to the Linux implementation. This documentation is also intended to be useful as a reference for the Windows implementation of AIOUSB, with differences between the Linux and Windows implementations clearly highlighted. Compiling Programs

Assuming you have installed the AIOUSB C library according to the above instructions, compiling a program to use it is as simple as:

3.3 Compiling sample

Please see compiling for Linux / Mac or compiling on windows.

Chapter 4

AIOUSB C++ Class library

4.1 Introduction

The AIOUSB C++ Class Library is an object-oriented C++ layer that runs on top of the AIOUSB library. All access to the USB devices is through fully object-oriented C++ classes. The user never needs to call the underlying AIOUSB library, although that is possible if necessary. This C++ library supports all the features of all the USB products except the D/A streaming features of the USB-DA12-8A product, although support for those features will be provided eventually. While the underlying AIOUSB library has been thoroughly tested, this C++ library has not yet been thoroughly tested and should be considered beta software.

4.2 Packaging

As with the AIOUSB library, the AIOUSB C++ class library is packaged into several library (.a) files. libclassaiousb.a is a release version of the library and libclassaiousbdbg.a is a debug version, compiled with the "-ggdb" compiler option and with assertion checks enabled. When linking programs that use the AIOUSB C++ class library, you must not only specify the AIOUSB C++ class library on the linker command line, but the AIOUSB C library as well since the AIOUSB C++ class library uses AIOUSB. Sample Program

Below is an example of a minimalist C++ program that demonstrates how to properly initialize the library, query the device manager for devices, query an individual device for its product ID and name and then terminate use of the library. If the AIOUSB C library and the AIOUSB C++ class library are properly installed, you should be able to copy this sample program from this document, paste it into a file named test.cpp and compile it using the command shown below. This program uses the first ACCES device it finds on the bus. A "real" application would probably be looking for devices of a particular type, which can be located using one of the AIOUSB::USBDeviceManager::getDeviceByProductID(int productID) const methods.

```
#include <iostream>
#include <iomanip>
#include <USBDeviceManager.hpp>
using namespace AIOUSB;
using namespace std;
int main( int argc, char *argv[] ) {
  int result = 0;
  USBDeviceManager deviceManager;
  try {
    deviceManager.open();
    USBDeviceArray devices = deviceManager.getDeviceByProductID(
      USBDeviceManager::MIN_PRODUCT_ID,
      USBDeviceManager::MAX_PRODUCT_ID );
    if( devices.size() > 0 ) {
      USBDevice &device = *devices.at( 0 );
      cout << "Found a device with product ID " << hex << device.getProductID() << " and name \'" << device
      .getName() << "\'" << endl;
    } else
      cout << "No devices found" << endl;
    deviceManager.close();
  } catch( exception &ex ) {
    cerr << "Error \'" << ex.what() << "\' occurred while manipulating device" << endl;
    result = 1;
    if( deviceManager.isOpen() )
      deviceManager.close();
  }  // catch( ...
  return result;
}  // main()
```

4.3 Deprecated

This API is deprecated by ACCES I/O Products Inc. We are unable to devote our mental compute cycles to maintaining the C++ wrapper code for our existing C code.

We are keeping this directory for legacy customers but we urge new adopters of our Hardware and Software to use only the C API which is under ../lib. We provide better support and coverage for that library and will make sure that it can be compiled in a manner that will still suit C++ developers. In addition, if you are looking to develop an application in a language other than C or C++ we provide a wrappers directory that includes support for Java, Perl, Python, Ruby, R , Php, Octave and Matlab.

We apologize for any inconvenience this might have caused,

Sincerely,

The ACCES I/O Products Software team.

Chapter 5

C/C++ Samples

5.1 Overview

Assuming you have installed the AIOUSB C library according to the above instructions, compiling a program to use it is as simple as:

5.2 C and C++ samples for USB based acquisition cards

USB-AI16-16 samples

USB-IDIO-16 and USB-IDIO-8 samples

USB-IIRO-16 and USB-IIRO-8 samples

USB-DIO-32 samples

USB-DIO-16 samples

USB-AO16-16 samples

USB-DA12-8A samples

5.3 USB-AI16-16

These are the samples of the `USB-AI` series

extcal.c External calibration sample

continuous_mode.c Continuous Mode Samples

continuous_mode_from_json_config.c Continuous Mode sample usng a JSON config file

burst_test.c Example of a quick continuous mode sample that sets up an AIOContinuousBuf buffer and then proceeds to read data from it.

bulk_aquire_test.c Sample the demonstrates the older and unfortunately , less reliable Bulk Acquire API. This API has been effectively replaced by the continuous mode samples.

5.3.1 extcal.c

Extcal

Extcal.cpp is simple program that demonstrates using the AIOUSB C library and C++ Classlib to perform an external calibration of an ACCES I/O model USB-AI16-16A analog input board. The program is not intended to be a comprehensive demonstration and is limited to demonstrating the following features of the AIOUSB API:

- Initializing and shutting down the API – USBDeviceManager::open(), USBDeviceManager::close()
- Finding devices on the USB bus – USBDeviceManager::getDeviceByProductID()
- Configuring the board – USBDevice::setCommTimeout(), AnalogInputSubsystem::setCalMode(), AnalogInputSubsystem::setDiscardFirstSample(), AnalogInputSubsystem::setTriggerMode(), AnalogInputSubsystem::setGainCodeAndDiffMode(), AnalogInputSubsystem::setOversample()
- Installing a default calibration table – AnalogInputSubsystem::calibrate(bool,...)
- Reading the analog inputs in counts – AnalogInputSubsystem::read()
- Generating an external calibration table – AnalogInputSubsystem::calibrate(double[],...)

Todo Setup BUILDING Tag

5.3.2 continuous_mode.c

Continuous Mode

continuous_mode.cpp is simple program that demonstrates using the AIOUSB C library's Continuous mode acquisition API.

The key steps for running a continuous mode acquisition are:

1. Allocate an AIOContinuousBuf using NewAIOContinuousBuf()
2. Set the channel ranges (using AIOContinuousBufSetStartAndEndChannel), the number of oversamples (using AIOContinuousBufSetOversample) and then the gain mode for each channel and whether you will use differential or single ended mode (using AIOContinuousBufSetAllGainCodeAndDiffMode).
3. Save the settings to the AIO board using AIOContinuousBufSaveConfig().
4. Set the clock rate for the acquisition using AIOContinuousBufSetClock().
5. Start the acquisition with AIOContinuousBufInitiateCallbackAcquisition().
6. Process (ie read) the data in the AIOContinuousBuf using AIOContinuousBufReadIntegerScanCounts()

Todo Reference building tag

1. Each buf should have a device index associated with it, you must setit first
2. Setup the Config object for Acquisition, either the more complicated part in comments (BELOW) or using a simple interface.

Alternative setup for the AIOContinuousBuf oversamples, gain code and trigger modes

```
ADConfigBlock configBlock;

AIOUSB_InitConfigBlock( &configBlock,
      AIOContinuousBuf_GetDeviceIndex(buf),
      AIOUSB_FALSE );
AIOUSB_SetAllGainCodeAndDiffMode( &configBlock,
      AD_GAIN_CODE_0_5V, AIOUSB_FALSE );
AIOUSB_SetTriggerMode( &configBlock, AD_TRIGGER_SCAN |
      AD_TRIGGER_TIMER ); // 0x05
AIOUSB_SetScanRange( &configBlock, 0, 15 );

ADC_QueryCal( AIOContinuousBuf_GetDeviceIndex(buf) );

result = ADC_SetConfig( AIOContinuousBuf_GetDeviceIndex(buf),
      configBlock.registers, &configBlock.size );
```

5.3.3 continuous_mode_from_json_config.c

This C sample is simple program that demonstrates using the AIOUSB C library's Continuous mode acquisition API but makes it much easier than other samples. The end user just has to make use of a standard JSON configuration object that can stand in for the multiple calls to the AIOUSB API that are used to configure the board for acquisition.

5.3.4 burst_test.c

Overview

burst_test.c is simple program that performs a high speed continuous acquisition using the AIOUSB C library's Continuous mode acquisition API. It allows one to setup a simple AIOContinuousBuf, specify the clock rate for the acquisition ( ) , specify the number of channels that the user would like to acquire , start the acquisition and then write to the file called "output.txt".

The output file, *output.txt*, is just a Command Separated Value (csv) file that can be analyzed using R, Matlab or Excel to examine the waveforms generated.

Parts of the sample

Command line parsing

This is just the introductory code that handles command line parsing for most of the Linux and Mac based AIOUSB samples. There is a standard set of parameters that you can examine if you run

```
shell> ./burst_test --help
./burst_test - Options
        -D | --debug  ARG
             --dump
             --dumpadcconfig
        -S | --buffer_size  ARG
        -N | --num_scans  ARG
        -n | --num_channels  ARG
        -O | --num_oversamples  ARG
        -g | --gaincode  ARG
        -c | --clockrate  ARG
        -C | --calibration  ARG
        -h | --help
        -i | --index  ARG
        -R | --range  ARG
             --repeat  ARG
        -r | --reset
        -f | --outfile  ARG
        -V | --verbose
        -B | --block_size  ARG
        -T | --timing
        -q | --query
        -L | --ratelimit  ARG
        -p | --physical
             --counts
        -Y | --yaml
        -J | --json
             --jsonconfig  ARG
```

Todo Document the Command line Parsing helper library

Create a new AIOContinuousBuf

This blurb allocates a new AIOContinuousBuf buffer for reading counts, or unsigned shorts, from the Analog board.

```
    buf = (AIOContinuousBuf *)NewAIOContinuousBufForCounts(
      options.index, options.num_scans, options.num_channels );
    if( !buf ) {
      fprintf(stderr,"Can't create AIOContinuousBuf \n");
      exit(1);
    }
```

Set the device index for the AIOContinuousBuf

The following command associates the AIOContinuousBuf with a particular index. Typically you either pass in the device index to the constructor (see NewAIOContinuousBuf()) or, you can set it after the fact with this routine.

```
    AIOContinuousBufSetDeviceIndex( buf, options.index );
```

Initialize the AIOContinuousBuf

Setup the AIOContinuousBuf 's ADCConfig object for Acquisition. This is a simplified interface for an easy configuration. Alternatively, you may use functionality provided through ADC_SetConfig() to set the configuration registers and ADC_-GetConfig() to read the configuration registers.

```
    AIOContinuousBufInitConfiguration( buf );
```

Set the Clock rate / Data Acquisition speed

Setup the sampling clock rate, in this case 10000000 / 1000

```
    AIOContinuousBufSetClock( buf, options.clock_rate );
```

Start the continuous mode callback

Start the Callback that fills up the AIOContinuousBuf. This fires up an thread that performs the acquistion, while you go about doing other things.

```
    AIOContinuousBufInitiateCallbackAcquisition(buf);
```

in this example we read bytes in blocks of our core num_channels parameter. the channel order

Acquire data until completed

This part shows how to acquire data continuously until there is nore more data remaining. It makes use of the function AIOContinuousBufPending() that indicates that data is still available for acquisition.

```
    while ( AIOContinuousBufPending(buf) ) {

        if ( (scans_remaining = AIOContinuousBufCountScansAvailable(buf)
       ) > 0 ) {
```

Do something with the data

```
        } else {
            usleep(100);
        }
    }
```

5.3.5 bulk_aquire_test.c

Deprecated This is a Deprecated sample. Please look at burst_test.c, continuous_mode_callback.c or continuous_-mode_callback.c

5.4 USB-IDIO-16 and USB-IDIO-8

Overview

These are the samples of the ACCES I/O Products `USB-IDIO` series data acquisition cards.

idio_sample.c Basic sample for USB-IDIO products

idio_sample2.c Sample program demonstrating stuff

Build

5.4.1 idio_sample.c

Sample2

sample2.c is simple program that demonstrates using the AIOUSB C library's Continuous mode acquisition API.

Todo Complete this example

5.4.2 idio_sample2.c

Sample2

sample2.c is simple program that demonstrates using the AIOUSB C library's Continuous mode acquisition API.

Todo Complete this example

5.5 USB-IIRO-16 and USB-IIRO-8

Overview

These are the samples of the ACCES I/O Products `USB-IDIO` series data acquisition cards.

iiro_sample.c Sample program for the USB-IIRO-8 and USB-IIRO-16

Build

5.5.1 iiro_sample.c

iiro_sample

iiro_sample.c is simple program that demonstrates using the IIRO relay based USB product line.

Todo Complete this example

5.6 USB-DIO-32

USB-DIO-32 Sample Program Release Notes

5.6.1 Overview

This directory contains several sample programs for the USB-DIO-32 which demonstrate use of different features and libraries.

- C/C++ Language Sample - sample.cpp
- Java Sample - Sample.java

5.6.2 C/C++ Language Sample

Sample.cpp is a simple program to demonstrate using the AIOUSB module to control an ACCES I/O model USB-DIO-32 digital I/O board. The program is not intended to be a comprehensive demonstration and is limited to demonstrating the following features of the AIOUSB API:

- Initializing and shutting down the API – AIOUSB_Init(), AIOUSB_Exit()
- Identifying devices on the USB bus – QueryDeviceInfo()
- Obtaining the serial number of a device on the bus – GetDeviceSerialNumber()
- Configuring the board – DIO_Configure()
- Reading the digital inputs – DIO_ReadAll()
- Writing the digital outputs – DIO_WriteAll()

For easy identification, the source code lines prefixed with the comment API denote calls to the AIOUSB API.

Building

Before building the program, make sure the libusb module is installed. Also refer to the comments at the top of sample.-cpp for additional details.

Also, make sure that the ACCES I/O AIOUSB module is installed (see Installing And Using AIOUSB Library).

The simplest way to build the sample program is to type make at the command line. The sample program is the default target in Makefile. Optionally, one can manually compile the program with the command:

g++ sample.cpp -laiousb -lusb-1.0 -o sample

Executing

Before executing the sample program, make sure the Linux system is configured to automatically detect ACCES I/O devices plugged into the USB bus and upload the appropriate firmware to those devices. The files that support this automatic configuration have recently been updated and new documentation prepared. Please refer to Configuring ACCES I/O USB Devices To Work Under Linux for details.

To execute the program, attach two USB-DIO-32 digital I/O boards to the USB bus and verify that their LEDs turn on, indicating that firmware has been successfully uploaded to the boards. Then simply type ./sample at the command line. There are no command line arguments to worry about. The program will search for the first two USB-DIO-32 digital I/O boards on the USB bus. (If you have only one board and want to use this sample program, simply change the DEVICE-S_REQUIRED constant at the top of the sample program to 1 and recompile the sample program by typing make at the command prompt.) If the sample program fails to find two boards, it will print an error message and quit. If it finds two such boards, the following output will appear:

```
USB-DIO-32 sample program version 1.17, 26 November 2009
  AIOUSB library version 1.84, 22 December 2009
  This program demonstrates communicating with 2 USB-DIO-32 devices on
  the same USB bus. For simplicity, it uses the first 2 such devices
  found on the bus.
ACCES devices found:
  Device at index 0:
    Product ID: 0x8040
    Product name: USB-AI16-16A
    Number of digital I/O bytes: 2
    Number of counters: 1
  Device at index 1:
    Product ID: 0x8001
    Product name: USB-DIO-32
    Number of digital I/O bytes: 4
    Number of counters: 3
  Device at index 2:
    Product ID: 0x8001
    Product name: USB-DIO-32
    Number of digital I/O bytes: 4
    Number of counters: 3
Serial number of device at index 1: 40e391cdff3dd1bb
Serial number of device at index 2: 40e391cdf95aa30c
Device at index 1 successfully configured
Device at index 2 successfully configured
Read the following values from device at index 1: 0x11 0x22 0x33 0x44 (correct)
Read the following values from device at index 2: 0x66 0x65 0x64 0x63 (correct)
Writing patterns to devices: 0 0x10 0x20 0x30 0x40 0x50 0x60 0x70 0x80 0x90 0xa0
 0xb0 0xc0 0xd0 0xe0 0xf0
All patterns written were read back correctly
```

The sample program prints out a list of all the ACCES devices found on the USB bus and then proceeds to exercise the two USB-DIO-32 boards found. Notice in the above example, the sample program also found a model USB-AI16-16A on the bus. The entire demonstration takes about 16 seconds.

5.6.3 Java Sample

Sample.java is a Java implementation of the above sample program. It demonstrates use of the Java class library. Refer to AIOUSB Java Class Library Reference for detailed documentation on the Java class library.

Building

The prerequisites for building Sample.class are that the `Java Development Kit (JDK)` must be installed. In addition, the AIOUSB Java library (aiousb.jar) must be installed somewhere on your system. To compile the program, either use the supplied Makefile or use the command:

```
javac -cp ../../java/aiousb.jar Sample.java
```

This sample program can demonstrate writing to the EEPROM. That demonstration is disabled by default, but if you wish to enable it, simply edit Sample.java and set the variable named DEMO_EEPROM_WRITE to true.

Executing

To execute the program, attach a USB-DIO-32 analog input board to the USB bus and verify that its LED turns on, indicating that firmware has been successfully uploaded to the board. Then type the command:

```
java -cp ../../java/aiousb.jar:. Sample
```

Notice that multiple class paths are specified in the above command: the path to aiousb.jar and ".", which represents the class path of Sample.class (assuming that it is the current directory).

Alternatively, assuming you used the make file to build the program, you can run it with the command:

```
java -jar Sample.jar
```

There are no command line arguments to worry about. The program will search for the first USB-DIO-32 analog input board on the USB bus. If it fails to find such a board, it will print an error message and quit. If it finds such a board, the following output will appear:

```
USB-DIO-32 sample program version: 1.3, 25 December 2009
  AIOUSB Java library version: 1.6, 17 December 2009
  AIOUSB library version: 1.84, 22 December 2009
  JRE version: 1.6.0_17
  OS version: Linux amd64 2.6.31.5-0.1-custom
  This program demonstrates controlling a USB-DIO-32 device on
  the USB bus. For simplicity, it uses the first such device found
  on the bus.
ACCES devices found:
  Device at index 0
    Product ID: 0x8040
    Product name: USB-AI16-16A
    Serial number: 0x40e38f15d5c94894
    Number of A/D channels: 16
    Number of MUXed A/D channels: 16
    Number of digital I/O ports: 2
    Number of digital I/O channels: 16
    Number of tristate groups: 0
    Number of tristate channels: 0
    Number of counter blocks: 1
    Number of counters: 3
  Device at index 1
    Product ID: 0x8001
    Product name: USB-DIO-32
    Serial number: 0x40e39acf9e8dd7cc
    Number of digital I/O ports: 4
    Number of digital I/O channels: 32
    Number of tristate groups: 0
    Number of tristate channels: 0
    Number of counter blocks: 3
    Number of counters: 9
EEPROM contents:
[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1]
Configuring digital I/O ... successful
Turning all outputs on: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
      31
Turning all outputs off: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
      31
Turning all outputs on ... successful
Turning all outputs off ... successful
```

5.7 USB-DIO-16

USB-DIO-16A Sample Program Release Notes

5.7.1 Overview

This directory contains several sample programs for the USB-DIO-16A which demonstrate use of different features and libraries.

Basic AIOUSB Sample - sample.cpp, receiver.cpp

AIOUSB Java Sample - Sample.java

5.7.2 Basic AIOUSB Sample

Sample.cpp and receiver.cpp are a pair of simple programs to demonstrate using the AIOUSB module to control an ACCES I/O model USB-DIO-16A digital I/O board. The program is not intended to be a comprehensive demonstration and is limited to demonstrating the following features of the AIOUSB API:

- Initializing and shutting down the API – AIOUSB_Init(), AIOUSB_Exit()
- Identifying devices on the USB bus – QueryDeviceInfo()
- Obtaining the serial number of a device on the bus – GetDeviceSerialNumber()
- Configuring the board – DIO_ConfigureEx()
- Writing to, and reading from a digital I/O stream – DIO_StreamSetClocks(), DIO_StreamOpen(), DIO_Stream-Frame(), DIO_StreamClose()

For easy identification, the source code lines prefixed with the comment / ∗API∗ / denote calls to the AIOUSB API.

5.7.3 Building

Before building the program, make sure the libusb module is installed. Also refer to the comments at the top of sample.-cpp for additional details.

Also, make sure that the ACCES I/O AIOUSB module is installed (see Installing And Using AIOUSB Library).

The simplest way to build the sample program is to type make at the command line. The sample program is the default target in Makefile. Optionally, one can manually compile the program with the commands:

g++ sample.cpp -laiousb -lusb-1.0 -o sample g++ receiver.cpp -laiousb -lusb-1.0 -o receiver

5.7.4 Executing

Before executing the sample program, make sure the Linux system is configured to automatically detect ACCES I/O devices plugged into the USB bus and upload the appropriate firmware to those devices. The files that support this automatic configuration have recently been updated and new documentation prepared. Please refer to Configuring ACCES I/O USB Devices To Work Under Linux for details.

To execute the program, attach two USB-DIO-16A digital I/O boards to the USB bus and verify that their LEDs turn on, indicating that firmware has been successfully uploaded to the boards. Then simply type ./sample at the command line. There are no command line arguments to worry about. The program will search for the first two USB-DIO-16A digital I/O boards on the USB bus. If the sample program fails to find two boards, it will print an error message and quit. If it finds two such boards, the following output will appear:

```
USB-DIO-16A sample program version 1.9, 29 January 2010
  AIOUSB library version 1.88, 18 January 2010
  This program demonstrates high speed streaming between 2 USB-DIO-16A
  devices on the same USB bus. For simplicity, it uses the first 2 such
  devices found on the bus.
ACCES devices found:
  Device at index 0:
    Product ID: 0x800f
    Product name: USB-DIO-16A
    Number of digital I/O bytes: 4
    Number of counters: 0
  Device at index 1:
    Product ID: 0x800f
    Product name: USB-DIO-16A
    Number of digital I/O bytes: 4
    Number of counters: 0
Sending device at index 0, serial number 40e3a0d0c488856d
Receiving device at index 1, serial number 40e3a0d0a53149dd
Stream clock for device at index 0 set to 1000432.0 Hz
1024000 point frame successfully written to device at index 0
1024000 point frame successfully read from device at index 1
```

The sample program prints out a list of all the ACCES devices found on the USB bus and then proceeds to exercise the two USB-DIO-16A boards found. Basically, sample executes receiver as a child process to receive the stream data from one of the two devices. Sample then transmits the stream data to the other device. The entire demonstration takes a couple of seconds.

Important: this sample program requires that the two USB-DIO-16A devices be electrically connected together so that one device can transmit to the other. This connection is accomplished by means of a standard 68-pin SCSI cable attached to the J1 connector of each device. Contact ACCES for more information or to purchase such a cable (part number C68PS18L).

5.7.5 AIOUSB Java Sample

Sample.java is a Java implementation of the above sample program. It demonstrates use of the Java class library, which utilizes the AIOUSB C-language library. Refer to AIOUSB Java Class Library Reference for detailed documentation on the Java class library.

Building

The prerequisites for building Sample.jar are that the Java Development Kit (JDK) must be installed. In addition, the AIOUSB Java library (aiousb.jar) must be installed somewhere on your system. To compile the program, either use the supplied Makefile or use the command:

```
javac -cp ../../java/aiousb.jar Sample.java
```

Executing

Like the C-language sample program above, this Java sample program requires two USB-DIO-16A devices, hooked together by means of a SCSI cable. The main difference between the Java program and the C program, aside from the languages and libraries used, is that the Jave version is multithreaded, whereas the C program utilizes two processes.

Assuming you have two USB-DIO-16A devices up and running (as indicated by their illuminated LEDs) and hooked together by means of a SCSI cable, type the following command to execute the sample program:

```
java -jar Sample.jar
```

or

```
java -cp ../../java/aiousb.jar:. Sample
```

There are no command line arguments to worry about. The program will search for the first two USB-DIO-16A devices on the USB bus. If it fails to find such a board, it will print an error message and quit. If it finds two such devices, the following output will appear:

```
USB-DIO-16A sample program version: 1.3, 29 January 2010
  AIOUSB Java library version: 1.7, 18 January 2010
  AIOUSB library version: 1.88, 18 January 2010
  JRE version: 1.6.0_17
  OS version: Linux amd64 2.6.31.5-0.1-custom
  This program demonstrates high speed streaming between 2 USB-DIO-16A
  devices on the same USB bus. For simplicity, it uses the first 2 such
  devices found on the bus.
ACCES devices found:
  Device at index 0
    Product ID: 0x800f
    Product name: USB-DIO-16A
    Serial number: 0x40e3a0d0c488856d
    Number of digital I/O ports: 4
    Number of digital I/O channels: 32
    Number of tristate groups: 2
    Number of tristate channels: 16
    Digital I/O streaming capability installed
  Device at index 1
    Product ID: 0x800f
    Product name: USB-DIO-16A
    Serial number: 0x40e3a0d0a53149dd
    Number of digital I/O ports: 4
    Number of digital I/O channels: 32
    Number of tristate groups: 2
    Number of tristate channels: 16
    Digital I/O streaming capability installed
Successfully sent 1024000 samples
Waiting for data to be received ...
Successfully received 1024000 samples
```

5.8 USB-AO16-16

USB-AO16-16A Sample Program Release Notes

5.8.1 Overview

Sample.cpp is a simple program to demonstrate using the AIOUSB module to control an ACCES I/O model USB-AO16-16A analog output board. The program is not intended to be a comprehensive demonstration and is limited to demonstrating the following features of the AIOUSB API:

- Initializing and shutting down the API – AIOUSB_Init(), AIOUSB_Exit()
- Identifying devices on the USB bus – QueryDeviceInfo()
- Obtaining the serial number of a device on the bus – GetDeviceSerialNumber()
- Setting the output range – DACSetBoardRange()
- Writing to a single D/A channel – DACDirect()
- Writing to multiple D/A channels – DACMultiDirect()

For easy identification, the source code lines prefixed with the comment API denote calls to the AIOUSB API.

Building

Before building the program, make sure the libusb module is installed. Also refer to the comments at the top of sample.-cpp for additional details.

Also, make sure that the ACCES I/O AIOUSB module is installed (see Installing And Using AIOUSB Library).

The simplest way to build the sample program is to type make at the command line. The sample program is the default target in Makefile. Optionally, one can manually compile the program with the command:

g++ sample.cpp -laiousb -lusb-1.0 -o sample

Executing

Before executing the sample program, make sure the Linux system is configured to automatically detect ACCES I/O devices plugged into the USB bus and upload the appropriate firmware to those devices. The files that support this automatic configuration have recently been updated and new documentation prepared. Please refer to Configuring ACCES I/O USB Devices To Work Under Linux for details.

To execute the program, attach a USB-AO16-16A analog output board to the USB bus and verify that its LED turns on, indicating that firmware has been successfully uploaded to the board. Then simply type ./sample at the command line. There are no command line arguments to worry about. The program will search for the first USB-AO16-16A analog output board on the USB bus. If it fails to find such a board, it will print an error message and quit. If it finds such a board, the following output will appear:

```
USB-AO16-16A sample program version 1.13, 26 November 2009
  AIOUSB library version 1.84, 22 December 2009
  This program demonstrates controlling a USB-AO16-16A device on
  the USB bus. For simplicity, it uses the first such device found
  on the bus.
ACCES devices found:
  Device at index 0:
    Product ID: 0x8040
    Product name: USB-AI16-16A
    Number of digital I/O bytes: 2
    Number of counters: 1
  Device at index 1:
    Product ID: 0x8060
    Product name: USB-AO16-16A
    Number of digital I/O bytes: 2
    Number of counters: 0
Serial number of device at index 1: 40e39396fc4198c0
D/A output range successfully set
32767 D/A counts successfully output to channel 0
D/A counts successfully output to 16 channels simultaneously
The sample program prints out a list of all the ACCES devices found on the USB bus and then proceeds to
      exercise the first USB-AO16-16A board found. Notice in the above example, the sample program also found a model
       USB-AI16-16A on the bus.
```

5.9 USB-DA12-8A

USB-DA12-8A Sample Program Release Notes

5.9.1 Overview

This directory contains several sample programs for the USB-DA12-8A which demonstrate use of different features and libraries.

- AIOUSB C Sample - sample.cpp
- AIOUSB C++ Sample - SampleClass.cpp
- AIOUSB Java Sample - Sample.java

5.9.2 AIOUSB C Sample

Sample.cpp is a simple program to demonstrate using the AIOUSB module to control an ACCES I/O model USB--DA12-8A analog output board. The program is not intended to be a comprehensive demonstration and is limited to demonstrating the following features of the AIOUSB API:

- Initializing and shutting down the API – AIOUSB_Init(), AIOUSB_Exit()
- Identifying devices on the USB bus – QueryDeviceInfo()
- Obtaining the serial number of a device on the bus – GetDeviceSerialNumber()
- Writing to a single D/A channel – DACDirect()
- Writing to multiple D/A channels – DACMultiDirect()
- For easy identification, the source code lines prefixed with the comment \/∗API∗\/ denote calls to the AIOUSB API.

Building

Before building the program, make sure the libusb module is installed. Also refer to the comments at the top of sample.-cpp for additional details.

Also, make sure that the ACCES I/O AIOUSB module is installed (see Installing And Using AIOUSB Library).

The simplest way to build the sample program is to type make at the command line. The sample program is the default target in Makefile. Optionally, one can manually compile the program with the command:

g++ sample.cpp -laiousbcpp -lusb-1.0 -o sample

Executing

Before executing the sample program, make sure the Linux system is configured to automatically detect ACCES I/O devices plugged into the USB bus and upload the appropriate firmware to those devices. The files that support this automatic configuration have recently been updated and new documentation prepared. Please refer to Configuring ACCES I/O USB Devices To Work Under Linux for details.

To execute the program, attach a USB-DA12-8A analog output board to the USB bus and verify that its LED turns on, indicating that firmware has been successfully uploaded to the board. Then simply type ./sample at the command line.

There are no command line arguments to worry about. The program will search for the first USB-DA12-8A analog output board on the USB bus. If it fails to find such a board, it will print an error message and quit. If it finds such a board, the following output will appear:

```
USB-DA12-8A sample program version 1.1, 29 January 2010
  AIOUSB library version 1.88, 18 January 2010
  This program demonstrates controlling a USB-DA12-8A device on
  the USB bus. For simplicity, it uses the first such device found
  on the bus.
ACCES devices found:
  Device at index 0:
    Product ID: 0x4002
    Product name: USB-DA12-8A
    Number of digital I/O bytes: 0
    Number of counters: 0
Serial number of device at index 0: 40e3a0d0a78887c2
Device properties successfully retrieved
2047 D/A counts successfully output to channel 0
D/A counts successfully output to 8 channels simultaneously
```

The sample program prints out a list of all the ACCES devices found on the USB bus and then proceeds to exercise the first USB-DA12-8A board found.

5.9.3 AIOUSB C++ Sample

SampleClass.cpp is a C++ implementation of the above sample program. It demonstrates use of the C++ class library, which utilizes the AIOUSB C-language library. Refer to AIOUSB C++ Class Library Reference for detailed documentation on the C++ class library.

Building

The prerequisites for building SampleClass are the same as for sample described above. In addition, the C++ class libraries must be installed and be accessible in the include path and linker library path. Once these requirements are satisfied, you can build the sample program with the supplied Makefile.

Executing

Assuming you have an USB-DA12-8A device up and running (as indicated by its illuminated LED), type the following command to execute the sample program:

./SampleClass

There are no command line arguments to worry about. The program will search for the first USB-DA12-8A device on the USB bus. If it fails to find such a board, it will print an error message and quit. If it finds such a device, the following output will appear:

```
USB-DA12-8A sample program version 1.1, 29 January 2010
  AIOUSB C++ class library version 1.8, 18 January 2010
  AIOUSB library version 1.88, 18 January 2010

  This program demonstrates controlling a USB-DA12-8A family device on
  the USB bus. For simplicity, it uses the first such device found
  on the bus and supports these product IDs: USB-DA12-8A-A, USB-DA12-8A
ACCES devices found:
  Device at index 0:
    Product ID: 0x4002
    Product name: USB-DA12-8A
    Serial number: 0x40e3a0d0a78887c2
    Number of D/A channels: 8
    D/A count range: 0-fff
Found device 'USB-DA12-8A' with serial number 40e3a0d0a78887c2
2047 D/A counts successfully output to channel 0
Multiple D/A counts successfully output to 8 channels
5 volts (3071 D/A counts) successfully output to channel 0
Multiple volts successfully output to 8 channels
```

5.9.4 AIOUSB Java Sample

Sample.java is a Java implementation of the above sample program. It demonstrates use of the Java class library, which utilizes the AIOUSB C-language library. Refer to AIOUSB Java Class Library Reference for detailed documentation on the Java class library.

Building

The prerequisites for building Sample.jar are that the Java Development Kit (JDK) must be installed. In addition, the AIOUSB Java library (aiousb.jar) must be installed somewhere on your system. To compile the program, either use the supplied Makefile or use the command:

javac -cp ../../java/aiousb.jar Sample.java

Executing

Assuming you have an USB-DA12-8A device up and running (as indicated by its illuminated LED), type the following command to execute the sample program:

```
java -jar Sample.jar
```

or

```
java -cp ../../java/aiousb.jar:. Sample
```

There are no command line arguments to worry about. The program will search for the first USB-DA12-8A device on the USB bus. If it fails to find such a board, it will print an error message and quit. If it finds such a device, the following output will appear:

```
USB-DA12-8A sample program version: 1.1, 29 January 2010
  AIOUSB Java library version: 1.7, 18 January 2010
  AIOUSB library version: 1.88, 18 January 2010
  JRE version: 1.6.0_17
  OS version: Linux amd64 2.6.31.5-0.1-custom
  This program demonstrates controlling a USB-DA12-8A device on
  the USB bus. For simplicity, it uses the first such device found
  on the bus.
ACCES devices found:
  Device at index 0
    Product ID: 0x4002
    Product name: USB-DA12-8A
    Serial number: 0x40e3a0d0a78887c2
    Number of D/A channels: 8
    D/A count range: 0-fff
Found device 'USB-DA12-8A' with serial number 40e3a0d0a78887c2
2047 D/A counts successfully output to channel 0
Multiple D/A counts successfully output to 8 channels
5.0 volts (3071 D/A counts) successfully output to channel 0
Multiple volts successfully output to 8 channels
```

Chapter 6

Wrappers

6.1 Overview

Assuming you have installed the AIOUSB C library according to the above instructions, compiling a program to use it is as simple as:

Building Wrappers

Todo Complete Wrapper Doxygen page

6.2 Building Wrappers

This directory contains the wrapper scripts for a number of scripting languages. Before trying to build any of these you MUST first source the file in ../.. . The instructions are for the Bash shell and should work for Zsh, Ash and Ksh.

1. Setup build variables

shell > cd ../.. shell > source sourceme.sh shell > cd -

1. Make sure that you have build libaiousb∗ . To do this you should have run "make" in the directories $AIOUSB_R-OOT/lib and $AIOUSB_ROOT/classlib.
2. Now you have setup your AIOUSB_ROOT envvar you are ready to start building the various languages. If you want to build all three wrapper scripts by default and install them, just run

sudo make -f GNUMakefile all

This will build each language and install them.

This directory contains the wrapper scripts for a number of scripting languages. Before trying to build any of these you MUST first source the file in ../.. . The instructions are for the Bash shell and should work for Zsh, Ash and Ksh.

1. Setup build variables

shell > cd ../.. shell > source sourceme.sh shell > cd -

1. Make sure that you have build libaiousb∗ . To do this you should have run "make" in the directories $AIOUSB_R-OOT/lib and $AIOUSB_ROOT/classlib.
2. Now you have setup your AIOUSB_ROOT envvar you are ready to start building the various languages. If you want to build all three wrapper scripts by default and install them, just run

sudo make -f GNUMakefile all

This will build each language and install them.

1.

Chapter 7

Firmware

7.1 Introduction

Configuring ACCES I/O USB Devices To Work Under Linux

7.2 Overview

This document explains how to set up and use ACCES USB devices in Linux. This document describes only configuring Linux to recognize connected devices and upload firmware to the devices. For information on using the AIOUSB library, refer to Installing And Using AIOUSB Library.

Besides merely attaching an ACCES USB device to the computer by means of a USB cable, ACCES' USB devices also require that firmware be uploaded to them before they can function properly. There are two means to accomplish this: automatically and manually, both of which are described below.

7.3 Automatic Device Initialization

With Linux's udev feature, the operating system can be configured to automatically upload the correct firmware whenever an ACCES USB device is plugged into the computer. This is definitely the preferred method of uploading firmware to the devices since it's automatic.

Setting up the operating system to automatically upload the firmware to the devices is exceedingly simple, consisting of just two simple steps, described below. The steps below generally must be performed as the 'root' user.

7.4 Copy Firmware Files To Share Directory

The first step is to copy the ACCES device firmware files (.hex) somewhere on the system. On some Linux systems, /usr/share/usb/ is already intended for this purpose, so that's the location we recommend. The commands to copy the firmware files are simply:

```
mkdir -p /usr/share/usb # if the directory does not already exist
cp -p *.hex /usr/share/usb/
chown root:root /usr/share/usb/*.hex # optional
chmod 444 /usr/share/usb/*.hex # optional
```

You may, of course, put the firmware files anywhere on your system, but if you put them somewhere other than the default of /usr/share/usb/, you will have to modify the 10-acces_usb.rules file and change all occurrences of "/usr/share/usb/" to the location where the firmware files reside. If you intend to use the accesloader.pl script, then it will also have to modified similarly.

Assuming you have copied the firmware files to the default location, a directory listing of /usr/share/usb/ should look similar to this:

```
shell> ls /usr/share/usb/
total 244
drwxr-xr-x   2 root root  4096 2009-12-18 19:41 ./
drwxr-xr-x 476 root root 20480 2009-12-12 07:17 ../
-r--r--r--   1 root root  4026 2009-10-23 19:04 a3load.hex
-r--r--r--   1 root root 10657 2009-06-16 13:00 PICO-DIO16RO8.hex
-r--r--r--   1 root root 20787 2009-12-22 14:27 USB-AI16-16.hex
-r--r--r--   1 root root 18190 2009-11-10 12:39 USB-AO16-16.hex
-r--r--r--   1 root root 13683 2009-07-22 12:25 USB-CTR-15.hex
-r--r--r--   1 root root  4383 2009-04-28 09:41 USB-DA12-8A.hex
-r--r--r--   1 root root  4285 2008-12-02 15:19 USB-DA12-8E.hex
-r--r--r--   1 root root 16972 2009-05-14 11:16 USB-DIO-16A.hex
-r--r--r--   1 root root 12333 2009-12-22 14:24 USB-DIO-32.hex
-r--r--r--   1 root root 12589 2009-11-06 14:53 USB-DIO-48.hex
-r--r--r--   1 root root 11151 2009-02-25 16:37 USB-DIO-96.hex
-r--r--r--   1 root root 13149 2009-12-04 13:53 USB-IDIO-16.hex
```

```
-r--r--r--   1 root root 11694 2008-03-17 12:51 USB-IIRO-16.hex
-r--r--r--   1 root root 11139 2006-04-25 11:10 USB-IIRO4-2SM.hex
-r--r--r--   1 root root 11139 2006-04-25 11:10 USB-IIRO4-COM.hex
-r--r--r--   1 root root 10657 2009-06-16 13:04 USBP-DIO16RO8.hex
```

(The file a3load.hex shown above is part of the fxload package and should be left alone.)

7.5 Copy Udev Rules File To System Directory

The second step to automatically load firmware into the devices is to add a udev rules file to the system, which you can do using the command:

```
cp -p 10-acces_usb.rules /etc/udev/rules.d/
chown root:root /etc/udev/rules.d/10-acces_usb.rules (optional)
chmod 444 /etc/udev/rules.d/10-acces_usb.rules (optional)
```

A directory listing of /etc/udev/rules.d/ should look similar to this:

```
shell> /bin/ls -l /etc/udev/rules.d/
total 348
drwxr-xr-x 2 root root   4096 2010-01-22 17:35 ./
drwxr-xr-x 3 root root   4096 2009-10-27 01:56 ../
-r--r--r-- 1 root root  26983 2009-12-04 19:34 10-acces_usb.rules
-rw-r--r-- 1 root root    357 2009-10-23 21:11 40-alsa.rules
-rw-r--r-- 1 root root   2230 2009-10-19 11:09 40-lomoco.rules
-rw-r--r-- 1 root root    218 2009-08-04 15:03 40-xend.rules
-rw-r--r-- 1 root root    750 2009-10-27 09:18 40-xen.rules
-rw-r--r-- 1 root root     99 2009-10-23 21:20 41-soundfont.rules
...
```

Once these two simple steps are completed, plugging an ACCES USB device into the system should result in its firmware being automatically uploaded and the device being made available for use.

7.6 Troubleshooting The Udev Rules File

It seems that some versions of Linux (older ones?) prefer referring to device nodes with $ENV{DEVNAME}, while other versions of Linux (newer ones?) prefer $tempnode. So if one form doesn't work, try the other. The file 10-acces_-usb.rules uses the $tempnode syntax and the file 10-acces_usb.alt.rules uses the $ENV{DEVNAME} syntax. Also, the MODE= action doesn't seem to work on some versions of Linux, which is why chmod is used instead.

7.7 Manually Uploading Firmware to USB Devices

If automatic device configuration with udev doesn't work, ACCES USB devices can be initialized manually, using the accesloader.pl script. This script must be run with 'root' privileges and will display something like the following on the screen:

This script will upload the appropriate firmware to any ACCES USB devices that are found on the system. If firmware is uploaded to any devices, then the script will pause for five seconds before attempting to make all ACCES USB devices on the system usable by users other than root. This script must be run with root privileges.

```
fxload -t fx2 -D /dev/bus/usb/008/006 -I /usr/share/usb/USB-AI16-16.hex
chmod 0666 /dev/bus/usb/008/007
```

In the above example a model USB-AI16-16A device was detected, the appropriate firmware was uploaded to it and it was made readable and writable by all users.

7.8 Minimum Required Files

The table below summarizes the files and utilities required for automatic and manual configuration of ACCES' USB devices.

Required Files and Utilities		
Files	**Automatic configuration**	**Manual configuration**
∗.hex files copied to /usr/share/usb/	X	X
10-acces_usb.rules file copied to /etc/udev/rules.d/	X	

accesloader.pl		X
fxload [1]	X	X
chmod	X	X
lsusb		X
perl [2]		X

1. In order to upload firmware to USB devices you must have the fxload package installed on your system. To check if you have fxload installed on your system simply type fxload -V or /sbin/fxload -V on the command line. If fxload is installed on your system you will see version information displayed on your screen. If fxload is not installed on your system you can find more information at `http://linux-hotplug.sourceforge.net/` (click on the "Downloads" link or use this `link` to download fxload). You can also check the package manager for your Linux distribution to see if the fxload package is available for installation.
2. A recent version of perl is required, with the switch module installed.

Chapter 8

LIBUSB Overview

8.1 Overview

Somet stuff

8.2 Libusb Other Stuff

Assuming you have installed the AIOUSB C library according to the above instructions, compiling a program to use it is as simple as:

8.3 Compiling sample

```
g++ -pthread -fPIC sample.cpp -laiousbcpp -lusb-1.0 -o sample (C++)
```

or

```
gcc -std=gnu99 -D_GNU_SOURCE -pthread -fPIC sample.c -laiousb -lusb-1.0 -lm -o sample (C)
```

Chapter 9

README

Chapter 10

How to run read_channels_test_java

“‘bash curdir= cd ../../.. source sourceme.sh cd ${AIO_LIB_DIR} && make cd ${AIO_LIB_DIR}/wrappers/java && make -f GNUMakefile inplace_java cd ${curdir} make jar java -jar ∗.jar -N 10000 “‘

or using Gradle

“‘bash gradle fatJar java -jar build/libs/read_channels_test-all-1.0.jar -N 1000000 “‘

Chapter 11

How to run

cd into a directory and read the README.md file for how to build that sample

Chapter 12

How to run extcal.java

“‘bash curdir= cd ../../.. source sourceme.sh cd ${AIO_LIB_DIR} && make cd ${AIO_LIB_DIR}/wrappers/java && make -f GNUMakefile inplace_java cd ${curdir} make jar java -jar ∗.jar -N 10000 “‘

or using Gradle

“‘bash gradle fatJar java -jar build/libs/extcal-all-1.0.jar -N 1000000 “‘

Chapter 13

Native Utils

A simple library class which helps with loading dynamic libraries stored in the JAR archive. These libraries usualy contain implementation of some methods in native code (using JNI - Java Native Interface).

Notes

- The temporary file is stored into temp directory specified by java.io.tmpdir (by default it's the operating system's temporary directory). It should be automatically deleted when the application exits.
- Although the code has some try-finally section (to be sure that streams are closed properly in case an exception is thrown), it does not catch exceptions. The exception has to be handled by the application. I belive this approach is cleaner and has some benefits.

Usage

To load the dynamic library, just make sure it is packed inside the JAR archive and call method loadLibraryFromJar:

```
import cz.adamh.NativeUtils;

public class HelloJNI {
    static {
        try {
            NativeUtils.loadLibraryFromJar("/resources/libHelloJNI.so");
        } catch (IOException e) {
            // This is probably not the best way to handle exception :-)
            e.printStackTrace();
        }
    }

    public native void hello();
}
```

More information

More information can be found in accompanying blog post.

Chapter 14

How to run read_channels_test_java

“‘bash curdir= cd ../../.. source sourceme.sh cd ${AIO_LIB_DIR} && make cd ${AIO_LIB_DIR}/wrappers/java && make -f GNUMakefile inplace_java cd ${curdir} make jar java -jar ∗.jar -N 10000 “‘

or using Gradle

“‘bash gradle fatJar java -jar build/libs/read_channels_test-all-1.0.jar -N 1000000 “‘

Chapter 15

How to run

cd into a directory and read the README.md file for how to build that sample

Chapter 16

Todo List

Global AIOBufIteratorGetValue (AIOBufIterator ∗biter)

make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

Global AIOChannelMaskToString (AIOChannelMask ∗mask)

Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

Global AIOChannelMaskToString (AIOChannelMask ∗mask)

Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

File AIOContinuousBuffer.c

Make the number of channels in the ContinuousBuffer match the number of channels in the config object

Make the number of channels in the ContinuousBuffer match the number of channels in the config object

Global AIOContinuousBufReset (AIOContinuousBuf ∗buf)

Fix this to use condition variable

Global AIODeviceTablePopulateTable (void)

Rely on Global Header files for the functionality of devices / cards as opposed to hard coding

Note

populate device table so users can use diFirst and diOnly immediately; be sure to call PopulateDeviceTable() after 'aiousbInit = AIOUSB_INIT_PATTERN;'

File AIOUSB_Properties.c

Implement a friendly FindDevices() function as well as FindDeviceByCriteria() function to replace all of the standard looping while (deviceMask != 0)...

Page burst_test.c

Document the Command line Parsing helper library

Page Compiling and Installation

Complete the Windows port of the AIOUSB libraries

Page continuous_mode.c

Reference building tag

Global ConvertCountsToVoltsFunction (void ∗object)

Ensure that copying matches the actual size of the data

Global CreateSmartBuffer (unsigned long DeviceIndex)

Replace 16 with correct channels returned by probing the device

Global DeviceTableAtIndex_Lock (unsigned long DeviceIndex)

Replace AIOUSB_Lock() with thread safe lock on a per device index basis

Insert correct error messages into global error string in case of failure

Class DIOBuf

Provide Binary operators such as AND, OR, And Not between two different DIOBuf's

Page extcal.c

Setup BUILDING Tag

Page idio_sample.c

Complete this example

Page idio_sample2.c

Complete this example

Page iiro_sample.c

Complete this example

Global NewAIOChannelMaskFromStr (const char ∗bitfields)

Add smarter error checking

Page Wrappers

Complete Wrapper Doxygen page

Chapter 17

Deprecated List

Page bulk_aquire_test.c

This is a Deprecated sample. Please look at burst_test.c, continuous_mode_callback.c or continuous_mode_-callback.c

Global DIO_ReadIntoDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf) ACCES_DEPRECATED("Please use DI-O_ReadAllToDIOBuf")

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

Global DIO_ReadIntoDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf) ACCES_DEPRECATED("Please use DI-O_ReadAllToDIOBuf")

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

Chapter 18

Namespace Index

18.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 19

Hierarchical Index

19.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 20

Data Structure Index

20.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 21

File Index

21.1 File List

Here is a list of all files with brief descriptions:

Chapter 22

Namespace Documentation

22.1 AIOUSB Namespace Reference

Data Structures

- class AI16_DataPoint

    Class AI16_DataPoint represents a single data point captured from a USB_AI16_Family device.
- class AI16_DataPointArray
- class AI16_DataSet

    Class AI16_DataSet represents a data set captured from a USB_AI16_Family device.
- class AI16_InputRange
- class AnalogInputSubsystem

    Class AnalogInputSubsystem represents the analog input subsystem of a device.
- class AnalogIORange

    Class AnalogIORange helps manage analog I/O range settings and provides voltage-count conversion utilities.
- class AnalogOutputSubsystem

    Class AnalogOutputSubsystem is the superclass of the analog output subsystem of a device.
- class AO16_AnalogOutputSubsystem

    Class AO16_AnalogOutputSubsystem represents the analog output subsystem of a device.
- class AO16_OutputRange
- class Counter

    Class Counter represents a single counter/timer.
- class CounterList
- class CounterSubsystem

    Class CounterSubsystem represents the counter/timer subsystem of a device.
- class DA12_AnalogOutputSubsystem

    Class DA12_AnalogOutputSubsystem represents the analog output subsystem of a device.
- class DA12_OutputRange
- class DeviceSubsystem

    Class DeviceSubsystem is the abstract super class for all device subsystems.
- class DigitalIOSubsystem

    Class DigitalIOSubsystem represents the digital I/O subsystem of a device.
- class DIOStreamSubsystem

    Class DIOStreamSubsystem represents the digital I/O streaming subsystem of a device.
- class OutputVoltagePoint

    Class OutputVoltagePoint represents a single analog output data point, consisting of a D/A channel number and a voltage to output to that channel.
- class OutputVoltagePointArray
- class USB_AI16_Family
- class USB_AIO16_Family

    Class USB_AIO16_Family represents a USB-AI16-family device, which encompasses the following product IDs: USB-_AI16_16A, USB_AI16_16E, USB_AI12_16A, USB_AI12_16, USB_AI12_16E, USB_AI16_64MA, USB_AI16_64ME, U-SB_AI12_64MA, USB_AI12_64M, USB_AI12_64ME, USB_AI16_32A, USB_AI16_32E, USB_AI12_32A, USB_AI12_32, USB_AI12_32E, USB_AI16_64A, USB_AI16_64E, USB_AI12_64A, USB_AI12_64, USB_AI12_64E, USB_AI16_96A, U-SB_AI16_96E, USB_AI12_96A, USB_AI12_96, USB_AI12_96E, USB_AI16_128A, USB_AI16_128E, USB_AI12_128A, USB_AI12_128, USB_AI12_128E.
- class USB_AO16_Family

    Class USB_AO16_Family represents a USB-AO16-family device, which encompasses the following product IDs: USB_-AO16_16A, USB_AO16_16, USB_AO16_12A, USB_AO16_12, USB_AO16_8A, USB_AO16_8, USB_AO16_4A, USB_-AO16_4, USB_AO12_16A, USB_AO12_16, USB_AO12_12A, USB_AO12_12, USB_AO12_8A, USB_AO12_8, USB_A-O12_4A, USB_AO12_4.
- class USB_CTR_15_Family

Class USB_CTR_15_Family represents a USB-CTR-15-family device, which encompasses the following product IDs: USB_CTR_15.
- class USB_DA12_8A_Family

  Class USB_DA12_8A_Family represents a USB-DA12-8A-family device, which encompasses the following product IDs: USB_DA12_8A_REV_A, USB_DA12_8A.
- class USB_DA12_8E_Family

  Class USB_DA12_8E_Family represents a USB-DA12-8E-family device, which encompasses the following product IDs: USB_DA12_8E.
- class USB_DIO_16_Family

  Class USB_DIO_16_Family represents a USB-DIO-16-family device, which encompasses the following product IDs: USB_DI16A_REV_A1, USB_DO16A_REV_A1, USB_DI16A_REV_A2, USB_DIO_16H, USB_DI16A, USB_DO16A, USB_DIO_16A.
- class USB_DIO_32_Family

  Class USB_DIO_32_Family represents a USB-DIO-32-family device, which encompasses the following product IDs: USB_DIO_32.
- class USB_DIO_Family

  Class USB_DIO_Family represents a USB-DIO-family device, which performs basic digital I/O and encompasses the following product IDs: USB_DIO_48, USB_DIO_96, USB_IIRO_16, USB_II_16, USB_RO_16, USB_IIRO_8, USB_II_8, USB_IIRO_4, USB_IDIO_16, USB_II_16_OLD, USB_IDO_16, USB_IDIO_8, USB_II_8_OLD, USB_IDIO_4, USB_IIRO4_2SM, USB_IIRO4_COM, USB_DIO16RO8, PICO_DIO16RO8.
- class BoolArray
- class UCharArray
- class UShortArray
- class IntArray
- class DoubleArray
- class StringArray
- class USBDeviceArray
- class USBDeviceBase

  Class USBDeviceBase is the abstract super class of all USB device families.
- class USBDeviceManager

  Class USBDeviceManager manages all the USB devices on the bus.
- class OperationFailedException

  Class OperationFailedException is thrown whenever an operation attempted on a device fails.
- class IllegalArgumentException

  Class IllegalArgumentException is thrown whenever an invalid argument is passed to a method.

Functions

- ostream & operator<< (ostream &out, USBDeviceBase &device)
- ostream & operator<< (ostream &out, USBDeviceBase ∗device)
- std::ostream & operator<< (std::ostream &out, USBDeviceBase &device)
- std::ostream & operator<< (std::ostream &out, USBDeviceBase ∗device)

22.1.1 Function Documentation

ostream& AIOUSB::operator<< (ostream & *out*, USBDeviceBase & *device*)

ostream& AIOUSB::operator<< (ostream & *out*, USBDeviceBase ∗ *device*)

std::ostream& AIOUSB::operator<< (std::ostream & *out*, USBDeviceBase & *device*)

std::ostream& AIOUSB::operator<< (std::ostream & *out*, USBDeviceBase ∗ *device*)

Chapter 23

Data Structure Documentation

23.1 ad_gain_pairs Struct Reference

```
#include <AIOChannelRange.h>
```

Data Fields

- ADGainCode gain
- const char ∗ name

23.1.1 Field Documentation

ADGainCode gain

const char∗ name

The documentation for this struct was generated from the following file:

- lib/AIOChannelRange.h

23.2 ADCConfigBlock Struct Reference

```
#include <ADCConfigBlock.h>
```

Data Fields

- AIOUSBDevice ∗ device

  Pointer to the device Descriptor.
- unsigned long size
- unsigned char registers [AD_MAX_CONFIG_REGISTERS+1]
- unsigned timeout
- ADCMuxSettings mux_settings
- int clock_rate
- AIOUSB_BOOL discardFirstSample
- AIOUSB_BOOL debug
- AIOUSB_BOOL testing

  For making Unit tests that don't talk to hardware.

23.2.1 Field Documentation

AIOUSBDevice∗ device

Pointer to the device Descriptor.

unsigned long size

unsigned char registers[AD_MAX_CONFIG_REGISTERS+1]

unsigned timeout

ADCMuxSettings mux_settings

int clock_rate

AIOUSB_BOOL discardFirstSample

AIOUSB_BOOL debug

AIOUSB_BOOL testing

For making Unit tests that don't talk to hardware.

The documentation for this struct was generated from the following file:

- lib/ADCConfigBlock.h

23.3 ADRange Struct Reference

```
#include <AIOUSB_Core.h>
```

Data Fields

- double minVolts
- double range

23.3.1 Field Documentation

double minVolts

double range

The documentation for this struct was generated from the following file:

- lib/AIOUSB_Core.h

23.4 AI16_DataPoint Class Reference

Class AI16_DataPoint represents a single data point captured from a USB_AI16_Family device.

```
#include <AI16_DataPoint.hpp>
```

Public Member Functions

- AI16_DataPoint ()
- int getChannel () const

  Gets the channel number from which this data point was captured.
- int getRange () const

  Gets the range that was in effect when this data point was captured.
- std::string getRangeText () const

  Gets the textual representation of the range that was in effect when this data point was captured.
- bool isDifferentialMode () const

  Gets the differential/single-ended mode that was in effect when this data point was captured.
- int getCounts () const

  Gets the captured data in A/D counts.
- double getVolts () const

  Gets the captured data in volts.
- std::string toString () const

  Gets a single-line string summary of this data point.

Data Fields

- int counts
- int channel
- int range
- bool differentialMode

Friends

- class AI16_DataPointArray
- class AI16_DataSet
- class AnalogInputSubsystem
- class std::vector< AI16_DataPoint >

23.4.1 Detailed Description

Class AI16_DataPoint represents a single data point captured from a USB_AI16_Family device.

It encapsulates not only the captured sample, but the channel from which the sample was captured and the range and differential mode in effect when the sample was captured, providing a fairly complete representation of the captured data. This class also provides methods to retrieve the captured data in either A/D counts or volts.

23.4.2 Constructor & Destructor Documentation

AI16_DataPoint ( )

23.4.3 Member Function Documentation

int getChannel ( ) const `[inline]`

Gets the channel number from which this data point was captured.

Returns

The channel number from which this data point was captured.

int getRange ( ) const `[inline]`

Gets the range that was in effect when this data point was captured.

Returns

The range that was in effect when this data point was captured.

See Also

AnalogInputSubsystem::getRange(int channel) const

std::string getRangeText ( ) const

Gets the textual representation of the range that was in effect when this data point was captured.

Returns

The textual representation of the range that was in effect when this data point was captured.

bool isDifferentialMode ( ) const `[inline]`

Gets the differential/single-ended mode that was in effect when this data point was captured.

Returns

The differential/single-ended mode that was in effect when this data point was captured.

See Also

AnalogInputSubsystem::isDifferentialMode(int channel) const

int getCounts ( ) const `[inline]`

Gets the captured data in A/D counts.

Returns

The captured data in A/D counts.

double getVolts ( ) const

Gets the captured data in volts.

Returns

The captured data in volts.

std::string toString ( ) const

Gets a single-line string summary of this data point.

Mainly useful for diagnostic purposes.

Returns

A single-line string summary of this data point.

23.4.4 Friends And Related Function Documentation

friend class AI16_DataPointArray `[friend]`

friend class AI16_DataSet `[friend]`

friend class AnalogInputSubsystem `[friend]`

friend class std::vector< AI16_DataPoint > `[friend]`

23.4.5 Field Documentation

int counts

int channel

int range

bool differentialMode

The documentation for this class was generated from the following files:

- deprecated/classlib/AI16_DataPoint.hpp
- deprecated/classlib/AI16_DataPoint.cpp

23.5 AI16_DataPointArray Class Reference

`#include <AI16_DataPoint.hpp>`

Public Member Functions

- AI16_DataPointArray (int size=0)

23.5.1 Constructor & Destructor Documentation

AI16_DataPointArray (int *size* = 0) `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/AI16_DataPoint.hpp

23.6 AI16_DataSet Class Reference

Class AI16_DataSet represents a data set captured from a USB_AI16_Family device.

```
#include <AI16_DataSet.hpp>
```

Public Member Functions

- virtual ~AI16_DataSet ()

  Destructor for data set.
- AnalogInputSubsystem & getSubsystem ()

  Gets the subsystem from which this data set was obtained.
- const AI16_DataPointArray & getPoints ()

  Gets the data point array from this data set.
- long getTimeStamp ()

  Gets the approximate time stamp when this data set was captured.
- int getCalMode ()

  Gets the calibration mode that was in effect when this data set was captured.
- int getTriggerMode ()

  Gets the trigger mode that was in effect when this data set was captured.
- int getOverSample ()

  Gets the over-sample setting that was in effect when this data set was captured.
- bool isDiscardFirstSample ()

  Gets the sample discard mode that was in effect when this data set was captured.
- std::ostream & print (std::ostream &out)

  Prints this data set.

Protected Member Functions

- AI16_DataSet (AnalogInputSubsystem &subsystem, int numPoints, long timeStamp, int calMode, int triggerMode, int overSample, bool discardFirstSample)

Protected Attributes

- AnalogInputSubsystem ∗ subsystem
- AI16_DataPointArray points
- long timeStamp
- int calMode
- int triggerMode
- int overSample
- bool discardFirstSample

Friends

- class AnalogInputSubsystem

23.6.1 Detailed Description

Class AI16_DataSet represents a data set captured from a USB_AI16_Family device.

It comprises a fairly complete snapshot of both the data and the sampling parameters, including a time stamp.

See Also

AnalogInputSubsystem::read(int startChannel, int numChannels)

23.6.2 Constructor & Destructor Documentation

AI16_DataSet (AnalogInputSubsystem & *subsystem,* int *numPoints,* long *timeStamp,* int *calMode,* int *triggerMode,* int *overSample,* bool *discardFirstSample*) `[protected]`

~AI16_DataSet ( ) `[virtual]`

Destructor for data set.

Data sets returned by methods such as *AnalogInputSubsystem::read()* must be explicitly destroyed.

23.6.3 Member Function Documentation

AnalogInputSubsystem& getSubsystem ( ) `[inline]`

Gets the subsystem from which this data set was obtained.

Returns

The subsystem from which this data set was obtained.

const AI16_DataPointArray& getPoints ( ) `[inline]`

Gets the data point array from this data set.

Returns

The data point array from this data set.

long getTimeStamp ( ) `[inline]`

Gets the approximate time stamp when this data set was captured.

The system time (obtained from *time()*) is recorded immediately prior to the sampling of the data, so it approximately represents the time when the data capture started. This property is not intended to be precise, but merely to serve as a convenient reference.

Returns

The approximate time stamp when this data set was captured.

int getCalMode ( ) `[inline]`

Gets the calibration mode that was in effect when this data set was captured.

Returns

The calibration mode that was in effect when this data set was captured.

See Also

AnalogInputSubsystem::getCalMode() const

int getTriggerMode ( ) `[inline]`

Gets the trigger mode that was in effect when this data set was captured.

Returns

The trigger mode that was in effect when this data set was captured.

See Also

AnalogInputSubsystem::getTriggerMode() const

int getOverSample ( ) `[inline]`

Gets the over-sample setting that was in effect when this data set was captured.

Returns

The over-sample setting that was in effect when this data set was captured.

See Also

AnalogInputSubsystem::getOverSample() const

bool isDiscardFirstSample ( ) `[inline]`

Gets the sample discard mode that was in effect when this data set was captured.

Returns

The sample discard mode that was in effect when this data set was captured.

See Also

AnalogInputSubsystem::isDiscardFirstSample() const

std::ostream & print (std::ostream & *out*)

Prints this data set.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where the data set will be printed.

Returns

The print stream.

23.6.4 Friends And Related Function Documentation

friend class AnalogInputSubsystem `[friend]`

23.6.5 Field Documentation

AnalogInputSubsystem∗ subsystem `[protected]`

AI16_DataPointArray points `[protected]`

long timeStamp `[protected]`

int calMode `[protected]`

int triggerMode `[protected]`

int overSample `[protected]`

bool discardFirstSample `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/AI16_DataSet.hpp
- deprecated/classlib/AI16_DataSet.cpp

23.7 AI16_InputRange Class Reference

```
#include <AI16_InputRange.hpp>
```

Public Member Functions

- virtual AnalogIORange & setRange (int range)

  Sets the range ID.

Protected Member Functions

- AI16_InputRange ()
- AI16_InputRange (int minCounts, int maxCounts)
- virtual ∼AI16_InputRange ()

Friends

- class AnalogInputSubsystem
- class AI16_DataPoint

Additional Inherited Members

23.7.1 Constructor & Destructor Documentation

AI16_InputRange ( ) `[protected]`

AI16_InputRange (int *minCounts,* int *maxCounts*) `[protected]`

~AI16_InputRange ( ) `[protected]`, `[virtual]`

23.7.2 Member Function Documentation

AnalogIORange & setRange (int *range*) `[virtual]`

Sets the range ID.

Parameters

range	the new range ID (defined by class that owns this instance).

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented from AnalogIORange.

23.7.3 Friends And Related Function Documentation

friend class AnalogInputSubsystem `[friend]`

friend class AI16_DataPoint `[friend]`

The documentation for this class was generated from the following files:

- deprecated/classlib/AI16_InputRange.hpp
- deprecated/classlib/AI16_InputRange.cpp

23.8 aio_channel_range Struct Reference

```
#include <AIOChannelRange.h>
```

Data Fields

- int start
- int end
- ADGainCode gain

23.8.1 Field Documentation

int start

int end

ADGainCode gain

The documentation for this struct was generated from the following file:

- lib/AIOChannelRange.h

23.9 aio_counts_converter Struct Reference

```
#include <AIOCountsConverter.h>
```

Data Fields

- unsigned num_oversamples
- unsigned num_channels
- unsigned num_scans
- unsigned unit_size
- unsigned scan_count
- unsigned channel_count
- unsigned os_count
- unsigned converted_count
- unsigned sum
- void ∗ buf
- int(∗ continue_conversion)(struct aio_counts_converter ∗cc, unsigned rounded_num_counts)
- AIOGainRange ∗ gain_ranges
- AIORET_TYPE(∗ Convert)(struct aio_counts_converter ∗cc, void ∗tobuf, void ∗frombuf, unsigned num_bytes)
- AIORET_TYPE(∗ ConvertFifo)(struct aio_counts_converter ∗cc, void ∗tobuf, void ∗frombuf, unsigned num_bytes)
- AIOUSB_BOOL discardFirstSample

23.9.1 Field Documentation

unsigned num_oversamples

unsigned num_channels

unsigned num_scans

unsigned unit_size

unsigned scan_count

unsigned channel_count

unsigned os_count

unsigned converted_count

unsigned sum

void∗ buf

int(∗ continue_conversion)(struct aio_counts_converter ∗cc, unsigned rounded_num_counts)

AIOGainRange∗ gain_ranges

AIORET_TYPE(∗ Convert)(struct aio_counts_converter ∗cc, void ∗tobuf, void ∗frombuf, unsigned num_bytes)

AIORET_TYPE(∗ ConvertFifo)(struct aio_counts_converter ∗cc, void ∗tobuf, void ∗frombuf, unsigned num_bytes)

AIOUSB_BOOL discardFirstSample

The documentation for this struct was generated from the following file:

- lib/AIOCountsConverter.h

23.10 aio_either_val Struct Reference

```
#include <AIOEither.h>
```

Data Fields

- AIO_NUMBER number
- void ∗ object

23.10.1 Field Documentation

AIO_NUMBER number

void∗ object

The documentation for this struct was generated from the following file:

- lib/AIOEither.h

23.11 aio_ret_value Struct Reference

```
#include <AIOEither.h>
```

Data Fields

- int left
- char ∗ errmsg
- AIO_EITHER_VALUE_ITEM right
- AIO_EITHER_TYPE type
- int size

23.11.1 Field Documentation

int left

char∗ errmsg

AIO_EITHER_VALUE_ITEM right

AIO_EITHER_TYPE type

int size

The documentation for this struct was generated from the following file:

- lib/AIOEither.h

23.12 AIOArgument Struct Reference

```
#include <AIOConfiguration.h>
```

Data Fields

- AIOUSB_BOOL threaded
- AIOUSB_BOOL debug
- int ∗ size
- int actual_size
- AIOConfiguration config

23.12.1 Field Documentation

AIOUSB_BOOL threaded

AIOUSB_BOOL debug

int∗ size

int actual_size

AIOConfiguration config

The documentation for this struct was generated from the following file:

- lib/AIOConfiguration.h

23.13 AIOArguments Struct Reference

```
#include <AIOConfiguration.h>
```

Data Fields

- AIOArgument ∗ device_args
- int number_arguments

23.13.1 Field Documentation

AIOArgument∗ **device_args**

int number_arguments

The documentation for this struct was generated from the following file:

- lib/AIOConfiguration.h

23.14 AIOBuf Struct Reference

```
#include <AIOBuf.h>
```

Data Fields

- size_t size
- size_t endpos
- AIOBufType type
- AIOUSB_BOOL defined

23.14.1 Field Documentation

size_t size

size_t endpos

AIOBufType type

AIOUSB_BOOL defined

The documentation for this struct was generated from the following file:

- lib/AIOBuf.h

23.15 aiobuf_iterator Struct Reference

```
#include <AIOBuf.h>
```

Data Fields

- AIOBuf ∗ buf
- void ∗ loc
- void(∗ next)(struct aiobuf_iterator ∗)

23.15.1 Field Documentation

AIOBuf∗ **buf**

void∗ **loc**

void(∗ **next)(struct aiobuf_iterator** ∗**)**

The documentation for this struct was generated from the following file:

- lib/AIOBuf.h

23.16 AIOChannelMask Struct Reference

```
#include <AIOChannelMask.h>
```

Data Fields

- int ∗ signal_indices
- int signal_index
- unsigned active_signals
- aio_channel_obj ∗ signals
- unsigned number_signals
- unsigned pos
- int size
- char ∗ strrep
- char ∗ strrepsmall

23.16.1 Field Documentation

int∗ signal_indices

int signal_index

unsigned active_signals

aio_channel_obj∗ signals

unsigned number_signals

unsigned pos

int size

char∗ strrep

char∗ strrepsmall

The documentation for this struct was generated from the following file:

- lib/AIOChannelMask.h

23.17 AIOChannelRangeTmp Struct Reference

```
#include <AIOCommandLine.h>
```

Data Fields

- int start_channel
- int end_channel
- int gaincode

23.17.1 Field Documentation

int start_channel

int end_channel

int gaincode

The documentation for this struct was generated from the following file:

- lib/AIOCommandLine.h

23.18 AIOCmd Struct Reference

```
#include <AIOCmd.h>
```

Data Fields

- int stop_scan
- int stop_scan_arg
- int channel
- unsigned long num_scans
- unsigned num_channels
- unsigned num_samples

23.18.1 Field Documentation

int stop_scan

int stop_scan_arg

int channel

unsigned long num_scans

unsigned num_channels

unsigned num_samples

The documentation for this struct was generated from the following file:

- lib/AIOCmd.h

23.19 AIOCommandLineOptions Struct Reference

```
#include <AIOCommandLine.h>
```

Data Fields

- int pass_through
- int64_t num_scans
- int64_t default_num_scans
- int num_channels
- int default_num_channels
- int num_oversamples
- int default_num_oversamples
- int gain_code
- int clock_rate
- int default_clock_rate
- char ∗ outfile
- int reset
- int debug_level
- int number_ranges
- int verbose
- int start_channel
- int default_start_channel
- int end_channel
- int default_end_channel
- int index
- int block_size
- int with_timing
- int slow_acquire
- int buffer_size
- int rate_limit
- int physical
- int counts
- int calibration

- int repeat_number
- char ∗ aiobuf_json
- char ∗ default_aiobuf_json
- char ∗ adcconfig_json
- AIOChannelRangeTmp ∗∗ ranges

23.19.1 Field Documentation

int pass_through

int64_t num_scans

int64_t default_num_scans

int num_channels

int default_num_channels

int num_oversamples

int default_num_oversamples

int gain_code

int clock_rate

int default_clock_rate

char∗ outfile

int reset

int debug_level

int number_ranges

int verbose

int start_channel

int default_start_channel

int end_channel

int default_end_channel

int index

int block_size

int with_timing

int slow_acquire

int buffer_size

int rate_limit

int physical

int counts

int calibration

int repeat_number

char∗ aiobuf_json

char∗ default_aiobuf_json

char∗ adcconfig_json

AIOChannelRangeTmp∗∗ ranges

The documentation for this struct was generated from the following file:

- lib/AIOCommandLine.h

23.20 AIOContinuousBuf Struct Reference

AIOContinuousBuf provides a buffer that is used with the AIOUSB highspeed data acquisition API.

```
#include <AIOContinuousBuffer.h>
```

Data Fields

- void ∗(∗ callback)(void ∗object)
- pthread_t worker
- pthread_mutex_t lock
- pthread_attr_t tattr
- AIOUSB_WorkFn work
- int DeviceIndex
- AIOFifoTYPE ∗ fifo
- AIOBufferType ∗ buffer
- unsigned char ∗ countsbuf
- unsigned unit_size
- unsigned hz
- unsigned base_size
- unsigned size
- unsigned num_oversamples
- unsigned num_channels
- int64_t num_scans
- int64_t scans_read
- AIOUSB_BOOL start_scanning
- unsigned block_size
- int64_t bytes_processed
- unsigned counter_control
- unsigned timeout
- AIORET_TYPE exitcode
- AIOUSB_BOOL testing
- AIOUSB_BOOL debug
- AIOChannelMask ∗ mask

  Used for keeping track of channels.
- volatile THREAD_STATUS status
- AIO_CONT_BUF_TYPE type
- AIORET_TYPE(∗ PushN)(struct AIOContinuousBuf ∗buf, void ∗frombuf, unsigned int N)
- AIORET_TYPE(∗ PopN)(struct AIOContinuousBuf ∗buf, void ∗frombuf, unsigned int N)

23.20.1 Detailed Description

AIOContinuousBuf provides a buffer that is used with the AIOUSB highspeed data acquisition API.

It is designed to provide an ease of use with getting these acquisitions running with as little fuss as possible. They key flow for using this buffer is the following:

- Create a new AIOContinuousBuf of a certain size that is large enough to handle the running clock rate ∗ number-_of_oversamples ∗

- Assign a device index to the AIOContinuousBuf

- Start am acquisition by calling AIOContinuousBufInitiateCallbackAcquisition;

- Process the input data using either a simple while loop burst_test.c

  or using the callback function as in

23.20.2 Field Documentation

void∗(∗ callback)(void ∗object)

pthread_t worker

pthread_mutex_t lock

pthread_attr_t tattr

AIOUSB_WorkFn work

int DeviceIndex

AIOFifoTYPE∗ fifo

AIOBufferType∗ buffer

unsigned char∗ countsbuf

unsigned unit_size

unsigned hz

unsigned base_size

unsigned size

unsigned num_oversamples

unsigned num_channels

int64_t num_scans

int64_t scans_read

AIOUSB_BOOL start_scanning

unsigned block_size

int64_t bytes_processed

unsigned counter_control

unsigned timeout

AIORET_TYPE exitcode

AIOUSB_BOOL testing

AIOUSB_BOOL debug

AIOChannelMask∗ mask

Used for keeping track of channels.

volatile THREAD_STATUS status

AIO_CONT_BUF_TYPE type

AIORET_TYPE(∗ PushN)(struct AIOContinuousBuf ∗buf, void ∗frombuf, unsigned int N)

AIORET_TYPE(∗ PopN)(struct AIOContinuousBuf ∗buf, void ∗frombuf, unsigned int N)

The documentation for this struct was generated from the following file:

- lib/AIOContinuousBuffer.h

23.21 AIODeviceInfo Struct Reference

```
#include <AIODeviceInfo.h>
```

Data Fields

- unsigned long PID
- unsigned long NameSize
- char ∗ Name
- unsigned long DIOBytes
- unsigned long Counters

23.21.1 Field Documentation

unsigned long PID

unsigned long NameSize

char∗ Name

unsigned long DIOBytes

unsigned long Counters

The documentation for this struct was generated from the following file:

- lib/AIODeviceInfo.h

23.22 AIODeviceQuery Struct Reference

```
#include <AIODeviceQuery.h>
```

Data Fields

- unsigned long productID

  Product ID for the device.
- unsigned long nameSize

  Name length for the device.
- char ∗ name

  Name of the device.
- unsigned long numDIOBytes

  Number of digital bytes.
- unsigned long numCounters

  Number of counters.
- unsigned long index

  Index this is associated with.

23.22.1 Field Documentation

unsigned long productID

Product ID for the device.

unsigned long nameSize

Name length for the device.

char∗ name

Name of the device.

unsigned long numDIOBytes

Number of digital bytes.

unsigned long numCounters

Number of counters.

unsigned long index

Index this is associated with.

The documentation for this struct was generated from the following file:

- lib/AIODeviceQuery.h

23.23 aioerror Struct Reference

```
#include <AIOUSB_Core.h>
```

Data Fields

- AIORET_TYPE retval
- char ∗ error_message

23.23.1 Field Documentation

AIORET_TYPE retval

char∗ error_message

The documentation for this struct was generated from the following file:

- lib/AIOUSB_Core.h

23.24 AIOFifo Struct Reference

AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition.

```
#include <AIOFifo.h>
```

Data Fields

- AIO_FIFO_INTERFACE

    The Interface for the FIFO that describes the read / write and size functions that AIOFifo should provide.
- LOCKING_MECHANISM

    The Interface for the Locking mechanism that defines GRAB_RESOURCE and RELEASE_RESOURCE.

23.24.1 Detailed Description

AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition.

The definition of the structure is comprised of the base interface (created wtih a #define) in AIO_FIFO_INTERFACE which handles the basic read and writing to the fifo. In addition , it also includes the Interface called LOCKING_MECHANISM, that makes sure that a write access to the FIFO is atomic.

23.24.2 Field Documentation

AIO_FIFO_INTERFACE

The Interface for the FIFO that describes the read / write and size functions that AIOFifo should provide.

LOCKING_MECHANISM

The Interface for the Locking mechanism that defines GRAB_RESOURCE and RELEASE_RESOURCE.

The documentation for this struct was generated from the following file:

- lib/AIOFifo.h

23.25 AIOGainRange Struct Reference

```
#include <AIOCountsConverter.h>
```

Data Fields

- double min
- double max

23.25.1 Field Documentation

double min

double max

The documentation for this struct was generated from the following file:

- lib/AIOCountsConverter.h

23.26 AIOProductGroup Struct Reference

A smart product group that marks a range of ACCES I/O Products.

```
#include <AIOProductTypes.h>
```

23.26.1 Detailed Description

A smart product group that marks a range of ACCES I/O Products.

The documentation for this struct was generated from the following file:

- lib/AIOProductTypes.h

23.27 AIOProductRange Struct Reference

A simplified range of Products based off of device ids.

```
#include <AIOProductTypes.h>
```

23.27.1 Detailed Description

A simplified range of Products based off of device ids.

The documentation for this struct was generated from the following file:

- lib/AIOProductTypes.h

23.28 aiousb_libusb_args Struct Reference

```
#include <USBDevice.h>
```

Data Fields

- struct libusb_device ∗ dev
- struct libusb_device_handle ∗ handle
- struct libusb_device_descriptor ∗ deviceDesc

23.28.1 Field Documentation

struct libusb_device∗ dev

struct libusb_device_handle∗ handle

struct libusb_device_descriptor∗ deviceDesc

The documentation for this struct was generated from the following file:

- lib/USBDevice.h

23.29 AIOUSBDevice Struct Reference

```
#include <AIOUSBDevice.h>
```

Data Fields

- USBDevice ∗ usb_device
- AIOUSB_BOOL bOpen
- int deviceIndex
- AIOUSB_BOOL isInit
- unsigned long PID
- unsigned long DIOConfigBits
- AIOUSB_BOOL discardFirstSample

  AIOUSB_TRUE == discard first A/D sample in all A/D read methods.
- unsigned commTimeout

  timeout for device communication (ms.)
- double miscClockHz

  miscellaneous clock frequency setting
- unsigned ProductID
- unsigned DIOBytes
- unsigned Counters
- unsigned Tristates
- AIOUSB_BOOL bGateSelectable
- long RootClock
- AIOUSB_BOOL bGetName
- unsigned long ConfigBytes
- unsigned ImmDACs
- AIOUSB_BOOL bDACStream
- AIOUSB_BOOL bDACDIOStream
- AIOUSB_BOOL bDACSlowWaveStream
- AIOUSB_BOOL bDACDIOClock
- unsigned DACsUsed
- AIOUSB_BOOL bADCStream
- unsigned ADCChannels
- unsigned ADCMUXChannels
- unsigned char RangeShift
- unsigned ADCChannelsPerGroup

  number of A/D channels in each config.
- AIOUSB_BOOL bDIOStream
- unsigned long StreamingBlockSize
- AIOUSB_BOOL bDIODebounce
- AIOUSB_BOOL bDIOSPI
- AIOUSB_BOOL bSetCustomClocks
- unsigned WDGBytes
- AIOUSB_BOOL bClearFIFO
- unsigned ImmADCs
- AIOUSB_BOOL bDACBoardRange
- AIOUSB_BOOL bDACChannelCal
- unsigned FlashSectors
- AIOUSB_BOOL bDACOpen
- AIOUSB_BOOL bDACClosing
- AIOUSB_BOOL bDACAborting
- AIOUSB_BOOL bDACStarted

- unsigned char ∗∗ DACData
- unsigned char ∗ PendingDACData
- pthread_mutex_t hDACDataMutex
- sem_t hDACDataSem
- AIOUSB_BOOL bDIOOpen
- AIOUSB_BOOL bDIORead
- AIOUSB_BOOL bDeviceWasHere
- unsigned char ∗ LastDIOData
- char ∗ cachedName
- unsigned long cachedSerialNumber
- ADCConfigBlock cachedConfigBlock

    .size == 0 == uninitialized
- AIOUSB_BOOL workerBusy

    state of worker thread; these fields are deliberately unspecific so that the library can employ worker threads in a variety of situations
- unsigned long workerStatus

    thread-defined status information (e.g.
- unsigned long workerResult

    standard AIOUSB_∗ result code from worker thread (if workerBusy == AIOUSB_FALSE)
- ADCConfigBlock ∗ FastITConfig

    New entries for the FastIT behavior.
- ADCConfigBlock ∗ FastITBakConfig
- unsigned long FastITConfig_size
- unsigned char ∗ ADBuf
- int ADBuf_size
- AIOUSB_BOOL testing
- AIOUSB_BOOL valid

23.29.1 Field Documentation

USBDevice∗ usb_device

AIOUSB_BOOL bOpen

int deviceIndex

AIOUSB_BOOL isInit

unsigned long PID

unsigned long DIOConfigBits

AIOUSB_BOOL discardFirstSample

AIOUSB_TRUE == discard first A/D sample in all A/D read methods.

unsigned commTimeout

timeout for device communication (ms.)

double miscClockHz

miscellaneous clock frequency setting

unsigned ProductID

unsigned DIOBytes

unsigned Counters

unsigned Tristates

AIOUSB_BOOL bGateSelectable

long RootClock

AIOUSB_BOOL bGetName

unsigned long ConfigBytes

unsigned ImmDACs

AIOUSB_BOOL bDACStream

AIOUSB_BOOL bDACDIOStream

AIOUSB_BOOL bDACSlowWaveStream

AIOUSB_BOOL bDACDIOClock

unsigned DACsUsed

AIOUSB_BOOL bADCStream

unsigned ADCChannels

unsigned ADCMUXChannels

unsigned char RangeShift

unsigned ADCChannelsPerGroup

number of A/D channels in each config.
group (1, 4 or 8 depending on model)

AIOUSB_BOOL bDIOStream

unsigned long StreamingBlockSize

AIOUSB_BOOL bDIODebounce

AIOUSB_BOOL bDIOSPI

AIOUSB_BOOL bSetCustomClocks

unsigned WDGBytes

AIOUSB_BOOL bClearFIFO

unsigned ImmADCs

AIOUSB_BOOL bDACBoardRange

AIOUSB_BOOL bDACChannelCal

unsigned FlashSectors

AIOUSB_BOOL bDACOpen

AIOUSB_BOOL bDACClosing

AIOUSB_BOOL bDACAborting

AIOUSB_BOOL bDACStarted

unsigned char∗∗ DACData

unsigned char∗ PendingDACData

pthread_mutex_t hDACDataMutex

sem_t hDACDataSem

AIOUSB_BOOL bDIOOpen

AIOUSB_BOOL bDIORead

AIOUSB_BOOL bDeviceWasHere

unsigned char* LastDIOData

char* cachedName

unsigned long cachedSerialNumber

ADCConfigBlock cachedConfigBlock

.size == 0 == uninitialized

AIOUSB_BOOL workerBusy

state of worker thread; these fields are deliberately unspecific so that the library can employ worker threads in a variety of situations

AIOUSB_TRUE == worker thread is busy

unsigned long workerStatus

thread-defined status information (e.g.

bytes remaining to receive or transmit)

unsigned long workerResult

standard AIOUSB_* result code from worker thread (if workerBusy == AIOUSB_FALSE)

ADCConfigBlock* FastITConfig

New entries for the FastIT behavior.

ADCConfigBlock* FastITBakConfig

unsigned long FastITConfig_size

unsigned char* ADBuf

int ADBuf_size

AIOUSB_BOOL testing

AIOUSB_BOOL valid

The documentation for this struct was generated from the following file:

- lib/AIOUSBDevice.h

23.30 aiousboption Struct Reference

`#include <AIOUSB_Core.h>`

The documentation for this struct was generated from the following file:

- lib/AIOUSB_Core.h

23.31 AIOWDGConfig Struct Reference

`#include <AIOUSB_WDG.h>`

Data Fields

- int bufsize
- unsigned long L
- unsigned char ∗ wdgbuf
- unsigned long timeout

23.31.1 Field Documentation

int bufsize

unsigned long L

unsigned char∗ wdgbuf

unsigned long timeout

The documentation for this struct was generated from the following file:

- lib/AIOUSB_WDG.h

23.32 AnalogInputSubsystem Class Reference

Class AnalogInputSubsystem represents the analog input subsystem of a device.

```
#include <AnalogInputSubsystem.hpp>
```

Public Member Functions

- AnalogInputSubsystem & setScanRange (int startChannel, int numChannels)
- AnalogInputSubsystem (USBDeviceBase &parent)
- virtual ∼AnalogInputSubsystem ()
- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this subsystem.
- int getNumChannels () const

  Gets the number of primary analog input channels.
- int getNumMUXChannels () const

  Gets the number of analog input channels available through an optional multiplexer.
- int getChannelsPerGroup ()

  Gets the number of analog input channels in each configuration group (1, 4 or 8 depending on the device model).
- bool isAutoCalPresent (bool force)

  Tells if automatic calibration is possible with this device.
- bool isAutoConfig () const

  Tells whether the modified configuration will be automatically sent to the device.
- AnalogInputSubsystem & setAutoConfig (bool autoConfig)

  Enables or disables automatically sending the modified configuration to the device.
- AnalogInputSubsystem & readConfig ()

  Reads the A/D configuration from the device.
- AnalogInputSubsystem & writeConfig ()

  Writes the A/D configuration to the device.
- bool isDiscardFirstSample () const

  Tells if the read(), readCounts() and readVolts() functions will discard the first A/D sample taken.
- AnalogInputSubsystem & setDiscardFirstSample (bool discard)

  Specifies whether the read(), readCounts() and readVolts() functions will discard the first A/D sample taken.
- int getCalMode () const

  Gets the current calibration mode.
- AnalogInputSubsystem & setCalMode (int calMode)

  Sets the A/D calibration mode.
- int getTriggerMode () const

  Gets the current trigger mode.
- AnalogInputSubsystem & setTriggerMode (int triggerMode)

  Sets the trigger mode.
- int getRange (int channel) const

  Gets the current range for channel.

- IntArray getRange (int startChannel, int numChannels) const

  Gets the current range for multiple A/D channels.
- AnalogInputSubsystem & setRange (int channel, int range)

  Sets the range for a single A/D channel.
- AnalogInputSubsystem & setRange (int startChannel, const IntArray &range)

  Sets the range for multiple A/D channels.
- bool isDifferentialMode (int channel) const

  Tells if channel is configured for single-ended or differential mode.
- BoolArray isDifferentialMode (int startChannel, int numChannels) const

  Tells if multiple A/D channels are configured for single-ended or differential mode.
- AnalogInputSubsystem & setDifferentialMode (int channel, bool differentialMode)

  Sets a single A/D channel to differential or single-ended mode.
- AnalogInputSubsystem & setDifferentialMode (int startChannel, const BoolArray &differentialMode)

  Sets multiple A/D channels to differential or single-ended mode.
- AnalogInputSubsystem & setRangeAndDiffMode (int channel, int range, bool differentialMode)

  Sets the range and differential mode for a single A/D channel.
- AnalogInputSubsystem & setRangeAndDiffMode (int startChannel, const IntArray &range, const BoolArray &differentialMode)

  Sets the range and differential mode for multiple A/D channels.
- AnalogInputSubsystem & setRangeAndDiffMode (int range, bool differentialMode)

  Sets all the A/D channels to the same range and differential mode.
- int getOverSample () const

  Gets the current number of over-samples.
- AnalogInputSubsystem & setOverSample (int overSample)

  Sets the number of over-samples for all A/D channels.
- AnalogInputSubsystem & setCalibrationTable (const std::string &fileName)

  Loads a calibration table from a file into the A/D.
- AnalogInputSubsystem & setCalibrationTable (const UShortArray &calTable)

  Sets the calibration table in the A/D to the contents of calTable.
- int getStreamingBlockSize ()

  Gets the current streaming block size.
- AnalogInputSubsystem & setStreamingBlockSize (int blockSize)

  Sets the streaming block size.
- double getClock ()

  Gets the current clock frequency for timer-driven bulk reads (see setClock(double clockHz)).
- AnalogInputSubsystem & setClock (double clockHz)

  Sets the clock frequency for timer-driven bulk reads (see getClock() and readBulkStart(int startChannel, int numChannels, int numSamples)).
- UShortArray calibrate (bool autoCal, bool returnCalTable, const std::string &saveFileName)

  Calibrates the A/D, generating either a default table or using the internal voltage references to generate a calibration table.
- UShortArray calibrate (const DoubleArray &points, bool returnCalTable, const std::string &saveFileName)

  Permits the A/D to be calibrated using an external voltage source.
- AI16_DataSet ∗ read (int startChannel, int numChannels)

  Reads from multiple A/D channels and returns a data set containing both the data captured and the parameters in effect at the time the data was captured.
- unsigned short readCounts (int channel)

  Reads the A/D count value from a single channel.
- UShortArray readCounts (int startChannel, int numChannels)

  Reads the A/D count values from multiple channels.
- double readVolts (int channel)

  Reads the voltage from a single channel.
- DoubleArray readVolts (int startChannel, int numChannels)

  Reads the voltage from multiple channels.
- AnalogInputSubsystem & readBulkStart (int startChannel, int numChannels, int numSamples)

  Starts a large A/D acquisition process in a background thread and returns immediately.
- int readBulkSamplesAvailable ()

  Gets the number of samples available to be retrieved during a bulk acquisition process initiated by readBulkStart(int startChannel, int numChannels, int numSamples).
- UShortArray readBulkNext (int numSamples)

  Retrieves the next set of samples acquired during a bulk acquisition process initiated by readBulkStart(int startChannel, int numChannels, int numSamples).
- AnalogInputSubsystem & clearFIFO (FIFO_Method method)

  Clears the streaming FIFO, using one of several different methods.
- double countsToVolts (int channel, unsigned short counts) const

Converts a single A/D count value to volts, based on the current gain setting for the specified channel.

- DoubleArray countsToVolts (int startChannel, const UShortArray &counts) const

  Converts an array of A/D count values to an array of voltage values, based on the current gain setting for each of the specified channels.
- unsigned short voltsToCounts (int channel, double volts) const

  Converts a single voltage value to A/D counts, based on the current gain setting for the specified channel.
- UShortArray voltsToCounts (int startChannel, const DoubleArray &volts) const

  Converts an array of voltage values to an array of A/D count values, based on the current gain setting for each of the specified channels.

Static Public Member Functions

- static std::string getRangeText (int range)

  Gets the textual string for the specified range.

Static Public Attributes

- static const int CAL_MODE_NORMAL = 0

  Selects normal measurement mode (see setCalMode(int calMode)).
- static const int CAL_MODE_GROUND = 1

  Selects ground calibration mode (see setCalMode(int calMode)).
- static const int CAL_MODE_REFERENCE = 3

  Selects reference (full scale) calibration mode (see setCalMode(int calMode)).
- static const int TRIG_MODE_CTR0_EXT = 0x10

  If set, counter 0 is externally triggered (see setTriggerMode(int triggerMode)).
- static const int TRIG_MODE_FALLING_EDGE = 0x08

  If set, the A/D is triggered by the falling edge of its trigger source, otherwise it's triggered by the rising edge (see set-TriggerMode(int triggerMode)).
- static const int TRIG_MODE_SCAN = 0x04

  If set, each trigger will cause the A/D to scan all the channels, otherwise the A/D will read a single channel with each trigger (see setTriggerMode(int triggerMode)).
- static const int TRIG_MODE_EXTERNAL = 0x02

  If set, the A/D is triggered by an external pin on the board (see setTriggerMode(int triggerMode)).
- static const int TRIG_MODE_TIMER = 0x01

  If set, the A/D is triggered by counter 2 (see setTriggerMode(int triggerMode)).
- static const int RANGE_0_10V = 0

  Unipolar, 0-10 volt range (see setRange(int channel, int range)).
- static const int RANGE_10V = 1

  Bipolar, -10 to +10 volt range (see setRange(int channel, int range)).
- static const int RANGE_0_5V = 2

  Unipolar, 0-5 volt range (see setRange(int channel, int range)).
- static const int RANGE_5V = 3

  Bipolar, -5 to +5 volt range (see setRange(int channel, int range)).
- static const int RANGE_0_2V = 4

  Unipolar, 0-2 volt range (see setRange(int channel, int range)).
- static const int RANGE_2V = 5

  Bipolar, -2 to +2 volt range (see setRange(int channel, int range)).
- static const int RANGE_0_1V = 6

  Unipolar, 0-1 volt range (see setRange(int channel, int range)).
- static const int RANGE_1V = 7

  Bipolar, -1 to +1 volt range (see setRange(int channel, int range)).
- static const int MIN_COUNTS = 0

  Minimum number of counts A/D can read.
- static const int MAX_COUNTS = 0xffff

  Maximum number of counts A/D can read.
- static const int CAL_TABLE_WORDS = 64 $*$ 1024

  Number of 16-bit words in an A/D calibration table (65,536 16-bit words).

Protected Attributes

- int numChannels
- int numMUXChannels
- int channelsPerGroup
- int configBlockSize
- int autoCalFeature
- AI16_InputRange ∗ inputRange
- bool ∗ differentialMode
- int calMode
- int triggerMode
- int startChannel
- int endChannel
- int overSample
- unsigned short ∗ readBulkBuffer
- int readBulkSamplesRequested
- int readBulkSamplesRetrieved
- bool autoConfig

Static Protected Attributes

- static const char RANGE_TEXT [][10]
- static const int NUM_CONFIG_REGISTERS = 20
- static const int NUM_MUX_CONFIG_REGISTERS = 21
- static const int NUM_GAIN_CODE_REGISTERS = 16
- static const int REG_GAIN_CODE = 0
- static const int REG_CAL_MODE = 16
- static const int REG_TRIG_MODE = 17
- static const int REG_START_END = 18
- static const int REG_OVERSAMPLE = 19
- static const int REG_MUX_START_END = 20
- static const int DIFFERENTIAL_MODE = 8
- static const int MAX_OVERSAMPLE = 0xff
- static const int TRIG_MODE_VALID_MASK
- static const int AUTO_CAL_UNKNOWN = 0
- static const int AUTO_CAL_NOT_PRESENT = 1
- static const int AUTO_CAL_PRESENT = 2
- static const int MAX_CHANNELS = 128

Friends

- class USB_AI16_Family

Additional Inherited Members

23.32.1 Detailed Description

Class AnalogInputSubsystem represents the analog input subsystem of a device.

One accesses this analog input subsystem through its parent object, typically through a method such as *adc() (see USB_AI16_Family::adc())*.

23.32.2 Constructor & Destructor Documentation

AnalogInputSubsystem (USBDeviceBase & *parent*)

∼AnalogInputSubsystem ( ) [virtual]

23.32.3 Member Function Documentation

AnalogInputSubsystem & setScanRange (int *startChannel,* int *numChannels*)

ostream & print (std::ostream & *out*) [virtual]

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumChannels ( ) const `[inline]`

Gets the number of primary analog input channels.

Returns

Number of channels, numbered 0 to n-1.

int getNumMUXChannels ( ) const `[inline]`

Gets the number of analog input channels available through an optional multiplexer.

Returns

Number of channels, numbered 0 to n-1.

int getChannelsPerGroup ( ) `[inline]`

Gets the number of analog input channels in each configuration group (1, 4 or 8 depending on the device model).

Returns

The number of channels per group.

bool isAutoCalPresent (bool *force*)

Tells if automatic calibration is possible with this device.

Parameters

force	*True* forces this class to interrogate the device anew; *false* returns the previous result if available, or interrogates the device if a previous result is not available.

Returns

True indicates that automatic calibration is available.

See Also

calibrate(bool autoCal, bool returnCalTable, const std::string &saveFileName)

Exceptions

OperationFailedException	

std::string getRangeText (int *range*) `[static]`

Gets the textual string for the specified range.

Parameters

range	the range for which to obtain the textual string.

Returns

The textual string for the specified range.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

bool isAutoConfig ( ) const [inline]

Tells whether the modified configuration will be automatically sent to the device.

Returns

True indicates that the modified configuration will be automatically sent to the device, *false* indicates that you will have to explicitly call *writeConfig()* to send the configuration to the device.

See Also

setAutoConfig(bool autoConfig)

AnalogInputSubsystem& setAutoConfig (bool *autoConfig*) [inline]

Enables or disables automatically sending the modified configuration to the device.

Normally, it's desirable to send the modified configuration to the device immediately. However, in order to reduce the amount of communication with the device while setting multiple properties, this automatic sending mechanism can be disabled and the configuration can be sent by explicitly calling *writeConfig()* once all the properties have been set, like so:

```
device.adc()
  .setAutoConfig( false )
  .setCalMode( AnalogInputSubsystem::CAL_MODE_NORMAL )
  .setTriggerMode( AnalogInputSubsystem::TRIG_MODE_SCAN | AnalogInputSubsystem::TRIG_MODE_TIMER )
  .setOverSample( 50 )
  .writeConfig()
  .setAutoConfig( true );
```

Remember to call setAutoConfig(true) after configuring the properties, otherwise all subsequent configuration changes will have to be explicitly sent to the device by calling writeConfig().

Parameters

autoConfig	*True* enables automatically sending modified configuration, *false* disables it.

Returns

This subsystem, useful for chaining together multiple operations.

AnalogInputSubsystem & readConfig ( )

Reads the A/D configuration from the device.

This is done automatically when the class is instantiated, so it generally does not need to be done again. However, if the A/D configuration in the device has been changed without using this class (e.g. another program changed it), *readConfig()* can be used to copy the device's configuration into this class.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

AnalogInputSubsystem & writeConfig ( )

Writes the A/D configuration to the device.

This is done automatically whenever the pertinent settings within this class are changed. However, if the A/D configuration in the device has been changed without using this class (e.g. another program changed it), or if automatic sending of the configuration has been disabled *(see setAutoConfig(bool autoConfig))*, then *writeConfig()* can be used to copy this class' configuration settings into the device.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

bool isDiscardFirstSample ( ) const

Tells if the *read()*, *readCounts()* and *readVolts()* functions will discard the first A/D sample taken.

Returns

False indicates that no samples will be discarded; *true* indicates that the first sample will be discarded.

AnalogInputSubsystem & setDiscardFirstSample (bool *discard*)

Specifies whether the *read()*, *readCounts()* and *readVolts()* functions will discard the first A/D sample taken.

This setting does **not** pertain to the *readBulkNext()* function which returns all of the raw data captured. Discarding the first sample may be useful in cases in which voltage "residue" from reading a different channel affects the channel currently being read.

Parameters

discard	*false* indicates that no samples will be discarded; *true* indicates that the first sample will be discarded.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

int getCalMode ( ) const `[inline]`

Gets the current calibration mode.

Returns

Current calibration mode (*AnalogInputSubsystem::CAL_MODE_NORMAL*, *AnalogInputSubsystem::CAL_MODE-_GROUND* or *AnalogInputSubsystem::CAL_MODE_REFERENCE*).

See Also

setCalMode(int calMode)

AnalogInputSubsystem & setCalMode (int *calMode*)

Sets the A/D calibration mode.

If ground or reference mode is selected, only one A/D sample may be taken at a time. That means, one channel and no oversampling. Attempting to read more than one channel or use an oversample setting of more than zero will result in a timeout error because the device will not send more than one sample. In order to protect users from accidentally falling into this trap, the *read*()* functions automatically and temporarily correct the scan parameters, restoring them when they complete.

Parameters

calMode	the calibration mode. May be one of: *AnalogInputSubsystem::CAL_MODE_NORMAL* *Analog-InputSubsystem::CAL_MODE_GROUND* *AnalogInputSubsystem::CAL_MODE_REFERENCE*

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

int getTriggerMode ( ) const [inline]

Gets the current trigger mode.

Returns

Current trigger mode (bitwise OR of *TRIG_MODE_CTR0_EXT*, *TRIG_MODE_FALLING_EDGE*, *TRIG_MODE_-SCAN*, *TRIG_MODE_EXTERNAL* or *TRIG_MODE_TIMER*).

See Also

setTriggerMode(int triggerMode)

AnalogInputSubsystem & setTriggerMode (int *triggerMode*)

Sets the trigger mode.

Parameters

triggerMode	a bitwise OR of these flags: *AnalogInputSubsystem::TRIG_MODE_CTR0_EXT Analog-InputSubsystem::TRIG_MODE_FALLING_EDGE AnalogInputSubsystem::TRIG_MODE_SCA-N AnalogInputSubsystem::TRIG_MODE_EXTERNAL AnalogInputSubsystem::TRIG_MODE_T-IMER*

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

int getRange (int *channel*) const

Gets the current range for *channel*.

Parameters

channel	the channel for which to obtain the current range.

Returns

Current range for *channel*.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

IntArray getRange (int *startChannel*, int *numChannels*) const

Gets the current range for multiple A/D channels.

Parameters

startChannel	the first channel for which to obtain the current range.
numChannels	the number of channels for which to obtain the current range.

Returns

Array containing the current ranges for the specified channels.

See Also

setRange(int startChannel, const IntArray &range)

Exceptions

IllegalArgumentException	

AnalogInputSubsystem & setRange (int *channel,* int *range*)

Sets the range for a single A/D channel.

Parameters

channel	the channel for which to set the range.
range	the range (voltage range) for the channel. May be one of: *AnalogInputSubsystem::RANGE_0_1V AnalogInputSubsystem::RANGE_1V AnalogInputSubsystem::RANGE_0_2V AnalogInputSubsystem::RANGE_2V AnalogInputSubsystem::RANGE_0_5V AnalogInputSubsystem::RANGE_5V AnalogInputSubsystem::RANGE_0_10V AnalogInputSubsystem::RANGE_10V*

Returns

This subsystem, useful for chaining together multiple operations.

See Also

setDifferentialMode(int channel, bool differentialMode)

AnalogInputSubsystem & setRange (int *startChannel,* const IntArray & *range*)

Sets the range for multiple A/D channels.

Parameters

startChannel	the first channel for which to set the range.
range	an array of ranges, one per channel *(see setRange(int channel, int range))*.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

bool isDifferentialMode (int *channel*) const

Tells if *channel* is configured for single-ended or differential mode.

Parameters

channel	the channel for which to obtain the current differential mode.

Returns

Current range for *channel*.
False indicates single-ended mode; *true* indicates differential mode.

See Also

setDifferentialMode(int channel, boolean differentialMode)

Exceptions

IllegalArgumentException	

BoolArray isDifferentialMode (int *startChannel,* int *numChannels*) const

Tells if multiple A/D channels are configured for single-ended or differential mode.

Parameters

startChannel	the first channel for which to obtain the current differential mode.
numChannels	the number of channels for which to obtain the current differential mode.

Returns

Array containing the current differential modes for the specified channels. *False* indicates channel is configured for single-ended mode and *true* indicates channel is configured for differential mode.

See Also

setDifferentialMode(int startChannel, const BoolArray &differentialMode)

Exceptions

IllegalArgumentException	

AnalogInputSubsystem & setDifferentialMode (int *channel,* bool *differentialMode*)

Sets a single A/D channel to differential or single-ended mode.

When using differential mode, one should have a good understanding of how the hardware implements it. Considering the simple case of a device with only sixteen input channels, when differential mode is enabled for a channel, that channel is paired with another channel, eight higher than the one for which differential mode is enabled. For instance, if differential mode is enabled for channel 1, then it is paired with channel 9, meaning that channel 1 will return the voltage difference between channels 1 and 9, and channel 9 will no longer return a meaningful reading. This scheme also means that enabling differential mode for channels 8-15 has no effect. In fact, if one attempts to enable differential mode for channels 8-15, nothing happens and if the differential mode setting is read back from the device for those channels, it will likely no longer be enabled! Further confusing matters is that some newer firmware does not clear the differential mode setting for channels 8-15, meaning that it will be returned from the device exactly as set even though it has no effect. So ... one should not rely on the differential mode setting for channels 8-15 to behave in a consistent or predictable manner. For consistency and simplicity, one may read counts or volts from channels 8-15 even while differential mode is enabled, but the readings will not be meaningful. In differential mode, only the base channel (0-7) of the pair that's enabled for differential mode will return a meaningful reading. Channels 8-15 which are not enabled for differential mode will continue to return meaningful readings. For example, if differential mode is enabled for channel 1, then channel 1 will return a meaningful reading, channel 9 will not, and channels 8 and 10-15 will. Considering the more complex case of a device such as the USB-AI16-64MA, which has an additional MUX affording 32 differential, or 64 single-ended inputs, things are a bit more complex. In this case, channels 0-3 share the same differential mode (and range) setting; channels 4-7 share the same setting; and so on. For the sake of simplicity and to support future designs which may have distinct settings for all channels, this software permits the differential mode (and range) to be specified for *any* MUXed channel, even though ultimately multiple channels may share the same setting. For example, on such a device as this, setting the differential mode (or range) of channel 1 also sets the differential mode (or range) of channels 0, 2 and 3. There is yet another case to consider, that of devices such as the USB-AI16-128A. This device may have up to 128 channels, which share settings in groups of eight rather than four on the USB-AI16-64MA. Method *getChannelsPerGroup()* tells how many channels are grouped together on each device, and this topic is discussed more thoroughly in `http://accesio.com/MANUALS/USB-AI FAMILY.PDF`. The foregoing description also applies to the range setting, so one should refer to *setRange(int channel, int range)* as well.

Parameters

channel	the channel for which to set differential or single-ended mode.
differentialMode	*false* selects single-ended mode; *true* selects differential mode.

Returns

This subsystem, useful for chaining together multiple operations.

AnalogInputSubsystem & setDifferentialMode (int *startChannel,* const BoolArray & *differentialMode*)

Sets multiple A/D channels to differential or single-ended mode.

Parameters

startChannel	the first channel for which to set differential or single-ended mode.
differentialMode	an array of mode selectors, one per channel. For each element in the array, *false* selects single-ended mode for that channel and *true* selects differential mode.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

AnalogInputSubsystem & setRangeAndDiffMode (int *channel*, int *range*, bool *differentialMode*)

Sets the range and differential mode for a single A/D channel.

Parameters

channel	the channel for which to set the range.
range	the range (voltage range) for the channel *(see setRange(int channel, int range))*.
differentialMode	*false* selects single-ended mode; *true* selects differential mode.

Returns

This subsystem, useful for chaining together multiple operations.

AnalogInputSubsystem & setRangeAndDiffMode (int *startChannel*, const IntArray & *range*, const BoolArray & *differentialMode*)

Sets the range and differential mode for multiple A/D channels.

Parameters

startChannel	the first channel for which to set the range and differential mode.
range	an array of ranges, one per channel *(see setRange(int channel, int range))*.
differentialMode	an array of mode selectors, one per channel. For each element in the array, *false* selects single-ended mode for that channel and *true* selects differential mode.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

AnalogInputSubsystem & setRangeAndDiffMode (int *range*, bool *differentialMode*)

Sets all the A/D channels to the same range and differential mode.

Parameters

range	the range (voltage range) for the channels *(see setRange(int channel, int range))*.
differentialMode	*false* selects single-ended mode; *true* selects differential mode.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

int getOverSample ( ) const `[inline]`

Gets the current number of over-samples.

Returns

Current number of over-samples (0-255).

See Also

setOverSample(int oversample)

AnalogInputSubsystem & setOverSample (int *overSample*)

Sets the number of over-samples for all A/D channels.

Parameters

overSample	number of over-samples (0-255).

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

AnalogInputSubsystem & setCalibrationTable (const std::string & *fileName*)

Loads a calibration table from a file into the A/D.

Parameters

fileName	the name of a file containing the calibration table. A calibration table must consist of exactly 65,536 16-bit unsigned integers *(see calibrate(bool autoCal, bool returnCalTable, const std::string &saveFileName))*.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

AnalogInputSubsystem & setCalibrationTable (const UShortArray & *calTable*)

Sets the calibration table in the A/D to the contents of *calTable*.

Parameters

calTable	the calibration table to load. A calibration table must consist of exactly 65,536 16-bit unsigned integers *(see calibrate(bool autoCal, bool returnCalTable, const std::string &saveFileName))*.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

int getStreamingBlockSize ( ) `[inline]`

Gets the current streaming block size.

Returns

The current streaming block size. The value returned may not be the same as the value passed to *setStreamingBlockSize(int blockSize)* because that value is rounded up to a whole multiple of 512.

Exceptions

OperationFailedException	

AnalogInputSubsystem& setStreamingBlockSize (int *blockSize*) `[inline]`

Sets the streaming block size.

Parameters

blockSize	the streaming block size you wish to set. This will be rounded up to the next multiple of 512.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

double getClock ( ) [inline]

Gets the current clock frequency for timer-driven bulk reads *(see setClock(double clockHz))*.

Returns

The current frequency at which to take the samples (in Hertz).

AnalogInputSubsystem& setClock (double *clockHz*) [inline]

Sets the clock frequency for timer-driven bulk reads *(see getClock() and readBulkStart(int startChannel, int numChannels, int numSamples))*.

Parameters

clockHz	the frequency at which to take the samples (in Hertz).

Returns

This subsystem, useful for chaining together multiple operations.

UShortArray calibrate (bool *autoCal*, bool *returnCalTable*, const std::string & *saveFileName*)

Calibrates the A/D, generating either a default table or using the internal voltage references to generate a calibration table.

Parameters

autoCal	*true* uses the internal voltage references to automatically calibrate the A/D; *false* generates a default (uncalibrated) table.
returnCalTable	*true* causes *calibrate()* to return the generated calibration table; *false* returns an empty table.
saveFileName	the name of the file in which to save the generated calibration table. If empty, the generated calibration table is not saved to a file.

Returns

If *returnCalTable* is *true*, an array of 65,536 16-bit unsigned integers representing the generated calibration table is returned; otherwise, an empty table (containing zero elements) is returned.

Exceptions

OperationFailedException	

UShortArray calibrate (const DoubleArray & *points*, bool *returnCalTable*, const std::string & *saveFileName*)

Permits the A/D to be calibrated using an external voltage source.

The proper way to use this function is to configure the A/D with a default calibration table (such as by calling *calibrate(bool autoCal, bool returnCalTable, const std::string &saveFileName)*). Then inject a series of voltages into one of the A/D input channels, recording the count values reported by the A/D (by calling *readCounts(int channel)*). It's also a good idea to enable oversampling while recording these values in order to obtain the most stable readings. Alternatively, since *points* is an array of *double* values, you can obtain individual A/D count measurements and average them yourself, producing a *double* average, and put that value into the *points* array. The *points* array consists of voltage-count pairs; *points[0]* is the first input voltage; *points[1]* is the corresponding count value measured by the A/D; *points[2]* and *points[3]* contain the second pair of voltage-count values; and so on. You can provide any number of pairs, although more than a few dozen is probably overkill, not to mention would take a lot of effort to acquire. This calibration procedure uses the current gain A/D setting for channel 0, so it must be the same as that used to collect the measured A/D counts.

It's recommended that all the channels be set to the same gain, the one that will be used during normal operation. The calibration is gain dependent, so switching the gain after calibrating may introduce slight offset or gain changes. So for best results, the A/D should be calibrated on the same gain setting that will be used during normal operation. You can create any number of calibration tables. If your application needs to switch between ranges, you may wish to create a separate calibration table for each range your application will use. Then when switching to a different range, the application can load the appropriate calibration table. Although calibrating in this manner does take some effort, it produces the best results, eliminating all sources of error from the input pins onward. Furthermore, the calibration table can be saved to a file and reloaded into the A/D, ensuring consistency.

Parameters

points	array of voltage-count pairs to calibrate the A/D with.
returnCalTable	*true* causes *calibrate()* to return the generated calibration table; *false* returns an empty table.
saveFileName	the name of the file in which to save the generated calibration table. If empty, the generated calibration table is not saved to a file.

Returns

If *returnCalTable* is *true*, an array of 65,536 16-bit unsigned integers representing the generated calibration table is returned; otherwise, an empty table (containing zero elements) is returned.

Exceptions

IllegalArgumentException	
OperationFailedException	

AI16_DataSet ∗ read (int *startChannel*, int *numChannels*)

Reads from multiple A/D channels and returns a data set containing both the data captured and the parameters in effect at the time the data was captured.

Whereas readCounts(int startChannel, int numChannels) and readVolts(int startChannel, int numChannels) also read data from multiple channels, they return only the raw data. *read()* returns a richer snapshot of the data.

Parameters

startChannel	the first channel to read.
numChannels	the number of channels to read.

Returns

A data set containing the samples and the sampling parameters.

Exceptions

OperationFailedException	

unsigned short readCounts (int *channel*)

Reads the A/D count value from a single channel.

Parameters

channel	the channel to read.

Returns

A/D counts (0-65,535). The meaning of these counts depends on the current A/D range of the channel *(see setRange(int channel, int range))*. The count value may be converted to a voltage value using *countsToVolts(int channel, unsigned short counts) const*.

UShortArray readCounts (int *startChannel*, int *numChannels*)

Reads the A/D count values from multiple channels.

Parameters

startChannel	the first channel to read.

numChannels	the number of channels to read.

Returns

An array of A/D counts (0-65,535), one per channel read. The meaning of these counts depends on the current A/D range of each channel *(see setRange(int channel, int range))*. The array of count values may be converted to an array of voltage values using *countsToVolts(int startChannel, const UShortArray &counts) const*.

Exceptions

OperationFailedException	

double readVolts (int *channel*)

Reads the voltage from a single channel.

Parameters

channel	the channel to read.

Returns

A voltage value, limited to the current A/D range of the channel *(see setRange(int channel, int range))*. The voltage value may be converted to a count value using *voltsToCounts(int channel, double volts) const*.

DoubleArray readVolts (int *startChannel,* int *numChannels*)

Reads the voltage from multiple channels.

Parameters

startChannel	the first channel to read.
numChannels	the number of channels to read.

Returns

An array of voltage values, one per channel read, each limited to the current A/D range of each channel *(see setRange(int channel, int range))*. The array of voltage values may be converted to an array of count values using *voltsToCounts(int startChannel, const DoubleArray &volts) const*.

AnalogInputSubsystem & readBulkStart (int *startChannel,* int *numChannels,* int *numSamples*)

Starts a large A/D acquisition process in a background thread and returns immediately.

The status of the acquisition process can be monitored using *readBulkSamplesAvailable()*, which returns the number of samples available to be retrieved by *readBulkNext(int numSamples)*. When the last of the data has been retrieved using *readBulkNext()*, the bulk acquisition process is automatically terminated and becomes ready to be used again. *While a bulk acquisition process is in progress, no functions of the device other than readBulkSamplesAvailable() or readBulkNext() should be used.* This example shows the proper way to configure the device for a large A/D acquisition process using the internal timer.

```
device.adc()
  .setStreamingBlockSize( 100000 )
  .setCalMode( AnalogInputSubsystem::CAL_MODE_NORMAL )
  .setTriggerMode( AnalogInputSubsystem::TRIG_MODE_SCAN | AnalogInputSubsystem::TRIG_MODE_TIMER )
  .setClock( 100000 )
  .readBulkStart( 1, 1, numSamples );
do {
  UShortArray data = device.adc().readBulkNext( 20000 );
  ... do something with data ...
} while( ...more data is available... );
```

Parameters

startChannel	the first channel to read.
numChannels	the number of channels to read.
numSamples	the total number of *samples* to read.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

int readBulkSamplesAvailable ( )

Gets the number of samples available to be retrieved during a bulk acquisition process initiated by *readBulkStart(int startChannel, int numChannels, int numSamples)*.

Returns

The number of samples available.

Exceptions

OperationFailedException	

UShortArray readBulkNext (int *numSamples*)

Retrieves the next set of samples acquired during a bulk acquisition process initiated by *readBulkStart(int startChannel, int numChannels, int numSamples)*.

Parameters

numSamples	the number of samples to retrieve.

Returns

An array containing the number of samples requested, or all that are available. If fewer samples are available than are requested, only the samples available are returned. If zero samples are available, a zero-length array is returned.

Exceptions

IllegalArgumentException	
OperationFailedException	

AnalogInputSubsystem& clearFIFO (FIFO_Method *method*) `[inline]`

Clears the streaming FIFO, using one of several different methods.

Parameters

method	the method to use when clearing the FIFO. May be one of: *USBDeviceBase::CLEAR_FIFO_METHOD_IMMEDIATE USBDeviceBase::CLEAR_FIFO_METHOD_AUTO USBDeviceBase::CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT USBDeviceBase::CLEAR_FIFO_METHOD_WAIT*

Returns

This subsystem, useful for chaining together multiple operations.

double countsToVolts (int *channel*, unsigned short *counts*) const

Converts a single A/D count value to volts, based on the current gain setting for the specified channel.

Be careful to ensure that the count value was actually obtained from the specified channel and that the gain hasn't changed since the count value was obtained.

Parameters

channel	the channel number to use for converting counts to volts.
counts	the count value to convert to volts.

Returns

A voltage value calculated using the *current* A/D range of the channel *(see setRange(int channel, int range))*.

Exceptions

IllegalArgumentException	

DoubleArray countsToVolts (int *startChannel*, const UShortArray & *counts*) const

Converts an array of A/D count values to an array of voltage values, based on the current gain setting for each of the specified channels.

This method is intended to convert an array of readings from sequential channels, such as might have been obtained from *readCounts(int startChannel, int numChannels)*. Be careful to ensure that the count values were actually obtained from the specified channels and that the gains havn't changed since the count values were obtained.

Parameters

startChannel	the first channel number to use for converting counts to volts.
counts	the count values to convert to volts.

Returns

An array of voltage values calculated using the *current* A/D range of each of the channels *(see setRange(int channel, int range))*. The array returned has the same number of elements as *counts*.

Exceptions

IllegalArgumentException	
OperationFailedException	

unsigned short voltsToCounts (int *channel*, double *volts*) const

Converts a single voltage value to A/D counts, based on the current gain setting for the specified channel.

Be careful to ensure that the voltage value was actually obtained from the specified channel and that the gain hasn't changed since the voltage value was obtained.

Parameters

channel	the channel number to use for converting volts to counts.
volts	the voltage value to convert to counts.

Returns

A count value calculated using the *current* A/D range of the channel *(see setRange(int channel, int range))*.

Exceptions

IllegalArgumentException	

UShortArray voltsToCounts (int *startChannel*, const DoubleArray & *volts*) const

Converts an array of voltage values to an array of A/D count values, based on the current gain setting for each of the specified channels.

This method is intended to convert an array of readings from sequential channels, such as might have been obtained from *readVolts(int startChannel, int numChannels)*. Be careful to ensure that the voltage values were actually obtained from the specified channels and that the gains havn't changed since the voltage values were obtained.

Parameters

startChannel	the first channel number to use for converting volts to counts.
volts	the voltage values to convert to counts.

Returns

An array of count values calculated using the *current* A/D range of each of the channels *(see setRange(int channel, int range))*. The array returned has the same number of elements as *volts*.

Exceptions

IllegalArgumentException	
OperationFailedException	

23.32.4 Friends And Related Function Documentation

friend class USB_AI16_Family `[friend]`

23.32.5 Field Documentation

const int CAL_MODE_NORMAL = 0 `[static]`

Selects normal measurement mode *(see setCalMode(int calMode))*.

const int CAL_MODE_GROUND = 1 `[static]`

Selects ground calibration mode *(see setCalMode(int calMode))*.

const int CAL_MODE_REFERENCE = 3 `[static]`

Selects reference (full scale) calibration mode *(see setCalMode(int calMode))*.

const int TRIG_MODE_CTR0_EXT = 0x10 `[static]`

If set, counter 0 is externally triggered *(see setTriggerMode(int triggerMode))*.

const int TRIG_MODE_FALLING_EDGE = 0x08 `[static]`

If set, the A/D is triggered by the falling edge of its trigger source, otherwise it's triggered by the rising edge *(see setTriggerMode(int triggerMode))*.

const int TRIG_MODE_SCAN = 0x04 `[static]`

If set, each trigger will cause the A/D to scan all the channels, otherwise the A/D will read a single channel with each trigger *(see setTriggerMode(int triggerMode))*.

const int TRIG_MODE_EXTERNAL = 0x02 `[static]`

If set, the A/D is triggered by an external pin on the board *(see setTriggerMode(int triggerMode))*.

const int TRIG_MODE_TIMER = 0x01 `[static]`

If set, the A/D is triggered by counter 2 *(see setTriggerMode(int triggerMode))*.

const int RANGE_0_10V = 0 `[static]`

Unipolar, 0-10 volt range *(see setRange(int channel, int range))*.

const int RANGE_10V = 1 `[static]`

Bipolar, -10 to +10 volt range *(see setRange(int channel, int range))*.

const int RANGE_0_5V = 2 `[static]`

Unipolar, 0-5 volt range *(see setRange(int channel, int range))*.

const int RANGE_5V = 3 `[static]`

Bipolar, -5 to +5 volt range *(see setRange(int channel, int range))*.

const int RANGE_0_2V = 4 `[static]`

Unipolar, 0-2 volt range *(see setRange(int channel, int range))*.

const int RANGE_2V = 5 `[static]`

Bipolar, -2 to +2 volt range *(see setRange(int channel, int range))*.

const int RANGE_0_1V = 6 `[static]`

Unipolar, 0-1 volt range *(see setRange(int channel, int range))*.

const int RANGE_1V = 7 `[static]`

Bipolar, -1 to +1 volt range *(see setRange(int channel, int range))*.

const int MIN_COUNTS = 0 `[static]`

Minimum number of counts A/D can read.

const int MAX_COUNTS = 0xffff `[static]`

Maximum number of counts A/D can read.

const int CAL_TABLE_WORDS = 64 ∗ 1024 `[static]`

Number of 16-bit words in an A/D calibration table (65,536 16-bit words).

const char RANGE_TEXT `[static]`, `[protected]`

Initial value:

```
= {
          "0-10V"
        , "+/-10V"
        , "0-5V"
        , "+/-5V"
        , "0-2V"
        , "+/-2V"
        , "0-1V"
        , "+/-1V"
}
```

const int NUM_CONFIG_REGISTERS = 20 `[static]`, `[protected]`

const int NUM_MUX_CONFIG_REGISTERS = 21 `[static]`, `[protected]`

const int NUM_GAIN_CODE_REGISTERS = 16 `[static]`, `[protected]`

const int REG_GAIN_CODE = 0 `[static]`, `[protected]`

const int REG_CAL_MODE = 16 `[static]`, `[protected]`

const int REG_TRIG_MODE = 17 `[static]`, `[protected]`

const int REG_START_END = 18 `[static]`, `[protected]`

const int REG_OVERSAMPLE = 19 `[static]`, `[protected]`

const int REG_MUX_START_END = 20 `[static]`, `[protected]`

const int DIFFERENTIAL_MODE = 8 `[static]`, `[protected]`

const int MAX_OVERSAMPLE = 0xff `[static]`, `[protected]`

const int TRIG_MODE_VALID_MASK `[static]`, `[protected]`

Initial value:

```
= (
                TRIG_MODE_CTR0_EXT
                | TRIG_MODE_FALLING_EDGE
                | TRIG_MODE_SCAN
                | TRIG_MODE_EXTERNAL
                | TRIG_MODE_TIMER )
```

const int AUTO_CAL_UNKNOWN = 0 `[static], [protected]`

const int AUTO_CAL_NOT_PRESENT = 1 `[static], [protected]`

const int AUTO_CAL_PRESENT = 2 `[static], [protected]`

const int MAX_CHANNELS = 128 `[static], [protected]`

int numChannels `[protected]`

int numMUXChannels `[protected]`

int channelsPerGroup `[protected]`

int configBlockSize `[protected]`

int autoCalFeature `[protected]`

AI16_InputRange∗ inputRange `[protected]`

bool∗ differentialMode `[protected]`

int calMode `[protected]`

int triggerMode `[protected]`

int startChannel `[protected]`

int endChannel `[protected]`

int overSample `[protected]`

unsigned short∗ readBulkBuffer `[protected]`

int readBulkSamplesRequested `[protected]`

int readBulkSamplesRetrieved `[protected]`

bool autoConfig `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/AnalogInputSubsystem.hpp
- deprecated/classlib/AnalogInputSubsystem.cpp

23.33 AnalogIORange Class Reference

Class AnalogIORange helps manage analog I/O range settings and provides voltage-count conversion utilities.

`#include <AnalogIORange.hpp>`

Public Member Functions

- AnalogIORange ()

  Constructor which uses the default properties.
- AnalogIORange (int minCounts, int maxCounts)

  Constructor which sets the count range.
- virtual ∼AnalogIORange ()
- int getRange () const

  Gets the current range ID.
- virtual AnalogIORange & setRange (int range)

Sets the range ID.

- AnalogIORange & setCountRange (int minCounts, int maxCounts)

  Sets the A/D or D/A count range.
- AnalogIORange & setVoltRange (double minVolts, double maxVolts)

  Sets the voltage range.
- double countsToVolts (int counts) const

  Converts a single A/D or D/A count value to volts, based on the current range setting.
- int voltsToCounts (double volts) const

  Converts a single voltage value to A/D or D/A counts, based on the current range setting.

Protected Attributes

- int range
- int minCounts
- int maxCounts
- int rangeCounts
- double minVolts
- double maxVolts
- double rangeVolts

23.33.1 Detailed Description

Class AnalogIORange helps manage analog I/O range settings and provides voltage-count conversion utilities.

A single instance can be used with devices that support just one range, or multiple instances can be used with devices that support multiple ranges, such as a separate range per analog I/O channel. This class also supports changing the range properties. Some devices, for instance, permit the range to be changed at run-time. The class that owns this instance can change the range by calling one or more of the methods of this class. Or, for devices that do not support changing the range, the properties can be set up once and left alone. Or, some properties can be changed and others left alone. For example, devices that permit changing the voltage range usually use a fixed count range.

23.33.2 Constructor & Destructor Documentation

AnalogIORange ( )

Constructor which uses the default properties.

AnalogIORange (int *minCounts*, int *maxCounts*)

Constructor which sets the count range.

The count range is usually constant, so setting it one time in the constructor is convenient.

Parameters

minCounts	minimum counts for current range.
maxCounts	maximum counts for current range.

∼AnalogIORange ( ) `[virtual]`

23.33.3 Member Function Documentation

int getRange ( ) const `[inline]`

Gets the current range ID.

Returns

Current range ID (defined by class that owns this instance).

AnalogIORange & setRange (int *range*) `[virtual]`

Sets the range ID.

Parameters

range	the new range ID (defined by class that owns this instance).

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented in AI16_InputRange, AO16_OutputRange, and DA12_OutputRange.

AnalogIORange & setCountRange (int *minCounts,* int *maxCounts*)

Sets the A/D or D/A count range.

Parameters

minCounts	minimum counts for current range.
maxCounts	maximum counts for current range.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

AnalogIORange & setVoltRange (double *minVolts,* double *maxVolts*)

Sets the voltage range.

Parameters

minVolts	minimum volts for current range.
maxVolts	maximum volts for current range.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

double countsToVolts (int *counts*) const

Converts a single A/D or D/A count value to volts, based on the current range setting.

Parameters

counts	the count value to convert to volts.

Returns

A voltage value calculated using the current range. The voltage value returned is constrained to the current minimum-maximum voltage range.

int voltsToCounts (double *volts*) const

Converts a single voltage value to A/D or D/A counts, based on the current range setting.

Parameters

volts	the voltage value to convert to counts.

Returns

A count value calculated using the current D/A range. The count value returned is constrained to the current minimum-maximum count range.

23.33.4 Field Documentation

int range [protected]

int minCounts [protected]

int maxCounts [protected]

int rangeCounts [protected]

double minVolts [protected]

double maxVolts [protected]

double rangeVolts [protected]

The documentation for this class was generated from the following files:

- deprecated/classlib/AnalogIORange.hpp
- deprecated/classlib/AnalogIORange.cpp

23.34 AnalogOutputSubsystem Class Reference

Class AnalogOutputSubsystem is the superclass of the analog output subsystem of a device.

`#include <AnalogOutputSubsystem.hpp>`

Public Member Functions

- AnalogOutputSubsystem (USBDeviceBase &parent)
- virtual ∼AnalogOutputSubsystem ()
- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this subsystem.
- int getNumChannels () const

  Gets the number of analog output channels.
- AnalogOutputSubsystem & writeCounts (int channel, unsigned short counts)

  Writes a count value to a D/A channel.
- AnalogOutputSubsystem & writeCounts (const UShortArray &points)

  Writes a block of count values to one or more D/A channels.

Protected Attributes

- int numChannels
- int minCounts
- int maxCounts

Additional Inherited Members

23.34.1 Detailed Description

Class AnalogOutputSubsystem is the superclass of the analog output subsystem of a device.

This class provides basic features, such as writing count values to the D/As. More sophisticated functions are provided by device-specific subclasses. One accesses this analog output subsystem through its parent object, typically through a method such as *dac() (see USB_AO16_Family::dac())*.

23.34.2 Constructor & Destructor Documentation

AnalogOutputSubsystem (USBDeviceBase & *parent*)

∼AnalogOutputSubsystem ( ) [virtual]

23.34.3 Member Function Documentation

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumChannels ( ) const [inline]

Gets the number of analog output channels.

Returns

Number of channels, numbered 0 to n-1.

AnalogOutputSubsystem & writeCounts (int *channel*, unsigned short *counts*)

Writes a count value to a D/A channel.

Parameters

channel	the channel to write to.
counts	the D/A count value to output. The number of bits of resolution for the D/A outputs varies from model to model, however it's usually 12 or 16 bits. Moreover, some of the 12-bit models actually accept a 16-bit value and simply truncate the least significant 4 bits. Consult the manual for the specific device to determine the range of D/A values the device will accept. In general, 12-bit devices accept a count range of 0-0xfff, and 16-bit devices accept a count range of 0-0xffff.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

AnalogOutputSubsystem & writeCounts (const UShortArray & *points*)

Writes a block of count values to one or more D/A channels.

Parameters

points	an array of 16-bit integers representing channel-count pairs. The first integer of each pair is the D/A channel number and the second integer is the D/A count value to output to the specified channel. Refer to *writeCounts(int channel, unsigned short counts)* for an explanation of the channel addressing and count values.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

23.34.4 Field Documentation

int numChannels [protected]

int minCounts [protected]

int maxCounts [protected]

The documentation for this class was generated from the following files:

- deprecated/classlib/AnalogOutputSubsystem.hpp
- deprecated/classlib/AnalogOutputSubsystem.cpp

23.35 AO16_AnalogOutputSubsystem Class Reference

Class AO16_AnalogOutputSubsystem represents the analog output subsystem of a device.

```
#include <AO16_AnalogOutputSubsystem.hpp>
```

Public Member Functions

- int getRange () const
  Gets the current voltage range of the D/A outputs.
- AO16_AnalogOutputSubsystem & setRange (int range)
  Sets the voltage range of the D/A outputs.
- AO16_AnalogOutputSubsystem & writeVolts (int channel, double volts)
  Writes a voltage value to a D/A channel.
- AO16_AnalogOutputSubsystem & writeVolts (const OutputVoltagePointArray &points)
  Writes a block of voltage values to one or more D/A channels.
- double countsToVolts (unsigned short counts) const
  Converts a single D/A count value to volts, based on the current range setting.
- unsigned short voltsToCounts (double volts) const
  Converts a single voltage value to D/A counts, based on the current range setting.

Static Public Member Functions

- static std::string getRangeText (int range)
  Gets the textual string for the specified range.

Static Public Attributes

- static const int RANGE_0_5V = 0
  Unipolar, 0-5 volt range (see setRange(int range)).
- static const int RANGE_5V = 1
  Bipolar, -5 to +5 volt range (see setRange(int range)).
- static const int RANGE_0_10V = 2
  Unipolar, 0-10 volt range (see setRange(int range)).
- static const int RANGE_10V = 3
  Bipolar, -10 to +10 volt range (see setRange(int range)).
- static const int MIN_COUNTS = 0
  Minimum number of counts D/A can output.
- static const int MAX_COUNTS = 0xffff
  Maximum number of counts D/A can output.

Protected Member Functions

- AO16_AnalogOutputSubsystem (USBDeviceBase &parent)
- virtual ~AO16_AnalogOutputSubsystem ()

Protected Attributes

- AO16_OutputRange outputRange

Static Protected Attributes

- static const char RANGE_TEXT [][10]

Friends

- class USB_AO16_Family

23.35.1 Detailed Description

Class AO16_AnalogOutputSubsystem represents the analog output subsystem of a device.

One accesses this analog output subsystem through its parent object, typically through a method such as *dac() (see USB_AO16_Family::dac())*.

23.35.2 Constructor & Destructor Documentation

AO16_AnalogOutputSubsystem (USBDeviceBase & *parent*) [protected]

~AO16_AnalogOutputSubsystem ( ) [protected], [virtual]

23.35.3 Member Function Documentation

std::string getRangeText (int *range*) [static]

Gets the textual string for the specified range.

Parameters

range	the range for which to obtain the textual string.

Returns

The textual string for the specified range.

See Also

setRange(int range)

Exceptions

IllegalArgumentException	

int getRange ( ) const [inline]

Gets the current voltage range of the D/A outputs.

Returns

Current voltage range.

See Also

setRange(int range)

AO16_AnalogOutputSubsystem & setRange (int *range*)

Sets the voltage range of the D/A outputs.

Parameters

range	the voltage range to select. May be one of: *AO16_AnalogOutputSubsystem::RANGE_0_5V* *AO16_AnalogOutputSubsystem::RANGE_5V* *AO16_AnalogOutputSubsystem::RANGE_0_10V* *AO16_AnalogOutputSubsystem::RANGE_10V*

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

AO16_AnalogOutputSubsystem & writeVolts (int *channel*, double *volts*)

Writes a voltage value to a D/A channel.

Parameters

channel	the channel to write to.
volts	the voltage value to output.

Returns

This subsystem, useful for chaining together multiple operations.

AO16_AnalogOutputSubsystem & writeVolts (const OutputVoltagePointArray & *points*)

Writes a block of voltage values to one or more D/A channels.

Parameters

points	an array of OutputVoltagePoint points representing channel-voltage pairs.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

double countsToVolts (unsigned short *counts*) const `[inline]`

Converts a single D/A count value to volts, based on the current range setting.

Parameters

counts	the count value to convert to volts.

Returns

A voltage value calculated using the current D/A range. The voltage value returned is constrained to the current minimum-maximum voltage range of the D/A. *(see setRange(int range))*.

unsigned short voltsToCounts (double *volts*) const `[inline]`

Converts a single voltage value to D/A counts, based on the current range setting.

Parameters

volts	the voltage value to convert to counts.

Returns

A count value calculated using the current D/A range. The count value returned is constrained to the current minimum-maximum count range of the D/A. *(see setRange(int range))*.

23.35.4 Friends And Related Function Documentation

friend class USB_AO16_Family `[friend]`

23.35.5 Field Documentation

const int RANGE_0_5V = 0 `[static]`

Unipolar, 0-5 volt range *(see setRange(int range))*.

const int RANGE_5V = 1 `[static]`

Bipolar, -5 to +5 volt range *(see setRange(int range))*.

const int RANGE_0_10V = 2 `[static]`

Unipolar, 0-10 volt range *(see setRange(int range))*.

const int RANGE_10V = 3 `[static]`

Bipolar, -10 to +10 volt range *(see setRange(int range))*.

const int MIN_COUNTS = 0 `[static]`

Minimum number of counts D/A can output.

const int MAX_COUNTS = 0xffff `[static]`

Maximum number of counts D/A can output.

const char RANGE_TEXT `[static], [protected]`

Initial value:

```
= {
          "0-5V"
        , "+/-5V"
        , "0-10V"
        , "+/-10V"
}
```

AO16_OutputRange outputRange `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/AO16_AnalogOutputSubsystem.hpp
- deprecated/classlib/AO16_AnalogOutputSubsystem.cpp

23.36 AO16_OutputRange Class Reference

```
#include <AO16_OutputRange.hpp>
```

Public Member Functions

- virtual AnalogIORange & setRange (int range)
  Sets the range ID.

Protected Member Functions

- AO16_OutputRange ()
- AO16_OutputRange (int minCounts, int maxCounts)
- virtual ∼AO16_OutputRange ()

Friends

- class AO16_AnalogOutputSubsystem

Additional Inherited Members

23.36.1 Constructor & Destructor Documentation

AO16_OutputRange ( ) `[protected]`

AO16_OutputRange (int *minCounts*, int *maxCounts*) `[protected]`

∼AO16_OutputRange ( ) `[protected], [virtual]`

23.36.2 Member Function Documentation

AnalogIORange & setRange (int *range*) `[virtual]`

Sets the range ID.

Parameters

range	the new range ID (defined by class that owns this instance).

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented from AnalogIORange.

23.36.3 Friends And Related Function Documentation

friend class AO16_AnalogOutputSubsystem `[friend]`

The documentation for this class was generated from the following files:

- deprecated/classlib/AO16_OutputRange.hpp
- deprecated/classlib/AO16_OutputRange.cpp

23.37 BoolArray Class Reference

```
#include <USBDeviceBase.hpp>
```

Public Member Functions

- BoolArray (int size=0)

23.37.1 Constructor & Destructor Documentation

BoolArray (int *size* = `0` **)** `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceBase.hpp

23.38 BulkAcquireWorkerParams Struct Reference

```
#include <AIOUSB_Core.h>
```

Data Fields

- unsigned long DeviceIndex
- unsigned long BufSize
- void ∗ pBuf

23.38.1 Field Documentation

unsigned long DeviceIndex

unsigned long BufSize

void∗ pBuf

The documentation for this struct was generated from the following file:

- lib/AIOUSB_Core.h

23.39 channel_range Struct Reference

```
#include <aiocommon.h>
```

Data Fields

- int start_channel
- int end_channel
- int gaincode

23.39.1 Field Documentation

int start_channel

int end_channel

int gaincode

The documentation for this struct was generated from the following file:

- samples/TestLib/aiocommon.h

23.40 cJSON Struct Reference

```
#include <cJSON.h>
```

Data Fields

- struct cJSON ∗ next
- struct cJSON ∗ prev
- struct cJSON ∗ child
- int type
- char ∗ valuestring
- int valueint
- double valuedouble
- char ∗ string

23.40.1 Field Documentation

struct cJSON∗ next

struct cJSON ∗ prev

struct cJSON∗ child

int type

char∗ valuestring

int valueint

double valuedouble

char∗ string

The documentation for this struct was generated from the following file:

- lib/cJSON.h

23.41 cJSON_Hooks Struct Reference

```
#include <cJSON.h>
```

Data Fields

- void ∗(∗ malloc_fn)(size_t sz)
- void(∗ free_fn)(void ∗ptr)

23.41.1 Field Documentation

void∗(∗ malloc_fn)(size_t sz)

void(∗ free_fn)(void ∗ptr)

The documentation for this struct was generated from the following file:

- lib/cJSON.h

23.42 config_options Struct Reference

Data Fields

- unsigned long targetSerialNumber
- unsigned long framePoints
- int buffer_size
- int clock_rate
- int number_channels
- int write_clock_rate

23.42.1 Field Documentation

unsigned long targetSerialNumber

unsigned long framePoints

int buffer_size

int clock_rate

int number_channels

int write_clock_rate

The documentation for this struct was generated from the following file:

- samples/USB-AI16-16/HOLD/slow_receiver_test.cpp

23.43 configuration Struct Reference

```
#include <AIOConfiguration.h>
```

Data Fields

- ConfigurationType type
- int timeout
- int discard_first_sample
- int device_index
- int number_scans
- ADCSetCalFunction calibration
- ADCScanType scan_type
- char ∗ calibration_file
- int debug
- char ∗ output_file
- FILE ∗ file_handle
- char ∗ file_name
- AIORET_TYPE(∗ configure)(struct configuration ∗)
- AIORET_TYPE(∗ run)(struct configuration ∗)

23.43.1 Field Documentation

ConfigurationType type

int timeout

int discard_first_sample

int device_index

int number_scans

ADCSetCalFunction calibration

ADCScanType scan_type

char∗ calibration_file

int debug

char∗ output_file

FILE∗ file_handle

char∗ file_name

AIORET_TYPE(∗ configure)(struct configuration ∗)

AIORET_TYPE(∗ run)(struct configuration ∗)

The documentation for this struct was generated from the following file:

- lib/AIOConfiguration.h

23.44 Counter Class Reference

Class Counter represents a single counter/timer.

```
#include <Counter.hpp>
```

Public Member Functions

- int getDeviceIndex () const

  Gets the index of the parent device on the USB bus.
- Counter & setMode (int mode)

  Sets the counter's mode.
- Counter & setCount (unsigned short count)

  Loads a count value into the counter.
- Counter & setModeAndCount (int mode, unsigned short count)

  Sets a counter mode and loads a count value into the counter.
- unsigned short readCount ()

  Reads a counter's current count value.
- UShortArray readCountAndStatus ()

  Reads a counter's current count value and status.
- unsigned short readCountAndSetModeAndCount (int mode, unsigned short count)

  Reads a counter's current count value, then sets a new mode and loads a new count value into the counter.

Static Public Attributes

- static const int MODE_TERMINAL_COUNT = 0

  Mode 0: interrupt on terminal count (see setMode(int mode)).
- static const int MODE_ONE_SHOT = 1

  Mode 1: hardware retriggerable one-shot (see setMode(int mode)).
- static const int MODE_RATE_GENERATOR = 2

  Mode 2: rate generator (see setMode(int mode)).

- static const int MODE_SQUARE_WAVE = 3

  Mode 3: square wave mode (see setMode(int mode)).
- static const int MODE_SW_TRIGGERED = 4

  Mode 4: software triggered mode (see setMode(int mode)).
- static const int MODE_HW_TRIGGERED = 5

  Mode 5: hardware triggered strobe (retriggerable) (see setMode(int mode)).

Protected Member Functions

- Counter (CounterSubsystem ∗parent, int counterIndex)

Protected Attributes

- CounterSubsystem ∗ parent
- int counterIndex

Friends

- class CounterSubsystem

23.44.1 Detailed Description

Class Counter represents a single counter/timer.

One accesses a counter through its CounterSubsystem parent object *(see CounterSubsystem::getCounter(int counter)).*

23.44.2 Constructor & Destructor Documentation

Counter (CounterSubsystem ∗ *parent,* int *counterIndex*) `[protected]`

23.44.3 Member Function Documentation

int getDeviceIndex ( ) const

Gets the index of the parent device on the USB bus.

Used internally in calls to the underlying API.

Returns

The index of the parent device on the USB bus.

Counter & setMode (int *mode*)

Sets the counter's mode.

Parameters

mode	the counter mode. May be one of: *Counter::MODE_TERMINAL_COUNT Counter::MODE_O-NE_SHOT Counter::MODE_RATE_GENERATOR Counter::MODE_SQUARE_WAVE Counter-::MODE_SW_TRIGGERED Counter::MODE_HW_TRIGGERED*

Returns

This counter, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

Counter & setCount (unsigned short *count*)

Loads a count value into the counter.

Parameters

count	the count value (0-65,535) to load into the counter.

Returns

This counter, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

Counter & setModeAndCount (int *mode,* unsigned short *count*)

Sets a counter mode and loads a count value into the counter.

Parameters

mode	the counter mode *(see setMode(int mode)).*
count	the count value (0-65,535) to load into the counter.

Returns

This counter, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

unsigned short readCount ( )

Reads a counter's current count value.

Returns

The current count value (0-65,535).

Exceptions

OperationFailedException	

UShortArray readCountAndStatus ( )

Reads a counter's current count value and status.

Returns

An array of 2 16-bit integers: char[0] contains the current count value (0-65,535) char[1] contains the current counter status (0-255)

Exceptions

OperationFailedException	

unsigned short readCountAndSetModeAndCount (int *mode,* unsigned short *count*)

Reads a counter's current count value, *then* sets a new mode and loads a new count value into the counter.

Parameters

mode	the counter mode *(see setMode(int mode)).*
count	the count value (0-65,535) to load into the counter.

Returns

The 16-bit count value (0-65,535) prior to setting the new mode and count.

Exceptions

IllegalArgumentException	
OperationFailedException	

23.44.4 Friends And Related Function Documentation

friend class CounterSubsystem `[friend]`

23.44.5 Field Documentation

const int MODE_TERMINAL_COUNT = 0 `[static]`

Mode 0: interrupt on terminal count *(see setMode(int mode))*.

const int MODE_ONE_SHOT = 1 `[static]`

Mode 1: hardware retriggerable one-shot *(see setMode(int mode))*.

const int MODE_RATE_GENERATOR = 2 `[static]`

Mode 2: rate generator *(see setMode(int mode))*.

const int MODE_SQUARE_WAVE = 3 `[static]`

Mode 3: square wave mode *(see setMode(int mode))*.

const int MODE_SW_TRIGGERED = 4 `[static]`

Mode 4: software triggered mode *(see setMode(int mode))*.

const int MODE_HW_TRIGGERED = 5 `[static]`

Mode 5: hardware triggered strobe (retriggerable) *(see setMode(int mode))*.

CounterSubsystem* parent `[protected]`

int counterIndex `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/Counter.hpp
- deprecated/classlib/Counter.cpp

23.45 CounterList Class Reference

`#include <Counter.hpp>`

The documentation for this class was generated from the following file:

- deprecated/classlib/Counter.hpp

23.46 CounterSubsystem Class Reference

Class CounterSubsystem represents the counter/timer subsystem of a device.

`#include <CounterSubsystem.hpp>`

Public Member Functions

- CounterSubsystem (USBDeviceBase &parent)
- virtual ∼CounterSubsystem ()
- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this subsystem.
- int getNumCounterBlocks () const

  Gets the number of counter blocks.
- int getNumCounters () const

  Gets the number of individual counters, indexed from 0 to n-1.
- Counter & getCounter (int counter)

  Gets a reference to an individual counter.
- UShortArray readCounts (bool oldData)

  Reads the current count values of all the counters, optionally including an "old data" indication.
- CounterSubsystem & selectGate (int counter)

  Selects the counter to use as a gate in frequency measurement on other counters.
- double startClock (int counterBlock, double clockHz)

  Selects an output frequency for a counter block and starts the counters.
- CounterSubsystem & stopClock (int counterBlock)

  Halts the counter started by startClock(int counterBlock, double clockHz)

Protected Attributes

- int numCounterBlocks
- int numCounters
- CounterList counters

Static Protected Attributes

- static const int COUNTERS_PER_BLOCK = 3

Friends

- class Counter
- class USB_AI16_Family
- class USB_CTR_15_Family
- class USB_DIO_32_Family

Additional Inherited Members

23.46.1 Detailed Description

Class CounterSubsystem represents the counter/timer subsystem of a device.

One accesses this counter/timer subsystem through its parent object, typically through a method such as *ctr() (see USB_AI16_Family::ctr())*.

23.46.2 Constructor & Destructor Documentation

CounterSubsystem (USBDeviceBase & *parent*)

∼CounterSubsystem ( ) `[virtual]`

23.46.3 Member Function Documentation

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumCounterBlocks ( ) const `[inline]`

Gets the number of counter blocks.

Typically there are three counters per counter block.

Returns

The number of counter blocks.

int getNumCounters ( ) const `[inline]`

Gets the number of individual counters, indexed from 0 to n-1.

Returns

The number of individual counters.

Counter & getCounter (int *counter*)

Gets a reference to an individual counter.

You must obtain a reference to a counter before you can perform counter operations.

Parameters

counter	the counter for which to obtain a reference (0 to n-1).

Returns

A reference to the specified counter.

Exceptions

IllegalArgumentException	

UShortArray readCounts (bool *oldData*)

Reads the current count values of all the counters, optionally including an "old data" indication.

If *oldData* is *true*, then an extra word will be returned (one word for each counter, plus one extra word) that contains an "old data" indication, which is useful for optimizing polling rates. If the value of the final word is zero, then the data is "old data," meaning you are polling the counters faster than your gate signal is running. If *oldData* is *false*, then only the count values are returned.

Parameters

oldData	*true* includes the "old data" indication in the returned data; *false* returns just the count values.

Returns

An array containing the current count values for all the counters plus an optional "old data" indication in the final word.

Exceptions

OperationFailedException	

CounterSubsystem & selectGate (int *counter*)

Selects the counter to use as a gate in frequency measurement on other counters.

Parameters

counter	the counter to select as a gate (0 to n-1).

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

double startClock (int *counterBlock,* double *clockHz*)

Selects an output frequency for a counter block and starts the counters.

selectGate(int counter) and *readCounts(bool oldData)* are used in measuring frequency. To measure frequency one must count pulses for a known duration. In simplest terms, the number of pulses that occur for 1 second translates directly to Hertz. In the USB-CTR-15 and other supported devices, you can create a known duration by configuring one counter to act as a "gating" signal for any collection of other counters. The other "measurement" counters will only count during the "high" side of the gate signal, which we can control. So, to measure frequency you:

1. Create a gate signal of known duration
2. Connect this gating signal to the gate pins of all the "measurement" counters
3. Call *selectGate()* to tell the board which counter is generating that gate
4. Call *readCounts(true)* periodically to read the latched count values from all the "measurement" counters. In practice, it may not be possible to generate a gating signal of sufficient duration from a single counter. Simply concatenate two or more counters into a series, or daisy-chain, and use the last counter's output as your gating signal. This last counter in the chain should be selected as the "gate source" using *selectGate(int counter)*. Once a value has been read from a counter using the *readCounts(true)* call, it can be translated into actual Hz by dividing the count value returned by the high-side-duration of the gating signal, in seconds. For example, if your gate is configured for 10Hz, the high-side lasts 0.05 seconds. If you read 1324 counts via the *readCounts(true)* call, the frequency would be "1324 / 0.05", or 26.48 KHz.

   Parameters

counterBlock	the counter block to use to generate the frequency.
clockHz	the desired output frequency (in Hertz).

   Returns

   The actual frequency that will be generated, limited by the device's capabilities.

   Exceptions

IllegalArgumentException	
OperationFailedException	

CounterSubsystem& stopClock (int *counterBlock*) [inline]

Halts the counter started by *startClock(int counterBlock, double clockHz)*

Parameters

counterBlock	the counter block to halt generating a frequency.

Returns

This subsystem, useful for chaining together multiple operations.

23.46.4 Friends And Related Function Documentation

friend class Counter [friend]

friend class USB_AI16_Family [friend]

friend class USB_CTR_15_Family [friend]

friend class USB_DIO_32_Family [friend]

23.46.5 Field Documentation

const int COUNTERS_PER_BLOCK = 3 `[static], [protected]`

int numCounterBlocks `[protected]`

int numCounters `[protected]`

CounterList counters `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/CounterSubsystem.hpp
- deprecated/classlib/CounterSubsystem.cpp

23.47 CStringArray Struct Reference

`#include <CStringArray.h>`

The documentation for this struct was generated from the following file:

- lib/CStringArray.h

23.48 DA12_AnalogOutputSubsystem Class Reference

Class DA12_AnalogOutputSubsystem represents the analog output subsystem of a device.

`#include <DA12_AnalogOutputSubsystem.hpp>`

Public Member Functions

- int getRange (int channel) const
  Gets the current voltage range of a D/A channel.
- IntArray getRange (int startChannel, int numChannels) const
  Gets the current voltage range of multiple D/A channels.
- DA12_AnalogOutputSubsystem & setRange (int channel, int range)
  Sets the voltage range of a D/A channel.
- DA12_AnalogOutputSubsystem & setRange (int startChannel, const IntArray &range)
  Sets the current voltage range of multiple D/A channels.
- DA12_AnalogOutputSubsystem & setRange (int range)
  Sets the current voltage range of all D/A channels to the same value.
- DA12_AnalogOutputSubsystem & writeVolts (int channel, double volts)
  Writes a voltage value to a D/A channel.
- DA12_AnalogOutputSubsystem & writeVolts (const OutputVoltagePointArray &points)
  Writes a block of voltage values to one or more D/A channels.
- double countsToVolts (int channel, unsigned short counts) const
  Converts a single D/A count value to volts, based on the current range setting.
- unsigned short voltsToCounts (int channel, double volts) const
  Converts a single voltage value to D/A counts, based on the current range setting.

Static Public Member Functions

- static std::string getRangeText (int range)
  Gets the textual string for the specified range.

Static Public Attributes

- static const int RANGE_0_2_5V = 0
  Unipolar, 0-2.5 volt range (see setRange(int channel, int range)).
- static const int RANGE_2_5V = 1
  Bipolar, -2.5 to +2.5 volt range (see setRange(int channel, int range)).
- static const int RANGE_0_5V = 2
  Unipolar, 0-5 volt range (see setRange(int channel, int range)).

- static const int RANGE_5V = 3

  Bipolar, -5 to +5 volt range (see setRange(int channel, int range)).
- static const int RANGE_0_10V = 4

  Unipolar, 0-10 volt range (see setRange(int channel, int range)).
- static const int RANGE_10V = 5

  Bipolar, -10 to +10 volt range (see setRange(int channel, int range)).
- static const int MIN_COUNTS = 0

  Minimum number of counts D/A can output.
- static const int MAX_COUNTS = 0xfff

  Maximum number of counts D/A can output.

Protected Member Functions

- DA12_AnalogOutputSubsystem (USBDeviceBase &parent)
- virtual ∼DA12_AnalogOutputSubsystem ()

Protected Attributes

- DA12_OutputRange ∗ outputRange

Static Protected Attributes

- static const char RANGE_TEXT [][10]

Friends

- class USB_DA12_8A_Family
- class USB_DA12_8E_Family

23.48.1 Detailed Description

Class DA12_AnalogOutputSubsystem represents the analog output subsystem of a device.

One accesses this analog output subsystem through its parent object, typically through a method such as *dac() (see USB_DA12_8E_Family::dac())*.

23.48.2 Constructor & Destructor Documentation

DA12_AnalogOutputSubsystem (USBDeviceBase & *parent*) `[protected]`

∼DA12_AnalogOutputSubsystem ( ) `[protected], [virtual]`

23.48.3 Member Function Documentation

std::string getRangeText (int *range*) `[static]`

Gets the textual string for the specified range.

Parameters

range	the range for which to obtain the textual string.

Returns

The textual string for the specified range.

See Also

setRange(int range)

Exceptions

IllegalArgumentException	

int getRange (int *channel*) const

Gets the current voltage range of a D/A channel.

Parameters

channel	the channel for which to obtain the current range.

Returns

Current voltage range.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

IntArray getRange (int *startChannel,* int *numChannels*) const

Gets the current voltage range of multiple D/A channels.

Parameters

startChannel	the first channel for which to obtain the current range.
numChannels	the number of channels for which to obtain the current range.

Returns

Array containing the current range for each of the specified channels.

See Also

setRange(int startChannel, const IntArray &range)

Exceptions

IllegalArgumentException	

DA12_AnalogOutputSubsystem & setRange (int *channel,* int *range*)

Sets the voltage range of a D/A channel.

The ranges in this device are selected by means of hardware jumpers, so these range settings here do not affect the hardware. However, they are used to perform conversions between volts and counts. Moreover, the range setting is per D/A channel, so care must be taken when setting the ranges to ensure that the software setting matches the hardware jumper configuration, otherwise the voltage-count conversions will be incorrect.

Parameters

channel	the channel for which to set the range.
range	the voltage range to select. May be one of: *DA12_AnalogOutputSubsystem::RANGE_0_2_5V* *DA12_AnalogOutputSubsystem::RANGE_2_5V* *DA12_AnalogOutputSubsystem::RANGE_0_5V* *DA12_AnalogOutputSubsystem::RANGE_5V* *DA12_AnalogOutputSubsystem::RANGE_0_10V* *DA12_AnalogOutputSubsystem::RANGE_10V*

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

DA12_AnalogOutputSubsystem & setRange (int *startChannel,* const IntArray & *range*)

Sets the current voltage range of multiple D/A channels.

Parameters

startChannel	the first channel for which to set the range.
range	an array of voltage ranges to select, one per channel. The length of this array implicitly specifies the number of channels to configure.

Returns

This subsystem, useful for chaining together multiple operations.

See Also

setRange(int channel, int range)

Exceptions

IllegalArgumentException	

DA12_AnalogOutputSubsystem & setRange (int *range*)

Sets the current voltage range of all D/A channels to the same value.

Parameters

range	the voltage range to select.

Returns

This subsystem, useful for chaining together multiple operations.

See Also

setRange(int channel, int range)

DA12_AnalogOutputSubsystem & writeVolts (int *channel,* double *volts*)

Writes a voltage value to a D/A channel.

Parameters

channel	the channel to write to.
volts	the voltage value to output.

Returns

This subsystem, useful for chaining together multiple operations.

DA12_AnalogOutputSubsystem & writeVolts (const OutputVoltagePointArray & *points*)

Writes a block of voltage values to one or more D/A channels.

Parameters

points	an array of OutputVoltagePoint points representing channel-voltage pairs.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	

double countsToVolts (int *channel,* unsigned short *counts*) const

Converts a single D/A count value to volts, based on the current range setting.

Parameters

channel	the channel whose current range will be used to perform the conversion.
counts	the count value to convert to volts.

Returns

A voltage value calculated using the current D/A range. The voltage value returned is constrained to the current minimum-maximum voltage range of the D/A. *(see setRange(int channel, int range))*.

Exceptions

IllegalArgumentException	

unsigned short voltsToCounts (int *channel*, double *volts*) const

Converts a single voltage value to D/A counts, based on the current range setting.

Parameters

channel	the channel whose current range will be used to perform the conversion.
volts	the voltage value to convert to counts.

Returns

A count value calculated using the current D/A range. The count value returned is constrained to the current minimum-maximum count range of the D/A. *(see setRange(int channel, int range))*.

Exceptions

IllegalArgumentException	

23.48.4 Friends And Related Function Documentation

friend class USB_DA12_8A_Family `[friend]`

friend class USB_DA12_8E_Family `[friend]`

23.48.5 Field Documentation

const int RANGE_0_2_5V = 0 `[static]`

Unipolar, 0-2.5 volt range *(see setRange(int channel, int range))*.

const int RANGE_2_5V = 1 `[static]`

Bipolar, -2.5 to +2.5 volt range *(see setRange(int channel, int range))*.

const int RANGE_0_5V = 2 `[static]`

Unipolar, 0-5 volt range *(see setRange(int channel, int range))*.

const int RANGE_5V = 3 `[static]`

Bipolar, -5 to +5 volt range *(see setRange(int channel, int range))*.

const int RANGE_0_10V = 4 `[static]`

Unipolar, 0-10 volt range *(see setRange(int channel, int range))*.

const int RANGE_10V = 5 `[static]`

Bipolar, -10 to +10 volt range *(see setRange(int channel, int range))*.

const int MIN_COUNTS = 0 `[static]`

Minimum number of counts D/A can output.

const int MAX_COUNTS = 0xfff `[static]`

Maximum number of counts D/A can output.

const char RANGE_TEXT `[static]`, `[protected]`

Initial value:

```
= {
          "0-2.5V"
        , "+/-2.5V"
        , "0-5V"
        , "+/-5V"
        , "0-10V"
        , "+/-10V"
}
```

DA12_OutputRange∗ outputRange `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/DA12_AnalogOutputSubsystem.hpp
- deprecated/classlib/DA12_AnalogOutputSubsystem.cpp

23.49 DA12_OutputRange Class Reference

`#include <DA12_OutputRange.hpp>`

Public Member Functions

- virtual AnalogIORange & setRange (int range)

  Sets the range ID.

Protected Member Functions

- DA12_OutputRange ()
- DA12_OutputRange (int minCounts, int maxCounts)
- virtual ∼DA12_OutputRange ()

Friends

- class DA12_AnalogOutputSubsystem

Additional Inherited Members

23.49.1 Constructor & Destructor Documentation

DA12_OutputRange ( ) `[protected]`

DA12_OutputRange (int *minCounts*, int *maxCounts*) `[protected]`

∼DA12_OutputRange ( ) `[protected]`, `[virtual]`

23.49.2 Member Function Documentation

AnalogIORange & setRange (int *range*) `[virtual]`

Sets the range ID.

Parameters

range	the new range ID (defined by class that owns this instance).

Returns

This subsystem, useful for chaining together multiple operations.

Reimplemented from AnalogIORange.

23.49.3 Friends And Related Function Documentation

friend class DA12_AnalogOutputSubsystem `[friend]`

The documentation for this class was generated from the following files:

- deprecated/classlib/DA12_OutputRange.hpp
- deprecated/classlib/DA12_OutputRange.cpp

23.50 DeviceInfo Struct Reference

Data Fields

- unsigned char outputMask [MASK_BYTES]
- unsigned char readBuffer [MAX_DIO_BYTES]
- unsigned char writeBuffer [MAX_DIO_BYTES]
- char name [MAX_NAME_SIZE+2]
- unsigned long productID
- unsigned long nameSize
- unsigned long numDIOBytes
- unsigned long numCounters
- uint64_t serialNumber
- int index

23.50.1 Field Documentation

unsigned char outputMask

unsigned char readBuffer

unsigned char writeBuffer

char name

unsigned long productID

unsigned long nameSize

unsigned long numDIOBytes

unsigned long numCounters

uint64_t serialNumber

int index

The documentation for this struct was generated from the following files:

- samples/USB-DIO-96/read_and_write_sample.c
- samples/USB-DIO-96/write_sample.c

23.51 DeviceProperties Struct Reference

Allows us to keep track of streaming (bulk) acquires without making the user keep track of the memory management.

```
#include <AIOTypes.h>
```

Data Fields

- char * Name

  null-terminated device name or 0
- uint64_t SerialNumber

  64-bit serial number or 0
- unsigned ProductID

  16-bit product ID

- unsigned DIOPorts

  number of digital I/O ports (bytes)
- unsigned Counters

  number of 8254 counter blocks
- unsigned Tristates

  number of tristates
- long RootClock

  base clock frequency
- unsigned DACChannels

  number of D/A channels
- unsigned ADCChannels

  number of A/D channels
- unsigned ADCMUXChannels

  number of MUXed A/D channels
- unsigned ADCChannelsPerGroup

  number of A/D channels in each config.

23.51.1 Detailed Description

Allows us to keep track of streaming (bulk) acquires without making the user keep track of the memory management.

23.51.2 Field Documentation

char∗ Name

null-terminated device name or 0

uint64_t SerialNumber

64-bit serial number or 0

unsigned ProductID

16-bit product ID

unsigned DIOPorts

number of digital I/O ports (bytes)

unsigned Counters

number of 8254 counter blocks

unsigned Tristates

number of tristates

long RootClock

base clock frequency

unsigned DACChannels

number of D/A channels

unsigned ADCChannels

number of A/D channels

unsigned ADCMUXChannels

number of MUXed A/D channels

unsigned ADCChannelsPerGroup

number of A/D channels in each config.

group

The documentation for this struct was generated from the following file:

- lib/AIOTypes.h

23.52 DeviceSubsystem Class Reference

Class DeviceSubsystem is the abstract super class for all device subsystems.

`#include <DeviceSubsystem.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)=0
- USBDeviceBase & getParent ()

  Gets the parent device that this subsystem is part of.

Protected Member Functions

- DeviceSubsystem (USBDeviceBase &parent)
- virtual ∼DeviceSubsystem ()
- int getDeviceIndex () const

Protected Attributes

- USBDeviceBase ∗ parent

23.52.1 Detailed Description

Class DeviceSubsystem is the abstract super class for all device subsystems.

23.52.2 Constructor & Destructor Documentation

DeviceSubsystem (USBDeviceBase & *parent*) `[protected]`

∼DeviceSubsystem ( ) `[protected], [virtual]`

23.52.3 Member Function Documentation

int getDeviceIndex ( ) const `[protected]`

virtual std::ostream& print (std::ostream & *out*) `[pure virtual]`

Implemented in AnalogInputSubsystem, DigitalIOSubsystem, CounterSubsystem, DIOStreamSubsystem, and Analog-OutputSubsystem.

USBDeviceBase& getParent ( ) `[inline]`

Gets the parent device that this subsystem is part of.

Returns

The parent device that this subsystem is part of.

23.52.4 Field Documentation

USBDeviceBase∗ parent `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/DeviceSubsystem.hpp
- deprecated/classlib/DeviceSubsystem.cpp

23.53 DigitalIOSubsystem Class Reference

Class DigitalIOSubsystem represents the digital I/O subsystem of a device.

`#include <DigitalIOSubsystem.hpp>`

Public Member Functions

- UCharArray & bitsToBytes (UCharArray &dest, int bit, const BoolArray &src)
- BoolArray & bytesToBits (BoolArray &dest, const UCharArray &src, int bit)
- DigitalIOSubsystem (USBDeviceBase &parent)
- virtual ∼DigitalIOSubsystem ()
- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this subsystem.
- int getNumPorts () const

  Gets the number of digital I/O ports (bytes).
- int getNumChannels () const

  Gets the number of digital I/O channels (bits).
- int getNumTristateGroups () const

  Gets the number of tristate groups (bytes).
- int getNumTristateChannels () const

  Gets the number of tristate channels (bits).
- DigitalIOSubsystem & configure (bool tristate, const BoolArray &outputs, const BoolArray &values)

  Configures the digital I/O ports.
- DigitalIOSubsystem & configure (const BoolArray &tristates, const BoolArray &outputs, const BoolArray &values)

  Configures the digital I/O ports.
- DigitalIOSubsystem & getConfiguration (BoolArray ∗tristates, BoolArray ∗outputs)

  Gets the current configuration of the digital I/O ports.
- bool read (int channel)

  Reads a single digital input channel.
- BoolArray read (int startChannel, int numChannels)

  Reads multiple digital input channels.
- DigitalIOSubsystem & write (int channel, bool value)

  Writes a single digital output channel.
- DigitalIOSubsystem & write (int startChannel, const BoolArray &values)

  Writes multiple digital output channels.

Protected Attributes

- int numPorts
- int numChannels
- int numTristateGroups
- int numTristateChannels
- UCharArray writeValues

Friends

- class USB_AI16_Family
- class USB_AO16_Family
- class USB_DIO_Family
- class USB_DIO_16_Family
- class USB_DIO_32_Family

Additional Inherited Members

23.53.1 Detailed Description

Class DigitalIOSubsystem represents the digital I/O subsystem of a device.

One accesses this analog output subsystem through its parent object, typically through a method such as *dio() (see USB_AI16_Family::dio())*.

23.53.2 Constructor & Destructor Documentation

DigitalIOSubsystem (USBDeviceBase & *parent*)

~DigitalIOSubsystem ( ) `[virtual]`

23.53.3 Member Function Documentation

UCharArray & bitsToBytes (UCharArray & *dest,* int *bit,* const BoolArray & *src*)

BoolArray & bytesToBits (BoolArray & *dest,* const UCharArray & *src,* int *bit*)

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getNumPorts ( ) const `[inline]`

Gets the number of digital I/O ports (bytes).

Returns

Number of ports, numbered 0 to n-1.

int getNumChannels ( ) const `[inline]`

Gets the number of digital I/O channels (bits).

The number of "channels" is simply equal to the number of ports times the number of channels per port, which is eight.

Returns

Number of channels, numbered 0 to n-1.

int getNumTristateGroups ( ) const `[inline]`

Gets the number of tristate groups (bytes).

Returns

Number of tristate groups, numbered 0 to n-1.

int getNumTristateChannels ( ) const `[inline]`

Gets the number of tristate channels (bits).

The number of "channels" is simply equal to the number of tristate groups times the number of channels per group, which is eight.

Returns

Number of tristate channels, numbered 0 to n-1.

DigitalIOSubsystem & configure (bool *tristate*, const BoolArray & *outputs*, const BoolArray & *values*)

Configures the digital I/O ports.

Parameters

tristate	*true* causes all bits on the device to enter tristate (high-impedance) mode; *false* removes tristate mode. All devices with this feature power up in tristate mode, and tristate mode is changed after the remainder of the configuration has occurred.
outputs	an array of boolean values, one per digital I/O *port*. Each *true* value in the array configures the entire corresponding I/O port as an output port; each *false* value configures the entire corresponding I/O port as an input port.
values	an array of boolean values, one per digital I/O *bit*, starting with bit 0 of the device (that is, the least significant bit on the lowest numbered port). Each *true* value in the array sets the corresponding output bit to a "1"; each *false* value sets the corresponding output bit to a "0." The values are written to the digital output ports before the ports are taken out of tristate mode.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

DigitalIOSubsystem & configure (const BoolArray & *tristates*, const BoolArray & *outputs*, const BoolArray & *values*)

Configures the digital I/O ports.

If the device does not support the per-port tristate feature, then configure(bool tristate, const BoolArray &outputs, const BoolArray &values) should be used instead, otherwise an exception will be thrown.

Parameters

tristates	an array of boolean values, one per tristate group. Each *true* value in the array puts the entire corresponding I/O port into tristate (high-impedance) mode; each *false* value takes the entire corresponding I/O port out of tristate mode. All devices with this feature power up in tristate mode, and tristate mode is changed after the remainder of the configuration has occurred.
outputs	an array of boolean values, one per digital I/O *port*. Each *true* value in the array configures the entire corresponding I/O port as an output port; each *false* value configures the entire corresponding I/O port as an input port.
values	an array of boolean values, one per digital I/O *bit*, starting with bit 0 of the device (that is, the least significant bit on the lowest numbered port). Each *true* value in the array sets the corresponding output bit to a "1"; each *false* value sets the corresponding output bit to a "0." The values are written to the digital output ports before the ports are taken out of tristate mode.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

DigitalIOSubsystem & getConfiguration (BoolArray ∗ *tristates*, BoolArray ∗ *outputs*)

Gets the current configuration of the digital I/O ports.

If the device does not support the per-port tristate feature, then this method should not be used, otherwise an exception will be thrown.

Parameters

tristates	an array of boolean values, one per tristate group, which will receive the current tristate mode of each tristate group. Each *true* value returned in the array indicates that the entire corresponding I/O port is in tristate (high-impedance) mode; each *false* value indicates that the entire corresponding I/O port is not in tristate mode. If this parameter is *null*, then the tristate configuration is not returned.
outputs	an array of boolean values, one per digital I/O *port*, which will receive the current output mode of each I/O port. Each *true* value returned in the array indicates that the entire corresponding I/O port is configured as an output port; each *false* value indicates that the entire corresponding I/O port is configured as an input port. If this parameter is *null*, then the output configuration is not returned.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

bool read (int *channel*)

Reads a single digital input channel.

Parameters

channel	the channel to read.

Returns

True indicates that the bit is set ("1"); *false* indicates that the bit is clear ("0").

Exceptions

IllegalArgumentException	
OperationFailedException	

BoolArray read (int *startChannel*, int *numChannels*)

Reads multiple digital input channels.

Parameters

startChannel	the first channel to read.
numChannels	the number of channels to read.

Returns

An array containing the values read from the specified channels. For each channel, *true* indicates that the bit is set ("1"); *false* indicates that the bit is clear ("0").

Exceptions

IllegalArgumentException	
OperationFailedException	

DigitalIOSubsystem & write (int *channel*, bool *value*)

Writes a single digital output channel.

Parameters

channel	the channel to write.
value	the value to write to the specified channel. *True* sets the output bit to a "1" and *false* clears the output bit to a "0".

Returns

This subsystem, useful for chaining together multiple operations.

DigitalIOSubsystem & write (int *startChannel,* const BoolArray & *values*)

Writes multiple digital output channels.

Parameters

startChannel	the first channel to write.
values	an array containing the values to write to the specified channels. For each channel, *true* sets the output bit to a "1" and *false* clears the output bit to a "0".

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

23.53.4 Friends And Related Function Documentation

friend class USB_AI16_Family [friend]

friend class USB_AO16_Family [friend]

friend class USB_DIO_Family [friend]

friend class USB_DIO_16_Family [friend]

friend class USB_DIO_32_Family [friend]

23.53.5 Field Documentation

int numPorts [protected]

int numChannels [protected]

int numTristateGroups [protected]

int numTristateChannels [protected]

UCharArray writeValues [protected]

The documentation for this class was generated from the following files:

- deprecated/classlib/DigitalIOSubsystem.hpp
- deprecated/classlib/DigitalIOSubsystem.cpp

23.54 DIOBuf Struct Reference

DIOBuf: A Smart structure for maintaining bit vectors and for providing human-readable functionality to make it easy to operate on said bit vectors.

```
#include <DIOBuf.h>
```

Data Fields

- unsigned size

  Size of the buffer.
- unsigned char ∗ buffer

  Raw buffer data.
- char ∗ strbuf

  String representation in terms of 1's and 0's.
- int strbuf_size

  Strlen of the 1's and 0's version including some padding room.

23.54.1 Detailed Description

DIOBuf: A Smart structure for maintaining bit vectors and for providing human-readable functionality to make it easy to operate on said bit vectors.

The functionality provided by this structure makes it easy for a user to work with a binary string data type convert it between raw bytes and hexidecimal representations as well as use it for generating digital intput, output and tristate bits with the corresponding `ACCES I/O Products` USB Digital input and output boards.

There are methods to work with DIOBuf that will convert this structure to a character string of 1's and 0's, to a hexadecimal representation and to raw bytes that can be used in the transmission across a number of media. This later functionality would be useful in case you are working with a network server that would need to write an incoming byte stream to a digital buffer.

Todo Provide Binary operators such as AND, OR, And Not between two different DIOBuf's

23.54.2 Field Documentation

unsigned size

Size of the buffer.

unsigned char∗ buffer

Raw buffer data.

char∗ strbuf

String representation in terms of 1's and 0's.

int strbuf_size

Strlen of the 1's and 0's version including some padding room.

The documentation for this struct was generated from the following file:

- lib/DIOBuf.h

23.55 DIOStreamSubsystem Class Reference

Class DIOStreamSubsystem represents the digital I/O streaming subsystem of a device.

`#include <DIOStreamSubsystem.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this subsystem.
- int getStreamingBlockSize () const

  Gets the current streaming block size.
- DIOStreamSubsystem & setStreamingBlockSize (int blockSize)

  Sets the streaming block size.
- double getClock () const

  Gets the current internal read/write clock speed of a digital I/O stream.
- double setClock (bool directionRead, double clockHz)

  Sets the internal read/write clock speed of a digital I/O stream (see getClock()).
- DIOStreamSubsystem & stopClock ()

  Stops the internal read/write clocks of a digital I/O stream.
- DIOStreamSubsystem & open (bool directionRead)

  Opens a digital I/O stream.
- DIOStreamSubsystem & close ()

  Closes a digital I/O stream opened by a call to open(bool directionRead).
- UShortArray read (int numSamples)

  Reads a frame from a digital I/O stream opened by a call to open(true).
- int write (const UShortArray &values)

Writes a frame to a digital I/O stream opened by a call to open(false).

- DIOStreamSubsystem & clearFIFO (FIFO_Method method)

  Clears the streaming FIFO, using one of several different methods.

Protected Member Functions

- DIOStreamSubsystem (USBDeviceBase &parent)
- virtual ∼DIOStreamSubsystem ()

Protected Attributes

- double clockHz

Friends

- class USB_DIO_16_Family

23.55.1 Detailed Description

Class DIOStreamSubsystem represents the digital I/O streaming subsystem of a device.

One accesses this counter/timer subsystem through its parent object, typically through a method such as *diostream() (see USB_DIO_16_Family::diostream())*.

23.55.2 Constructor & Destructor Documentation

DIOStreamSubsystem (USBDeviceBase & *parent*) `[protected]`

∼DIOStreamSubsystem ( ) `[protected], [virtual]`

23.55.3 Member Function Documentation

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this subsystem.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Implements DeviceSubsystem.

int getStreamingBlockSize ( ) const `[inline]`

Gets the current streaming block size.

Returns

The current streaming block size. The value returned may not be the same as the value passed to *setStreamingBlockSize(int blockSize)* because that value is rounded up to a whole multiple of 256.

Exceptions

OperationFailedException	

DIOStreamSubsystem& setStreamingBlockSize (int *blockSize*) `[inline]`

Sets the streaming block size.

Parameters

blockSize	the streaming block size you wish to set. This will be rounded up to the next multiple of 256.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

double getClock ( ) const `[inline]`

Gets the current internal read/write clock speed of a digital I/O stream.

Returns

The actual frequency that will be generated, based on the last call to *setClock(bool directionRead, double clockHz)*.

double setClock (bool *directionRead*, double *clockHz*)

Sets the internal read/write clock speed of a digital I/O stream *(see getClock())*.

Only one clock - the read or write clock

- may be active at a time, so this method ***automatically turns off*** the clock not being set by this method. Therefore, do not call this method to explicitly turn off one of the clocks because it will turn off both of them. Also, when streaming between two devices, only one should have an active internal clock; the other should have its clocks turned off *(see stopClock())*.

  Parameters

directionRead	*true* sets read clock; *false* sets write clock.
clockHz	the frequency at which to stream the samples (in Hertz).

  Returns

  The actual frequency that will be generated, limited by the device's capabilities.

  Exceptions

OperationFailedException	

DIOStreamSubsystem& stopClock ( ) `[inline]`

Stops the internal read/write clocks of a digital I/O stream.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

DIOStreamSubsystem & open (bool *directionRead*)

Opens a digital I/O stream.

When you are done using the stream, you must close it by calling *close()*.

Parameters

directionRead	*true* open the stream for reading; *false* open the stream for writing.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

DIOStreamSubsystem & close ( )

Closes a digital I/O stream opened by a call to *open(bool directionRead)*.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

UShortArray read (int *numSamples*)

Reads a frame from a digital I/O stream opened by a call to *open(true)*.

You cannot read from, and write to a stream. A stream may be read-only or write-only.

Parameters

numSamples	the number of samples to read.

Returns

An array containing the samples read. The array may be smaller than the number of samples requested if fewer samples were received than were requested.

Exceptions

IllegalArgumentException	
OperationFailedException	

int write (const UShortArray & *values*)

Writes a frame to a digital I/O stream opened by a call to *open(false)*.

You cannot read from, and write to a stream. A stream may be read-only or write-only.

Parameters

values	an array containing the samples to write.

Returns

The number of samples actually written.

Exceptions

IllegalArgumentException	
OperationFailedException	

DIOStreamSubsystem& clearFIFO (FIFO_Method *method*) `[inline]`

Clears the streaming FIFO, using one of several different methods.

Parameters

method	the method to use when clearing the FIFO. May be one of: *USBDeviceBase::CLEAR_FIFO_METHOD_IMMEDIATE USBDeviceBase::CLEAR_FIFO_METHOD_AUTO USBDeviceBase::CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT USBDeviceBase::CLEAR_FIFO_METHOD_WAIT*

Returns

This subsystem, useful for chaining together multiple operations.

23.55.4 Friends And Related Function Documentation

friend class USB_DIO_16_Family `[friend]`

23.55.5 Field Documentation

double clockHz `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/DIOStreamSubsystem.hpp
- deprecated/classlib/DIOStreamSubsystem.cpp

23.56 DoubleArray Class Reference

```
#include <USBDeviceBase.hpp>
```

Public Member Functions

- DoubleArray (int size=0)

23.56.1 Constructor & Destructor Documentation

DoubleArray (int *size* = 0) `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceBase.hpp

23.57 Error Class Reference

```
#include <TestCaseSetup.h>
```

Public Member Functions

- Error ()
- Error (const char ∗entry)
- virtual const char ∗ what () const throw ()

Private Attributes

- const char ∗ message

23.57.1 Constructor & Destructor Documentation

Error ( ) `[explicit]`

Error (const char ∗ *entry*) `[inline]`

23.57.2 Member Function Documentation

virtual const char∗ what ( ) const throw) `[inline], [virtual]`

23.57.3 Field Documentation

const char∗ message `[private]`

The documentation for this class was generated from the following file:

- samples/TestLib/TestCaseSetup.h

23.58 IllegalArgumentException Class Reference

Class IllegalArgumentException is thrown whenever an invalid argument is passed to a method.

`#include <USBDeviceManager.hpp>`

Public Member Functions

- IllegalArgumentException (const std::string &message)
  Constructs the exception from a simple string message.

23.58.1 Detailed Description

Class IllegalArgumentException is thrown whenever an invalid argument is passed to a method.

23.58.2 Constructor & Destructor Documentation

IllegalArgumentException (const std::string & *message*) `[inline]`

Constructs the exception from a simple string message.

Parameters

message	The text of the message.

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceManager.hpp

23.59 IntArray Class Reference

`#include <USBDeviceBase.hpp>`

Public Member Functions

- IntArray (int size=0)

23.59.1 Constructor & Destructor Documentation

IntArray (int *size* = 0 **)** `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceBase.hpp

23.60 lookup Struct Reference

`#include <AIOTypes.h>`

Data Fields

- int value
- char ∗ str
- char ∗ strvalue

23.60.1 Field Documentation

int value

char∗ str

char∗ strvalue

The documentation for this struct was generated from the following file:

- lib/AIOTypes.h

23.61 mux_settings Struct Reference

```
#include <ADCConfigBlock.h>
```

Data Fields

- unsigned long ADCChannelsPerGroup
- unsigned long ADCMUXChannels
- AIOUSB_BOOL defined

23.61.1 Field Documentation

unsigned long ADCChannelsPerGroup

unsigned long ADCMUXChannels

AIOUSB_BOOL defined

The documentation for this struct was generated from the following file:

- lib/ADCConfigBlock.h

23.62 new_aio_fifo Struct Reference

```
#include <AIOFifo.h>
```

Data Fields

- AIO_FIFO_INTERFACE
- LOCKING_MECHANISM
- AIORET_TYPE(∗ Push)(struct new_aio_fifo ∗fifo, TYPE a)
- AIORET_TYPE(∗ PushN)(struct new_aio_fifo ∗fifo, INPUT_TYPE ∗a, unsigned N)
- AIOEither(∗ Pop)(struct new_aio_fifo ∗fifo)
- AIORET_TYPE(∗ PopN)(struct new_aio_fifo ∗fifo, INPUT_TYPE ∗a, unsigned N)

23.62.1 Field Documentation

AIO_FIFO_INTERFACE

LOCKING_MECHANISM

AIORET_TYPE(∗ Push)(struct new_aio_fifo ∗fifo, TYPE a)

AIORET_TYPE(∗ PushN)(struct new_aio_fifo ∗fifo, INPUT_TYPE ∗a, unsigned N)

AIOEither(∗ Pop)(struct new_aio_fifo ∗fifo)

AIORET_TYPE(∗ PopN)(struct new_aio_fifo ∗fifo, INPUT_TYPE ∗a, unsigned N)

The documentation for this struct was generated from the following file:

- lib/AIOFifo.h

23.63 OperationFailedException Class Reference

Class OperationFailedException is thrown whenever an operation attempted on a device fails.

```
#include <USBDeviceManager.hpp>
```

Public Member Functions

- OperationFailedException (int result)

  Constructs the exception from an AIOUSB module error code.
- OperationFailedException (const std::string &message)

  Constructs the exception from a simple string message.

23.63.1 Detailed Description

Class OperationFailedException is thrown whenever an operation attempted on a device fails.

The message is either generated by this Java class library or consists of the string representation of an error code returned by the AIOUSB module.

23.63.2 Constructor & Destructor Documentation

OperationFailedException (int *result*) `[inline]`

Constructs the exception from an AIOUSB module error code.

Parameters

result	AIOUSB module result code.

OperationFailedException (const std::string & *message*) `[inline]`

Constructs the exception from a simple string message.

Parameters

message	The text of the message.

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceManager.hpp

23.64 options Struct Reference

Data Fields

- int maxcount
- int use_maxcount

23.64.1 Field Documentation

int maxcount

int use_maxcount

The documentation for this struct was generated from the following file:

- samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp

23.65 opts Struct Reference

```
#include <aiocommon.h>
```

Data Fields

- int64_t num_scans
- int64_t default_num_scans
- int num_channels
- int default_num_channels
- int num_oversamples
- int default_num_oversamples

- int gain_code
- int clock_rate
- int default_clock_rate
- char ∗ outfile
- int reset
- int debug_level
- int number_ranges
- int verbose
- int start_channel
- int default_start_channel
- int end_channel
- int default_end_channel
- int index
- int block_size
- int with_timing
- int slow_acquire
- int buffer_size
- int rate_limit
- int physical
- int counts
- int calibration
- int repeat
- char ∗ aiobuf_json
- char ∗ default_aiobuf_json
- char ∗ adcconfig_json
- struct channel_range ∗∗ ranges
- int num_scans
- int clock_speed
- int cal_channel
- int max_channels
- int clock_scale
- int calibration_enabled

23.65.1 Field Documentation

int64_t num_scans

int64_t default_num_scans

int num_channels

int default_num_channels

int num_oversamples

int default_num_oversamples

int gain_code

int clock_rate

int default_clock_rate

char∗ outfile

int reset

int debug_level

int number_ranges

int verbose

int start_channel

int default_start_channel

int end_channel

int default_end_channel

int index

int block_size

int with_timing

int slow_acquire

int buffer_size

int rate_limit

int physical

int counts

int calibration

int repeat

char∗ aiobuf_json

char∗ default_aiobuf_json

char∗ adcconfig_json

struct channel_range∗∗ ranges

int num_scans

int clock_speed

int cal_channel

int max_channels

int clock_scale

int calibration_enabled

The documentation for this struct was generated from the following files:

- samples/TestLib/aiocommon.h
- samples/USB-AI16-16/bulk_acquire_sample.c

23.66 OutputVoltagePoint Class Reference

Class OutputVoltagePoint represents a single analog output data point, consisting of a D/A channel number and a voltage to output to that channel.

`#include <OutputVoltagePoint.hpp>`

Public Member Functions

- OutputVoltagePoint ()

  Default constructor for analog output data point.
- OutputVoltagePoint (int channel, double volts)

  Constructor for analog output data point.

Data Fields

- int channel

  Channel number to output voltage to.
- double volts

  Voltage to output.

23.66.1 Detailed Description

Class OutputVoltagePoint represents a single analog output data point, consisting of a D/A channel number and a voltage to output to that channel.

It is used by methods *AO16_AnalogOutputSubsystem::writeVolts(const OutputVoltagePointArray &points)* and *DA12-_AnalogOutputSubsystem::writeVolts(const OutputVoltagePointArray &points)* to output a series of voltages to multiple D/A channels.

23.66.2 Constructor & Destructor Documentation

OutputVoltagePoint ( ) `[inline]`

Default constructor for analog output data point.

OutputVoltagePoint (int *channel*, double *volts*) `[inline]`

Constructor for analog output data point.

Parameters

channel	the channel number to output voltage to.
volts	the voltage to output.

23.66.3 Field Documentation

int channel

Channel number to output voltage to.

double volts

Voltage to output.

The documentation for this class was generated from the following file:

- deprecated/classlib/OutputVoltagePoint.hpp

23.67 OutputVoltagePointArray Class Reference

`#include <OutputVoltagePoint.hpp>`

Public Member Functions

- OutputVoltagePointArray (int size=0)

23.67.1 Constructor & Destructor Documentation

OutputVoltagePointArray (int *size* = 0) `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/OutputVoltagePoint.hpp

23.68 ProductIDName Struct Reference

`#include <AIOUSB_Core.h>`

Data Fields

- unsigned int id
- char name [PROD_NAME_SIZE+2]

23.68.1 Field Documentation

unsigned int id

char name[PROD_NAME_SIZE+2]

The documentation for this struct was generated from the following file:

- lib/AIOUSB_Core.h

23.69 rangelookup Struct Reference

Data Fields

- int minvalue
- int maxvalue

23.69.1 Field Documentation

int minvalue

int maxvalue

The documentation for this struct was generated from the following file:

- lib/AIOContinuousBuffer.c

23.70 StringArray Class Reference

```
#include <USBDeviceBase.hpp>
```

Public Member Functions

- StringArray (int size=0)

23.70.1 Constructor & Destructor Documentation

StringArray (int *size* = 0) `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceBase.hpp

23.71 TestCaseSetup Class Reference

```
#include <TestCaseSetup.h>
```

Public Member Functions

- ∼TestCaseSetup ()
- TestCaseSetup ()
- TestCaseSetup (int deviceIndex, int numChannels)
- void findDevice ()
- void findDevice (AIOUSB_BOOL(∗find)(AIOUSBDevice ∗dev))
- void doSomething ()
- void setCurrentDeviceIndex (int DeviceIndex)
- void doBulkConfigBlock ()

  Uploads a bulk configuration block.
- void doPreSetup ()
- void doSetAutoCalibration ()

  Sets up the :auto: calibration mode.
- void doVerifyGroundCalibration ()

- void doVerifyReferenceCalibration ()

  Verify that A/D reference calibration is correct.
- void doDemonstrateReadVoltages ()

  DEMONSTRATE SCANNING CHANNELS AND MEASURING VOLTAGES.
- void doScanSingleChannel ()

  demonstrate reading a single channel in volts
- void doPreReadImmediateVoltages ()

  Performs an immediate read of voltages.
- void doCSVReadVoltages ()

  Simple version that just outputs data to csv file.
- void doCSVWithGetChannelV ()
- void doCleanupAfterBulk ()
- void doDACDirect (int channel, unsigned short voltage)
- void doDACDirectSetup ()
- void writeBuffer (char ∗filename)

  writes the bytes to a file in question.
- void setMaxCount (int val)
- void ThrowError (unsigned long, int)

  Exception handler.
- void doFastITScanSetup ()
- void doFastITScan (int numgets)
- unsigned short ∗ doGetBuffer ()
- void doTestSetAutoCalibration ()
- void doGenericVendorWrite (unsigned char Request, unsigned short Value, unsigned short Index, unsigned long ∗DataSize, void ∗pData)
- void doBulkAcquire ()

  Demonstrate bulk acquire.
- void doBulkAcquire (unsigned int block_size, unsigned int over_sample, unsigned int clock_speed)

  Demonstrate bulk acquire.
- void doDisplayBulkResults ()
- void resetCPU ()
- double ∗ getVolts ()
- unsigned short ∗ getCounts ()
- unsigned char ∗ getGainCodes ()

Static Public Member Functions

- static void THROW_IF_ERROR (int result, const char ∗format,...)
- static int envGetInteger (const char ∗env)
- static double envGetDouble (const char ∗env)

Data Fields

- unsigned long productID
- unsigned long nameSize
- unsigned long numDIOBytes
- unsigned long numCounters
- unsigned long DeviceIndex
- bool deviceFound
- const int CAL_CHANNEL
- const int MAX_CHANNELS
- int NUM_CHANNELS
- unsigned short ∗ counts
- double ∗ volts
- unsigned char ∗ gainCodes
- ADConfigBlock configBlock
- unsigned int number_oversamples
- unsigned int block_size
- unsigned int clock_speed
- int maxcounts
- AIOUSB_BOOL calibration_enabled

Private Member Functions

- void setupVoltageParameters (void)

  sets up the voltage parameters for runs
- unsigned long TEST_ADC_BulkPoll (unsigned long DeviceIndex, unsigned long ∗BytesLeft)

Private Attributes

- unsigned short ∗ dataBuf

23.71.1 Constructor & Destructor Documentation

∼TestCaseSetup ( )

TestCaseSetup ( )

TestCaseSetup (int *deviceIndex,* int *numChannels*)

23.71.2 Member Function Documentation

void findDevice (void)

void findDevice (AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *find*)

void doSomething ( )

void setCurrentDeviceIndex (int *DeviceIndex*)

void doBulkConfigBlock ( )

Uploads a bulk configuration block.

void doPreSetup ( )

void doSetAutoCalibration (void)

Sets up the :auto: calibration mode.

void doVerifyGroundCalibration (void)

void doVerifyReferenceCalibration (void)

Verify that A/D reference calibration is correct.

void doDemonstrateReadVoltages ( )

DEMONSTRATE SCANNING CHANNELS AND MEASURING VOLTAGES.

void doScanSingleChannel ( )

demonstrate reading a single channel in volts

void doPreReadImmediateVoltages ( )

Performs an immediate read of voltages.

void doCSVReadVoltages ( )

Simple version that just outputs data to csv file.

void doCSVWithGetChannelV ( )

void doCleanupAfterBulk ( )

void doDACDirect (int *channel*, unsigned short *voltage*)

void doDACDirectSetup ( )

void writeBuffer (char ∗ *filename*)

writes the bytes to a file in question.

Will be binary unless the user specifies CSV as an argument

void setMaxCount (int *val*)

void ThrowError (unsigned long *result*, int *linnum*)

Exception handler.

Parameters

result	
linnum	

void doFastITScanSetup ( )

void doFastITScan (int *numgets*)

unsigned short ∗ doGetBuffer ( )

void THROW_IF_ERROR (int *result*, const char ∗ *format*, ...) `[static]`

int envGetInteger (const char ∗ *env*) `[static]`

double envGetDouble (const char ∗ *env*) `[static]`

void doTestSetAutoCalibration (void)

void doGenericVendorWrite (unsigned char *Request*, unsigned short *Value*, unsigned short *Index*, unsigned long ∗ *DataSize*, void ∗ *pData*)

void doBulkAcquire (void)

Demonstrate bulk acquire.

void doBulkAcquire (unsigned int *blck_size*, unsigned int *ovr_sampl*, unsigned int *clk_speed*)

Demonstrate bulk acquire.

Parameters

blck_size	
ovr_sampl	
clk_speed	

void doDisplayBulkResults ( )

void resetCPU (void)

double ∗ getVolts ( )

unsigned short ∗ getCounts ( )

unsigned char ∗ getGainCodes ( )

void setupVoltageParameters (void) `[private]`

sets up the voltage parameters for runs

unsigned long TEST_ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long ∗ *BytesLeft*) `[private]`

23.71.3 Field Documentation

unsigned long productID

unsigned long nameSize

unsigned long numDIOBytes

unsigned long numCounters

unsigned long DeviceIndex

bool deviceFound

const int CAL_CHANNEL

const int MAX_CHANNELS

int NUM_CHANNELS

unsigned short∗ counts

double∗ volts

unsigned char∗ gainCodes

ADConfigBlock configBlock

unsigned int number_oversamples

unsigned int block_size

unsigned int clock_speed

int maxcounts

AIOUSB_BOOL calibration_enabled

unsigned short∗ dataBuf `[private]`

The documentation for this class was generated from the following files:

- samples/TestLib/TestCaseSetup.h
- samples/TestLib/TestCaseSetup.cpp

23.72 UCharArray Class Reference

```
#include <USBDeviceBase.hpp>
```

Public Member Functions

- UCharArray (int size=0)

23.72.1 Constructor & Destructor Documentation

UCharArray (int *size* = 0) `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceBase.hpp

23.73 USB_AI16_Family Class Reference

```
#include <USB_AI16_Family.hpp>
```

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

    Prints the properties of this device and all of its subsystems.
- AnalogInputSubsystem & adc ()

    Gets a reference to the analog input subsystem of this device.
- DigitalIOSubsystem & dio ()

    Gets a reference to the digital I/O subsystem of this device.
- CounterSubsystem & ctr ()

    Gets a reference to the counter/timer subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

    Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

    Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

    Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_AI16_Family (int productID, int deviceIndex)
- virtual ∼USB_AI16_Family ()

Protected Attributes

- AnalogInputSubsystem analogInputSubsystem
- DigitalIOSubsystem digitalIOSubsystem
- CounterSubsystem counterSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.73.1 Detailed Description

```
* Class USB_AI16_Family represents a USB-AI16-family device, which encompasses the following product IDs:
* USB_AI16_16A, USB_AI16_16E, USB_AI12_16A, USB_AI12_16, USB_AI12_16E, USB_AI16_64MA, USB_AI16_64ME, USB_AI12_64M
* USB_AI12_64M, USB_AI12_64ME, USB_AI16_32A, USB_AI16_32E, USB_AI12_32A, USB_AI12_32, USB_AI12_32E, USB_AI16_64A,
* USB_AI16_64E, USB_AI12_64A, USB_AI12_64, USB_AI12_64E, USB_AI16_96A, USB_AI16_96E, USB_AI12_96A, USB_AI12_96,
* USB_AI12_96E, USB_AI16_128A, USB_AI16_128E, USB_AI12_128A, USB_AI12_128, USB_AI12_128E.
* Instances of class <i>USB_AI16_Family</i> are automatically created by the USB device manager when they are
* detected on the bus. You should use one of the <i>USBDeviceManager</i> search methods, such as
* <i>USBDeviceManager::getDeviceByProductID( int productID ) const</i>,
* to obtain a reference to a <i>USB_AI16_Family</i> instance. You can then cast the <i>USBDeviceBase</i>
* reference obtained from one of those methods to a <i>USB_AI16_Family</i> and make use of this class' methods, l
* <pre>USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_AI12_32A, USB_AI12_32E );
*if( devices.size() > 0 )
*  USB_AI16_Family &device = *( USB_AI16_Family * ) devices.at( 0 );</pre>
*
```

23.73.2 Constructor & Destructor Documentation

USB_AI16_Family (int *productID*, int *deviceIndex*) `[protected]`

~USB_AI16_Family ( ) `[protected]`, `[virtual]`

23.73.3 Member Function Documentation

void initialize ( ) `[static]`, `[private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

AnalogInputSubsystem& adc ( ) `[inline]`

Gets a reference to the analog input subsystem of this device.

Returns

A reference to the analog input subsystem.

DigitalIOSubsystem& dio ( ) `[inline]`

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

CounterSubsystem& ctr ( ) `[inline]`

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

23.73.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

23.73.5 Field Documentation

IntArray supportedProductIDs `[static], [private]`

AnalogInputSubsystem analogInputSubsystem `[protected]`

DigitalIOSubsystem digitalIOSubsystem `[protected]`

CounterSubsystem counterSubsystem `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_AI16_Family.hpp
- deprecated/classlib/USB_AI16_Family.cpp

23.74 USB_AIO16_Family Class Reference

Class USB_AIO16_Family represents a USB-AI16-family device, which encompasses the following product IDs: US-B_AI16_16A, USB_AI16_16E, USB_AI12_16A, USB_AI12_16, USB_AI12_16E, USB_AI16_64MA, USB_AI16_64ME, USB_AI12_64MA, USB_AI12_64M, USB_AI12_64ME, USB_AI16_32A, USB_AI16_32E, USB_AI12_32A, USB_AI12-_32, USB_AI12_32E, USB_AI16_64A, USB_AI16_64E, USB_AI12_64A, USB_AI12_64, USB_AI12_64E, USB_AI16-_96A, USB_AI16_96E, USB_AI12_96A, USB_AI12_96, USB_AI12_96E, USB_AI16_128A, USB_AI16_128E, USB_A-I12_128A, USB_AI12_128, USB_AI12_128E.

```
#include <USB_AIO16_Family.hpp>
```

Public Member Functions

- USB_AIO16_Family (int productID, int deviceIndex)
- virtual ~USB_AIO16_Family ()
- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- AnalogInputSubsystem & adc ()

  Gets a reference to the analog input subsystem of this device.
- AnalogOutputSubsystem & dac ()
- DigitalIOSubsystem & dio ()

  Gets a reference to the digital I/O subsystem of this device.
- CounterSubsystem & ctr ()

  Gets a reference to the counter/timer subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

  Tells if a given product ID is supported by this USB device family.

Protected Attributes

- AnalogInputSubsystem analogInputSubsystem
- AnalogOutputSubsystem analogOutputSubsytem
- DigitalIOSubsystem digitalIOSubsystem
- CounterSubsystem counterSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.74.1 Detailed Description

Class USB_AIO16_Family represents a USB-AI16-family device, which encompasses the following product IDs: USB_AI16_16A, USB_AI16_16E, USB_AI12_16A, USB_AI12_16, USB_AI12_16E, USB_AI16_64MA, USB_AI16_64ME, USB_AI12_64MA, USB_AI12_64M, USB_AI12_64ME, USB_AI16_32A, USB_AI16_32E, USB_AI12_32A, USB_AI12_32, USB_AI12_32E, USB_AI16_64A, USB_AI16_64E, USB_AI12_64A, USB_AI12_64, USB_AI12_64E, USB_AI16_96A, USB_AI16_96E, USB_AI12_96A, USB_AI12_96, USB_AI12_96E, USB_AI16_128A, USB_AI16_128E, USB_AI12_128A, USB_AI12_128, USB_AI12_128E.

Instances of class *USB_AIO16_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_AIO16_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_AIO16_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_AI12_32A, USB_AI12_32E );
if( devices.size() > 0 )
  USB_AIO16_Family &device = *( USB_AIO16_Family * ) devices.at( 0 );
```

23.74.2 Constructor & Destructor Documentation

USB_AIO16_Family (int *productID*, int *deviceIndex*)

∼USB_AIO16_Family ( ) [virtual]

23.74.3 Member Function Documentation

void initialize ( ) [static], [private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) [static]

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) [virtual]

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

AnalogInputSubsystem& adc ( ) [inline]

Gets a reference to the analog input subsystem of this device.

Returns

A reference to the analog input subsystem.

AnalogOutputSubsystem& dac ( ) [inline]

DigitalIOSubsystem& dio ( ) [inline]

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

CounterSubsystem& ctr ( ) [inline]

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

23.74.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

23.74.5 Field Documentation

IntArray supportedProductIDs [static], [private]

AnalogInputSubsystem analogInputSubsystem [protected]

AnalogOutputSubsystem analogOutputSubsytem [protected]

DigitalIOSubsystem digitalIOSubsystem [protected]

CounterSubsystem counterSubsystem [protected]

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_AIO16_Family.hpp
- deprecated/classlib/USB_AIO16_Family.cpp

23.75 USB_AO16_Family Class Reference

Class USB_AO16_Family represents a USB-AO16-family device, which encompasses the following product IDs: USB_AO16_16A, USB_AO16_16, USB_AO16_12A, USB_AO16_12, USB_AO16_8A, USB_AO16_8, USB_AO16_4A, USB_AO16_4, USB_AO12_16A, USB_AO12_16, USB_AO12_12A, USB_AO12_12, USB_AO12_8A, USB_AO12_8, USB_AO12_4A, USB_AO12_4.

`#include <USB_AO16_Family.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- AO16_AnalogOutputSubsystem & dac ()

  Gets a reference to the analog output subsystem of this device.
- DigitalIOSubsystem & dio ()

  Gets a reference to the digital I/O subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_AO16_Family (int productID, int deviceIndex)
- virtual ∼USB_AO16_Family ()

Protected Attributes

- AO16_AnalogOutputSubsystem analogOutputSubsystem
- DigitalIOSubsystem digitalIOSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.75.1 Detailed Description

Class USB_AO16_Family represents a USB-AO16-family device, which encompasses the following product IDs: USB_AO16_16A, USB_AO16_16, USB_AO16_12A, USB_AO16_12, USB_AO16_8A, USB_AO16_8, USB_AO16_4A, USB_AO16_4, USB_AO12_16A, USB_AO12_16, USB_AO12_12A, USB_AO12_12, USB_AO12_8A, USB_AO12_8, USB_AO12_4A, USB_AO12_4.

Instances of class *USB_AO16_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_AO16_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_AO16_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_AO16_16A, USB_AO16_4 );
if( devices.size() > 0 )
  USB_AO16_Family &device = *( USB_AO16_Family * ) devices.at( 0 );
```

23.75.2 Constructor & Destructor Documentation

USB_AO16_Family (int *productID,* int *deviceIndex*) `[protected]`

~USB_AO16_Family ( ) `[protected], [virtual]`

23.75.3 Member Function Documentation

void initialize ( ) `[static], [private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

AO16_AnalogOutputSubsystem& dac ( ) `[inline]`

Gets a reference to the analog output subsystem of this device.

Returns

A reference to the analog output subsystem.

DigitalIOSubsystem& dio ( ) `[inline]`

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

23.75.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

23.75.5 Field Documentation

IntArray supportedProductIDs `[static], [private]`

AO16_AnalogOutputSubsystem analogOutputSubsystem `[protected]`

DigitalIOSubsystem digitalIOSubsystem `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_AO16_Family.hpp
- deprecated/classlib/USB_AO16_Family.cpp

23.76 USB_CTR_15_Family Class Reference

Class USB_CTR_15_Family represents a USB-CTR-15-family device, which encompasses the following product IDs: USB_CTR_15.

`#include <USB_CTR_15_Family.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- CounterSubsystem & ctr ()

  Gets a reference to the counter/timer subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_CTR_15_Family (int productID, int deviceIndex)
- virtual ~USB_CTR_15_Family ()

Protected Attributes

- CounterSubsystem counterSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.76.1 Detailed Description

Class USB_CTR_15_Family represents a USB-CTR-15-family device, which encompasses the following product IDs: USB_CTR_15.

Instances of class *USB_CTR_15_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_CTR_15_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_CTR_15_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_CTR_15 );
if( devices.size() > 0 )
  USB_CTR_15_Family &device = *( USB_CTR_15_Family * ) devices.at( 0 );
```

23.76.2 Constructor & Destructor Documentation

USB_CTR_15_Family (int *productID,* int *deviceIndex*) [protected]

~USB_CTR_15_Family ( ) [protected], [virtual]

23.76.3 Member Function Documentation

void initialize ( ) [static], [private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

CounterSubsystem& ctr ( ) `[inline]`

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

23.76.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

23.76.5 Field Documentation

IntArray supportedProductIDs `[static], [private]`

CounterSubsystem counterSubsystem `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_CTR_15_Family.hpp
- deprecated/classlib/USB_CTR_15_Family.cpp

23.77 USB_DA12_8A_Family Class Reference

Class USB_DA12_8A_Family represents a USB-DA12-8A-family device, which encompasses the following product IDs: USB_DA12_8A_REV_A, USB_DA12_8A.

`#include <USB_DA12_8A_Family.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)
  Prints the properties of this device and all of its subsystems.
- DA12_AnalogOutputSubsystem & dac ()
  Gets a reference to the analog output subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()
  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()
  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)
  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_DA12_8A_Family (int productID, int deviceIndex)
- virtual ~USB_DA12_8A_Family ()

Protected Attributes

- DA12_AnalogOutputSubsystem analogOutputSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.77.1 Detailed Description

Class USB_DA12_8A_Family represents a USB-DA12-8A-family device, which encompasses the following product IDs: USB_DA12_8A_REV_A, USB_DA12_8A.

Instances of class *USB_DA12_8A_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_DA12_8A_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_DA12_8A_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_DA12_8A_REV_A, USB_DA12_8A );
if( devices.size() > 0 )
  USB_DA12_8A_Family &device = *( USB_DA12_8A_Family * ) devices.at( 0 );
```

23.77.2 Constructor & Destructor Documentation

USB_DA12_8A_Family (int *productID,* int *deviceIndex*) [protected]

~USB_DA12_8A_Family ( ) [protected], [virtual]

23.77.3 Member Function Documentation

void initialize ( ) [static], [private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DA12_AnalogOutputSubsystem& dac ( ) `[inline]`

Gets a reference to the analog output subsystem of this device.

Returns

A reference to the analog output subsystem.

23.77.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

23.77.5 Field Documentation

IntArray supportedProductIDs `[static], [private]`

DA12_AnalogOutputSubsystem analogOutputSubsystem `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_DA12_8A_Family.hpp
- deprecated/classlib/USB_DA12_8A_Family.cpp

23.78 USB_DA12_8E_Family Class Reference

Class USB_DA12_8E_Family represents a USB-DA12-8E-family device, which encompasses the following product IDs: USB_DA12_8E.

```
#include <USB_DA12_8E_Family.hpp>
```

Public Member Functions

- virtual std::ostream & print (std::ostream &out)
  Prints the properties of this device and all of its subsystems.
- DA12_AnalogOutputSubsystem & dac ()
  Gets a reference to the analog output subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()
  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()
  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)
  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_DA12_8E_Family (int productID, int deviceIndex)
- virtual ~USB_DA12_8E_Family ()

Protected Attributes

- DA12_AnalogOutputSubsystem analogOutputSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.78.1 Detailed Description

Class USB_DA12_8E_Family represents a USB-DA12-8E-family device, which encompasses the following product IDs: USB_DA12_8E.

Instances of class *USB_DA12_8E_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_DA12_8E_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_DA12_8E_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_DA12_8E );
if( devices.size() > 0 )
  USB_DA12_8E_Family &device = *( USB_DA12_8E_Family * ) devices.at( 0 );
```

23.78.2 Constructor & Destructor Documentation

USB_DA12_8E_Family (int *productID,* int *deviceIndex*) [protected]

~USB_DA12_8E_Family ( ) [protected], [virtual]

23.78.3 Member Function Documentation

void initialize ( ) [static], [private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DA12_AnalogOutputSubsystem& dac ( ) `[inline]`

Gets a reference to the analog output subsystem of this device.

Returns

A reference to the analog output subsystem.

23.78.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

23.78.5 Field Documentation

IntArray supportedProductIDs `[static], [private]`

DA12_AnalogOutputSubsystem analogOutputSubsystem `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_DA12_8E_Family.hpp
- deprecated/classlib/USB_DA12_8E_Family.cpp

23.79 USB_DIO_16_Family Class Reference

Class USB_DIO_16_Family represents a USB-DIO-16-family device, which encompasses the following product IDs-: USB_DI16A_REV_A1, USB_DO16A_REV_A1, USB_DI16A_REV_A2, USB_DIO_16H, USB_DI16A, USB_DO16A, USB_DIO_16A.

```
#include <USB_DIO_16_Family.hpp>
```

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- DigitalIOSubsystem & dio ()

  Gets a reference to the digital I/O subsystem of this device.
- DIOStreamSubsystem & diostream ()

  Gets a reference to the digital I/O streaming subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_DIO_16_Family (int productID, int deviceIndex)
- virtual ~USB_DIO_16_Family ()

Protected Attributes

- DigitalIOSubsystem digitalIOSubsystem
- DIOStreamSubsystem dioStreamSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.79.1 Detailed Description

Class USB_DIO_16_Family represents a USB-DIO-16-family device, which encompasses the following product IDs-: USB_DI16A_REV_A1, USB_DO16A_REV_A1, USB_DI16A_REV_A2, USB_DIO_16H, USB_DI16A, USB_DO16A, USB_DIO_16A.

Instances of class *USB_DIO_16_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_DIO_16_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_DIO_16_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_DIO_16H, USB_DIO_16A );
if( devices.size() > 0 )
  USB_DIO_16_Family &device = *( USB_DIO_16_Family * ) devices.at( 0 );
```

23.79.2 Constructor & Destructor Documentation

USB_DIO_16_Family (int *productID*, int *deviceIndex*) [protected]

~USB_DIO_16_Family ( ) [protected], [virtual]

23.79.3 Member Function Documentation

void initialize ( ) [static], [private]

StringArray getSupportedProductNames ( ) [static]

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) [static]

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) [static]

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) [virtual]

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DigitalIOSubsystem& dio ( ) [inline]

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

DIOStreamSubsystem& diostream ( ) [inline]

Gets a reference to the digital I/O streaming subsystem of this device.

Returns

A reference to the digital I/O streaming subsystem.

23.79.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

23.79.5 Field Documentation

IntArray supportedProductIDs [static], [private]

DigitalIOSubsystem digitalIOSubsystem [protected]

DIOStreamSubsystem dioStreamSubsystem [protected]

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_DIO_16_Family.hpp
- deprecated/classlib/USB_DIO_16_Family.cpp

23.80 USB_DIO_32_Family Class Reference

Class USB_DIO_32_Family represents a USB-DIO-32-family device, which encompasses the following product IDs: USB_DIO_32.

`#include <USB_DIO_32_Family.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- DigitalIOSubsystem & dio ()

  Gets a reference to the digital I/O subsystem of this device.
- CounterSubsystem & ctr ()

  Gets a reference to the counter/timer subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_DIO_32_Family (int productID, int deviceIndex)
- virtual ∼USB_DIO_32_Family ()

Protected Attributes

- DigitalIOSubsystem digitalIOSubsystem
- CounterSubsystem counterSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.80.1 Detailed Description

Class USB_DIO_32_Family represents a USB-DIO-32-family device, which encompasses the following product IDs: USB_DIO_32.

Instances of class *USB_DIO_32_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_DIO_32_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_DIO_32_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_DIO_32 );
if( devices.size() > 0 )
  USB_DIO_32_Family &device = *( USB_DIO_32_Family * ) devices.at( 0 );
```

23.80.2 Constructor & Destructor Documentation

USB_DIO_32_Family (int *productID*, int *deviceIndex*) `[protected]`

∼USB_DIO_32_Family ( ) `[protected]`, `[virtual]`

23.80.3 Member Function Documentation

void initialize ( ) `[static]`, `[private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DigitalIOSubsystem& dio ( ) `[inline]`

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

CounterSubsystem& ctr ( ) [inline]

Gets a reference to the counter/timer subsystem of this device.

Returns

A reference to the counter/timer subsystem.

23.80.4 Friends And Related Function Documentation

friend class USBDeviceManager [friend]

23.80.5 Field Documentation

IntArray supportedProductIDs [static], [private]

DigitalIOSubsystem digitalIOSubsystem [protected]

CounterSubsystem counterSubsystem [protected]

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_DIO_32_Family.hpp
- deprecated/classlib/USB_DIO_32_Family.cpp

23.81 USB_DIO_Family Class Reference

Class USB_DIO_Family represents a USB-DIO-family device, which performs basic digital I/O and encompasses the following product IDs: USB_DIO_48, USB_DIO_96, USB_IIRO_16, USB_II_16, USB_RO_16, USB_IIRO_8, USB_II_8, USB_IIRO_4, USB_IDIO_16, USB_II_16_OLD, USB_IDO_16, USB_IDIO_8, USB_II_8_OLD, USB_IDIO_4, USB_IIRO4_2SM, USB_IIRO4_COM, USB_DIO16RO8, PICO_DIO16RO8.

`#include <USB_DIO_Family.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- DigitalIOSubsystem & dio ()

  Gets a reference to the digital I/O subsystem of this device.

Static Public Member Functions

- static StringArray getSupportedProductNames ()

  Gets an array of all the product names supported by this USB device family.
- static IntArray getSupportedProductIDs ()

  Gets an array of all the product IDs supported by this USB device family.
- static bool isSupportedProductID (int productID)

  Tells if a given product ID is supported by this USB device family.

Protected Member Functions

- USB_DIO_Family (int productID, int deviceIndex)
- virtual ∼USB_DIO_Family ()

Protected Attributes

- DigitalIOSubsystem digitalIOSubsystem

Static Private Member Functions

- static void initialize ()

Static Private Attributes

- static IntArray supportedProductIDs

Friends

- class USBDeviceManager

Additional Inherited Members

23.81.1 Detailed Description

Class USB_DIO_Family represents a USB-DIO-family device, which performs basic digital I/O and encompasses the following product IDs: USB_DIO_48, USB_DIO_96, USB_IIRO_16, USB_II_16, USB_RO_16, USB_IIRO_8, USB_II_8, USB_IIRO_4, USB_IDIO_16, USB_II_16_OLD, USB_IDO_16, USB_IDIO_8, USB_II_8_OLD, USB_IDIO_4, USB_IIRO4_2SM, USB_IIRO4_COM, USB_DIO16RO8, PICO_DIO16RO8.

Instances of class *USB_DIO_Family* are automatically created by the USB device manager when they are detected on the bus. You should use one of the *USBDeviceManager* search methods, such as *USBDeviceManager::getDeviceByProductID(int productID) const*, to obtain a reference to a *USB_DIO_Family* instance. You can then cast the *USBDeviceBase* reference obtained from one of those methods to a *USB_DIO_Family* and make use of this class' methods, like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB_DIO_48, USB_DIO_96 );
if( devices.size() > 0 )
  USB_DIO_Family &device = *( USB_DIO_Family * ) devices.at( 0 );
```

23.81.2 Constructor & Destructor Documentation

USB_DIO_Family (int *productID*, int *deviceIndex*) `[protected]`

~USB_DIO_Family ( ) `[protected], [virtual]`

23.81.3 Member Function Documentation

void initialize ( ) `[static], [private]`

StringArray getSupportedProductNames ( ) `[static]`

Gets an array of all the product names supported by this USB device family.

Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *USBDeviceManager::open()* method is called.

Returns

An array of product names, sorted in ascending order of product ID.

IntArray getSupportedProductIDs ( ) `[static]`

Gets an array of all the product IDs supported by this USB device family.

Returns

An array of product IDs, sorted in ascending order.

bool isSupportedProductID (int *productID*) `[static]`

Tells if a given product ID is supported by this USB device family.

Parameters

productID	the product ID to check.

Returns

True if the given product ID is supported by this USB device family; otherwise, *false*.

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented from USBDeviceBase.

DigitalIOSubsystem& dio ( ) `[inline]`

Gets a reference to the digital I/O subsystem of this device.

Returns

A reference to the digital I/O subsystem.

23.81.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

23.81.5 Field Documentation

IntArray supportedProductIDs `[static], [private]`

DigitalIOSubsystem digitalIOSubsystem `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USB_DIO_Family.hpp
- deprecated/classlib/USB_DIO_Family.cpp

23.82 USBDevice Struct Reference

```
#include <USBDevice.h>
```

Data Fields

- int(∗ usb_control_transfer)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int(∗ usb_bulk_transfer)(USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int(∗ usb_request)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int(∗ usb_reset_device)(USBDevice ∗usbdev)
- int(∗ usb_put_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int(∗ usb_get_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- uint8_t timeout
- libusb_device ∗ device
- libusb_device_handle ∗ deviceHandle
- struct libusb_device_descriptor deviceDesc
- AIOUSB_BOOL debug
- int usblp_attached
- int iface
- int verbose
- int conf
- int origconf
- int altset

23.82.1 Field Documentation

int(∗ usb_control_transfer)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)

int(∗ usb_bulk_transfer)(USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)

int(∗ usb_request)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)

int(∗ usb_reset_device)(USBDevice ∗usbdev)

int(∗ usb_put_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)

int(∗ usb_get_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)

uint8_t timeout

libusb_device∗ device

libusb_device_handle∗ deviceHandle

struct libusb_device_descriptor deviceDesc

AIOUSB_BOOL debug

int usblp_attached

int iface

int verbose

int conf

int origconf

int altset

The documentation for this struct was generated from the following file:

- lib/USBDevice.h

23.83 USBDeviceArray Class Reference

`#include <USBDeviceBase.hpp>`

Public Member Functions

- USBDeviceArray (int size=0)

23.83.1 Constructor & Destructor Documentation

USBDeviceArray (int *size* = 0) `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceBase.hpp

23.84 USBDeviceBase Class Reference

Class USBDeviceBase is the abstract super class of all USB device families.

`#include <USBDeviceBase.hpp>`

Public Member Functions

- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device and all of its subsystems.
- int getDeviceIndex () const

  Gets the device's index on the USB bus.
- int getProductID () const

  Gets the device's product ID.
- const std::string & getName () const

  Gets the device's name.
- __uint64_t getSerialNumber () const

  Gets the device's serial number.
- int getCommTimeout () const

  Gets the current timeout setting for USB communications.
- USBDeviceBase & setCommTimeout (int timeout)

  Sets the timeout for USB communications.
- USBDeviceBase & reset ()

  Perform a USB port reset to reinitialize the device.
- USBDeviceBase & customEEPROMWrite (int address, const UCharArray &data)

  Writes data to the custom programming area of the device EEPROM.
- UCharArray customEEPROMRead (int address, int numBytes)

  Reads data from the custom programming area of the device EEPROM.

Static Public Attributes

- static const int CUSTOM_EEPROM_SIZE = 0x200

  Size of custom EEPROM area (bytes).
- static const int CLEAR_FIFO_METHOD_IMMEDIATE = 0

  Clear FIFO as soon as command received (and disable auto-clear).
- static const int CLEAR_FIFO_METHOD_AUTO = 1

  Enable auto-clear FIFO every falling edge of DIO port D bit 1 (on digital boards, analog boards treat as 0).
- static const int CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT = 5

  Clear FIFO as soon as command received (and disable auto-clear), and abort stream.
- static const int CLEAR_FIFO_METHOD_WAIT = 86

  Clear FIFO and wait for it to be emptied.

Protected Member Functions

- USBDeviceBase (int productID, int deviceIndex)
- virtual ∼USBDeviceBase ()
- USBDeviceBase & clearFIFO (FIFO_Method method)
- double getMiscClock ()
- USBDeviceBase & setMiscClock (double clockHz)
- int getStreamingBlockSize ()
- USBDeviceBase & setStreamingBlockSize (int blockSize)

Protected Attributes

- int deviceIndex
- int productID
- std::string name
- __uint64_t serialNumber

Friends

- class USBDeviceManager
- class DIOStreamSubsystem
- class AnalogInputSubsystem

23.84.1 Detailed Description

Class USBDeviceBase is the abstract super class of all USB device families.

23.84.2 Constructor & Destructor Documentation

USBDeviceBase (int *productID,* int *deviceIndex*) [protected]

~USBDeviceBase ( ) [protected], [virtual]

23.84.3 Member Function Documentation

USBDeviceBase & clearFIFO (FIFO_Method *method*) [protected]

double getMiscClock ( ) [inline], [protected]

USBDeviceBase & setMiscClock (double *clockHz*) [protected]

int getStreamingBlockSize ( ) [protected]

USBDeviceBase & setStreamingBlockSize (int *blockSize*) [protected]

ostream & print (std::ostream & *out*) [virtual]

Prints the properties of this device and all of its subsystems.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

Reimplemented in USB_AI16_Family, USB_DIO_Family, USB_AO16_Family, USB_DA12_8A_Family, USB_DIO_16_-Family, USB_DIO_32_Family, USB_AIO16_Family, USB_DA12_8E_Family, and USB_CTR_15_Family.

int getDeviceIndex ( ) const [inline]

Gets the device's index on the USB bus.

The device index isn't used within this Java class library, but is used in the underlying AIOUSB library. The device index is somewhat useful within this Java class library to differentiate between multiple devices of the same type.

Returns

The index of the device on the USB bus.

int getProductID ( ) const [inline]

Gets the device's product ID.

Returns

The device product ID.

const std::string& getName ( ) const [inline]

Gets the device's name.

Returns

The device name.

__uint64_t getSerialNumber ( ) const [inline]

Gets the device's serial number.

Returns

The device serial number (a 64-bit integer).

Exceptions

OperationFailedException	

int getCommTimeout ( ) const

Gets the current timeout setting for USB communications.

Returns

Current timeout setting (in milliseconds).

See Also

setCommTimeout(int timeout)

USBDeviceBase & setCommTimeout (int *timeout*)

Sets the timeout for USB communications.

Parameters

timeout	the new timeout setting (in milliseconds; default is 5,000).

Returns

This device, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

USBDeviceBase & reset ( )

Perform a USB port reset to reinitialize the device.

Returns

This device, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceBase & customEEPROMWrite (int *address*, const UCharArray & *data*)

Writes data to the custom programming area of the device EEPROM.

Beware that writing to the EEPROM is particularly slow. Writing the entire EEPROM may take several seconds. Before initiating a lengthy EEPROM write procedure, it is recommended that the communication timeout be increased to at least five seconds, if not ten *(see setCommTimeout(int timeout))*. Otherwise, a timeout error will occur before the write procedure finishes. Once the write procedure is finished, you can restore the timeout to a more reasonable value. If you are writing a smaller amount of data to the EEPROM, you may reduce the timeout proportionately.

Parameters

address	starting address from 0x000 to 0x1FF within the EEPROM.
data	an array of bytes containing the data to write to the EEPROM, beginning at the starting address. The starting address plus the data size may not exceed the maximum address of 0x1FF.

Returns

This device, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
OperationFailedException	

UCharArray customEEPROMRead (int *address*, int *numBytes*)

Reads data from the custom programming area of the device EEPROM.

Parameters

address	starting address from 0x000 to 0x1FF within the EEPROM.
numBytes	the number of bytes to read from the EEPROM, beginning at the starting address. The starting address plus the number of bytes to read may not exceed the maximum address of 0x1FF.

Returns

An array of bytes containing the data read from the EEPROM. The length of the array will be equal to *numBytes*.

Exceptions

IllegalArgumentException	
OperationFailedException	

23.84.4 Friends And Related Function Documentation

friend class USBDeviceManager `[friend]`

friend class DIOStreamSubsystem `[friend]`

friend class AnalogInputSubsystem `[friend]`

23.84.5 Field Documentation

const int CUSTOM_EEPROM_SIZE = 0x200 `[static]`

Size of custom EEPROM area (bytes).

const int CLEAR_FIFO_METHOD_IMMEDIATE = 0 `[static]`

Clear FIFO as soon as command received (and disable auto-clear).

const int CLEAR_FIFO_METHOD_AUTO = 1 `[static]`

Enable auto-clear FIFO every falling edge of DIO port D bit 1 (on digital boards, analog boards treat as 0).

const int CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT = 5 `[static]`

Clear FIFO as soon as command received (and disable auto-clear), and abort stream.

const int CLEAR_FIFO_METHOD_WAIT = 86 `[static]`

Clear FIFO and wait for it to be emptied.

int deviceIndex `[protected]`

int productID `[protected]`

std::string name `[protected]`

__uint64_t serialNumber `[protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USBDeviceBase.hpp
- deprecated/classlib/USBDeviceBase.cpp

23.85 USBDeviceManager Class Reference

Class USBDeviceManager manages all the USB devices on the bus.

```
#include <USBDeviceManager.hpp>
```

Public Member Functions

- USBDeviceManager ()
- virtual ∼USBDeviceManager ()
- virtual std::ostream & print (std::ostream &out)

  Prints the properties of this device manager and all of the devices found on the bus to the specified print stream.
- USBDeviceManager & printDevices ()

  Prints the properties of this device manager and all of the devices found on the bus to the standard output device.
- std::string getAIOUSBVersion () const

  Gets the version number of the underlying AIOUSB module.
- std::string getAIOUSBVersionDate () const

  Gets the version date of the underlying AIOUSB module.
- void listDevices () const

  Prints the properties of all the devices found on the bus to the standard output device.
- bool isOpen () const

  Tells if the USB device manager has been "opened" for use (see open()).
- USBDeviceManager & open ()

  "Opens" the USB device manager for use.
- USBDeviceManager & close ()

  "Closes" the USB device manager for use.
- USBDeviceManager & scanForDevices ()

  Re-scans the bus for devices.
- USBDeviceArray getDeviceByProductID (int productID) const

  Gets a list of all the devices found on the bus matching the specified product ID.
- USBDeviceArray getDeviceByProductID (int minProductID, int maxProductID) const

  Gets a list of all the devices found on the bus matching the specified product ID range.
- USBDeviceArray getDeviceByProductID (const IntArray &productIDs) const

  Gets a list of all the devices found on the bus matching the specified set of product IDs.
- USBDeviceArray getDeviceBySerialNumber (__uint64_t serialNumber) const

  Gets a list of all the devices found on the bus matching the specified serial number.

Static Public Member Functions

- static std::string productIDToName (int productID)

  Gets the product name for a product ID.
- static StringArray productIDToName (const IntArray &productID)

  Gets the product names for an array of product IDs.
- static int productNameToID (const std::string &productName)

  Gets the product ID for a product name.
- static IntArray productNameToID (const StringArray &productName)

  Gets the product IDs for an array of product names.
- static std::string getResultCodeAsString (int result)

  Gets the string representation of an AIOUSB result code, useful mainly for debugging purposes.

Static Public Attributes

- static const std::string VERSION_NUMBER = "1.8"

  The version number of this Java class library.
- static const std::string VERSION_DATE = "18 January 2010"

  The version date of this Java class library.
- static const int MIN_PRODUCT_ID = 0
- static const int MAX_PRODUCT_ID = 0xffff

Protected Member Functions

- void emptyDeviceList ()

Protected Attributes

- USBDeviceArray deviceList
- unsigned long openStatus

Static Protected Attributes

- static const unsigned long OPEN_PATTERN = 0x786938f5
- static const std::string MESSAGE_NOT_OPEN = "Not open, must call open() first"

23.85.1 Detailed Description

Class USBDeviceManager manages all the USB devices on the bus.

It scans the bus and builds a list of all the devices found. It also initializes and terminates use of the underlying AIOUSB module.

23.85.2 Constructor & Destructor Documentation

USBDeviceManager ( )

∼USBDeviceManager ( ) `[virtual]`

23.85.3 Member Function Documentation

void emptyDeviceList ( ) `[protected]`

ostream & print (std::ostream & *out*) `[virtual]`

Prints the properties of this device manager and all of the devices found on the bus to the specified print stream.

Mainly useful for diagnostic purposes.

Parameters

out	the print stream where properties will be printed.

Returns

The print stream.

USBDeviceManager & printDevices ( )

Prints the properties of this device manager and all of the devices found on the bus to the standard output device.

Mainly useful for diagnostic purposes.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

std::string getAIOUSBVersion ( ) const `[inline]`

Gets the version number of the underlying AIOUSB module.

Returns

The AIOUSB module version number as a string with the form, "1.78".

std::string getAIOUSBVersionDate ( ) const `[inline]`

Gets the version date of the underlying AIOUSB module.

Returns

The AIOUSB module version date as a string with the form, "15 November 2009".

std::string productIDToName (int *productID*) `[static]`

Gets the product name for a product ID.

This name is only "approximate," as an actual device reports its own name. Generally the names reported by the device are the same as those obtained from this method, but that is not guaranteed. This method provides a name that constitutes a user-friendly alternative to a product ID number. The complement of this method is *productNameToID(const std::string &productName)*. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productID	the product ID to translate to a product name.

Returns

A string containing the product name, or "UNKNOWN" if the product ID was not found.

Exceptions

IllegalArgumentException	

StringArray productIDToName (const IntArray & *productID*) `[static]`

Gets the product names for an array of product IDs.

Functionally identical to *productIDToName(int productID)* except that it operates on an array of product IDs rather than an individual product ID. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productID	an array of product IDs to translate to product names.

Returns

An array of strings containing the product names, or "UNKNOWN" for any product ID that was not found. The product names are returned in the same order as the product IDs passed in *productID[]*.

Exceptions

IllegalArgumentException	

int productNameToID (const std::string & *productName*) `[static]`

Gets the product ID for a product name.

This method is the complement of *productIDToName(int productID)* and one should read the notes for that method. It is not guaranteed that *productNameToID()* will successfully ascertain the product ID for a name obtained from a device, although it usually will. *ProductNameToID()* will always successfully ascertain the product ID for a name obtained from *productIDToName()*. If one has access to a device and its name, then they should obtain the product ID from the device itself rather than from this method. This method is mainly for easily converting between product names and IDs, primarily to serve the needs of user interfaces. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productName	the product name to translate to a product ID.

Returns

The product ID for the specified product name, or 0 (zero) if the name was not found.

Exceptions

IllegalArgumentException	

IntArray productNameToID (const StringArray & *productName*) `[static]`

Gets the product IDs for an array of product names.

Functionally identical to *productNameToID(const std::string &productName)* except that it operates on an array of product names rather than an individual product name. Although this method is *static*, an instance of USBDevice-Manager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for product name/ID lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

productName	an array of product names to translate to product IDs.

Returns

An array of integers containing the product IDs, or 0 (zero) for any product name that was not found. The product IDs are returned in the same order as the product names passed in *productName[]*.

Exceptions

IllegalArgumentException	

void listDevices ( ) const `[inline]`

Prints the properties of all the devices found on the bus to the standard output device.

This function is similar to *printDevices()* but is implemented by the underlying AIOUSB module and produces different output than *printDevices()*. Mainly useful for diagnostic purposes.

static std::string getResultCodeAsString (int *result*) `[inline], [static]`

Gets the string representation of an AIOUSB result code, useful mainly for debugging purposes.

This method is also used to convert an AIOUSB result code to a string when an OperationFailedException is thrown in response to an AIOUSB failure. Although this method is *static*, an instance of USBDeviceManager must be created and be "open" for use before this method can be used. This stipulation is imposed because the underlying library must be initialized in order for result code lookups to succeed, and that initialization occurs only when an instance of USBDeviceManager is created and its *open()* method is called.

Parameters

result	an AIOUSB result code.

Returns

The string representation of *result*.

bool isOpen ( ) const `[inline]`

Tells if the USB device manager has been "opened" for use *(see open())*.

Returns

True indicates that the device manager is open and ready to be used; *false* indicates that it is not open.

USBDeviceManager & open ( )

"Opens" the USB device manager for use.

Before the USB device manager may be used, *open()* must be called. *Open()* initializes the underlying AIOUSB module and scans the bus for devices, building a list of the devices found. When finished using the USB device manager, *close()* must be called. It is possible to call *close()* and then call *open()* again, which effectively reinitializes everything.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceManager & close ( )

"Closes" the USB device manager for use.

When finished using the USB device manager, and assuming *open()* was properly called, *close()* must be called. *Close()* terminates use of the underlying AIOUSB module and discards the list of devices found. *You must terminate use of all USB devices before calling close()!* You can call *open()* again to reinitialize things and reestablish connections to USB devices.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceManager & scanForDevices ( )

Re-scans the bus for devices.

ScanForDevices() is called automatically by *open()*. *You must terminate use of all USB devices before calling scanForDevices()!* After calling *scanForDevices()* you can reestablish connections to USB devices.

Returns

This device manager, useful for chaining together multiple operations.

Exceptions

OperationFailedException	

USBDeviceArray getDeviceByProductID (int *productID*) const

Gets a list of all the devices found on the bus matching the specified product ID.

Only devices exactly matching the specified product ID will be returned. You can search for devices by product name using productNameToID(const std::string &productName), like so:

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( deviceManager.productNameToID( "USB-CTR-
```

Parameters

productID	the product ID to search for.

Returns

An array of all the devices found. If no devices were found matching the specified product ID, the array will be empty (i.e. contain zero items).

USBDeviceArray getDeviceByProductID (int *minProductID,* int *maxProductID*) const

Gets a list of all the devices found on the bus matching the specified product ID range.

Any device with a product ID greater than or equal to *minProductID* and less than or equal to *maxProductID* will be returned. You can obtain the entire list of devices detected by passing a value of 0 for *minProductID* and a value of 0xffff for *maxProductID*. Then you can search the list obtained using your own search criteria.

Parameters

minProductID	the minimum product ID to search for.
maxProductID	the maximum product ID to search for.

Returns

An array of all the devices found. If no devices were found matching the specified product ID range, the array will be empty (i.e. contain zero items).

Exceptions

IllegalArgumentException	

USBDeviceArray getDeviceByProductID (const IntArray & *productIDs*) const

Gets a list of all the devices found on the bus matching the specified set of product IDs.

Any device with a product ID equal to one of the products listed in *productIDs[]* will be returned.

Parameters

productIDs	an array containing one or more product IDs to search for.

Returns

An array of all the devices found. If no devices were found matching the specified set of product IDs, the array will be empty (i.e. contain zero items).

Exceptions

IllegalArgumentException	

USBDeviceArray getDeviceBySerialNumber (__uint64_t *serialNumber*) const

Gets a list of all the devices found on the bus matching the specified serial number.

Only devices exactly matching the specified serial number will be returned. In theory, there ought to be only one device matching a given serial number, but this method returns a vector in order to be consistent with the other search methods, and in unlikely event that multiple devices do share the same serial number.

Parameters

serialNumber	the serial number to search for.

Returns

An array of all the devices found. If no devices were found matching the specified serial number, the array will be empty (i.e. contain zero items).

23.85.4 Field Documentation

const std::string VERSION_NUMBER = "1.8" `[static]`

The version number of this Java class library.

const std::string VERSION_DATE = "18 January 2010" `[static]`

The version date of this Java class library.

const int MIN_PRODUCT_ID = 0 `[static]`

const int MAX_PRODUCT_ID = 0xffff `[static]`

USBDeviceArray deviceList `[protected]`

const unsigned long OPEN_PATTERN = 0x786938f5 `[static], [protected]`

unsigned long openStatus `[protected]`

const std::string MESSAGE_NOT_OPEN = "Not open, must call open() first" `[static], [protected]`

The documentation for this class was generated from the following files:

- deprecated/classlib/USBDeviceManager.hpp
- deprecated/classlib/USBDeviceManager.cpp

23.86 ushort_array Struct Reference

```
#include <AIOTypes.h>
```

Data Fields

- unsigned size

23.86.1 Field Documentation

unsigned size

The documentation for this struct was generated from the following file:

- lib/AIOTypes.h

23.87 UShortArray Class Reference

```
#include <USBDeviceBase.hpp>
```

Public Member Functions

- UShortArray (int size=0)

23.87.1 Constructor & Destructor Documentation

UShortArray (int *size* = 0) `[inline]`

The documentation for this class was generated from the following file:

- deprecated/classlib/USBDeviceBase.hpp

Chapter 24

File Documentation

24.1 deprecated/classlib/AI16_DataPoint.cpp File Reference

```
#include <sstream>
#include <iomanip>
#include "AI16_DataPoint.hpp"
#include "AnalogInputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.2 deprecated/classlib/AI16_DataPoint.hpp File Reference

```
#include <string>
#include <vector>
```

Data Structures

- class AI16_DataPoint

  Class AI16_DataPoint represents a single data point captured from a USB_AI16_Family device.
- class AI16_DataPointArray

Namespaces

- AIOUSB

24.3 deprecated/classlib/AI16_DataSet.cpp File Reference

```
#include <iomanip>
#include <assert.h>
#include "AI16_DataSet.hpp"
#include "AnalogInputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.4 deprecated/classlib/AI16_DataSet.hpp File Reference

```
#include <ostream>
#include <time.h>
#include <AI16_DataPoint.hpp>
```

Data Structures

- class AI16_DataSet

  Class AI16_DataSet represents a data set captured from a USB_AI16_Family device.

Namespaces

- AIOUSB

24.5 deprecated/classlib/AI16_InputRange.cpp File Reference

```
#include "AnalogInputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.6 deprecated/classlib/AI16_InputRange.hpp File Reference

```
#include <AnalogIORange.hpp>
```

Data Structures

- class AI16_InputRange

Namespaces

- AIOUSB

24.7 deprecated/classlib/AnalogInputSubsystem.cpp File Reference

class AnalogInputSubsystem implementation

```
#include "CppCommon.h"
#include <assert.h>
#include <string.h>
#include <AIOUSB_Core.h>
#include "AIOUSB_ADC.h"
#include "AIODeviceTable.h"
#include "AIOUSBDevice.h"
#include "aiousb.h"
#include "USBDeviceManager.hpp"
#include "AnalogInputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.7.1 Detailed Description

class AnalogInputSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

24.8 deprecated/classlib/AnalogInputSubsystem.hpp File Reference

```
#include <AI16_InputRange.hpp>
#include <AI16_DataSet.hpp>
#include <DeviceSubsystem.hpp>
```

Data Structures

- class AnalogInputSubsystem

    Class AnalogInputSubsystem represents the analog input subsystem of a device.

Namespaces

- AIOUSB

24.9 deprecated/classlib/AnalogIORange.cpp File Reference

```
#include <assert.h>
#include <math.h>
#include "AnalogIORange.hpp"
```

Namespaces

- AIOUSB

24.10 deprecated/classlib/AnalogIORange.hpp File Reference

```
#include <USBDeviceManager.hpp>
```

Data Structures

- class AnalogIORange

    Class AnalogIORange helps manage analog I/O range settings and provides voltage-count conversion utilities.

Namespaces

- AIOUSB

24.11 deprecated/classlib/AnalogOutputSubsystem.cpp File Reference

```
#include "CppCommon.h"
#include <assert.h>
#include <math.h>
#include <aiousb.h>
#include "USBDeviceManager.hpp"
#include "AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.12 deprecated/classlib/AnalogOutputSubsystem.hpp File Reference

```
#include <DeviceSubsystem.hpp>
```

Data Structures

- class AnalogOutputSubsystem

  Class AnalogOutputSubsystem is the superclass of the analog output subsystem of a device.

Namespaces

- AIOUSB

24.13 deprecated/classlib/AO16_AnalogOutputSubsystem.cpp File Reference

```
#include "CppCommon.h"
#include <assert.h>
#include <aiousb.h>
#include "USBDeviceManager.hpp"
#include "AO16_AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.14 deprecated/classlib/AO16_AnalogOutputSubsystem.hpp File Reference

```
#include <AnalogOutputSubsystem.hpp>
#include <AO16_OutputRange.hpp>
#include <OutputVoltagePoint.hpp>
```

Data Structures

- class AO16_AnalogOutputSubsystem

  Class AO16_AnalogOutputSubsystem represents the analog output subsystem of a device.

Namespaces

- AIOUSB

24.15 deprecated/classlib/AO16_OutputRange.cpp File Reference

```
#include "AO16_OutputRange.hpp"
#include "AO16_AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.16 deprecated/classlib/AO16_OutputRange.hpp File Reference

```
#include <AnalogIORange.hpp>
```

Data Structures

- class AO16_OutputRange

Namespaces

- AIOUSB

24.17 deprecated/classlib/Counter.cpp File Reference

class Counter implementation

```
#include "CppCommon.h"
#include <assert.h>
#include <typeinfo>
#include <AIOUSB_Core.h>
#include "USBDeviceManager.hpp"
#include "Counter.hpp"
#include "CounterSubsystem.hpp"
#include "USB_CTR_15_Family.hpp"
```

Namespaces

- AIOUSB

24.17.1 Detailed Description

class Counter implementation

Author

Format:

an <ae>

Date

Format:

ad

24.18 deprecated/classlib/Counter.hpp File Reference

```
#include <vector>
```

Data Structures

- class Counter

  Class Counter represents a single counter/timer.
- class CounterList

Namespaces

- AIOUSB

24.19 deprecated/classlib/CounterSubsystem.cpp File Reference

class CounterSubsystem implementation

```
#include "CppCommon.h"
#include <assert.h>
#include <typeinfo>
#include <AIOUSB_Core.h>
#include "USBDeviceManager.hpp"
#include "CounterSubsystem.hpp"
#include "USB_CTR_15_Family.hpp"
```

Namespaces

- AIOUSB

24.19.1 Detailed Description

class CounterSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

24.20 deprecated/classlib/CounterSubsystem.hpp File Reference

```
#include <DeviceSubsystem.hpp>
#include <Counter.hpp>
```

Data Structures

- class CounterSubsystem

  Class CounterSubsystem represents the counter/timer subsystem of a device.

Namespaces

- AIOUSB

24.21 deprecated/classlib/CppCommon.h File Reference

24.22 deprecated/classlib/DA12_AnalogOutputSubsystem.cpp File Reference

class DA12_AnalogOutputSubsystem implementation

```
#include "CppCommon.h"
#include <assert.h>
#include <aiousb.h>
#include "USBDeviceManager.hpp"
#include "DA12_AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.22.1 Detailed Description

class DA12_AnalogOutputSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

24.23 deprecated/classlib/DA12_AnalogOutputSubsystem.hpp File Reference

```
#include <AnalogOutputSubsystem.hpp>
#include <DA12_OutputRange.hpp>
#include <OutputVoltagePoint.hpp>
```

Data Structures

- class DA12_AnalogOutputSubsystem

  Class DA12_AnalogOutputSubsystem represents the analog output subsystem of a device.

Namespaces

- AIOUSB

24.24 deprecated/classlib/DA12_OutputRange.cpp File Reference

```
#include "DA12_OutputRange.hpp"
#include "DA12_AnalogOutputSubsystem.hpp"
```

Namespaces

- AIOUSB

24.25 deprecated/classlib/DA12_OutputRange.hpp File Reference

```
#include <AnalogIORange.hpp>
```

Data Structures

- class DA12_OutputRange

Namespaces

- AIOUSB

24.26 deprecated/classlib/DeviceSubsystem.cpp File Reference

```
#include "DeviceSubsystem.hpp"
#include <assert.h>
```

Namespaces

- AIOUSB

24.27 deprecated/classlib/DeviceSubsystem.hpp File Reference

```
#include <USBDeviceBase.hpp>
```

Data Structures

- class DeviceSubsystem
  Class DeviceSubsystem is the abstract super class for all device subsystems.

Namespaces

- AIOUSB

24.28 deprecated/classlib/DigitalIOSubsystem.cpp File Reference

class DigitalIOSubsystem implementation

```
#include "CppCommon.h"
#include <assert.h>
#include "AIOUSB_Core.h"
#include "AIOTypes.h"
#include "DIOBuf.h"
#include "USBDeviceManager.hpp"
#include "DigitalIOSubsystem.hpp"
```

Namespaces

- AIOUSB

24.28.1 Detailed Description

class DigitalIOSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

24.29 deprecated/classlib/DigitalIOSubsystem.hpp File Reference

```
#include "AIOTypes.h"
#include <DeviceSubsystem.hpp>
```

Data Structures

- class DigitalIOSubsystem

  Class DigitalIOSubsystem represents the digital I/O subsystem of a device.

Namespaces

- AIOUSB

24.30 deprecated/classlib/DIOStreamSubsystem.cpp File Reference

class DIOStreamSubsystem implementation

```
#include "CppCommon.h"
#include <assert.h>
#include <aiousb.h>
#include "USBDeviceManager.hpp"
#include "DIOStreamSubsystem.hpp"
```

Namespaces

- AIOUSB

24.30.1 Detailed Description

class DIOStreamSubsystem implementation

Author

Format:

an <ae>

Date

Format:

ad

24.31 deprecated/classlib/DIOStreamSubsystem.hpp File Reference

```
#include <DeviceSubsystem.hpp>
```

Data Structures

- class DIOStreamSubsystem

  Class DIOStreamSubsystem represents the digital I/O streaming subsystem of a device.

Namespaces

- AIOUSB

24.32 deprecated/classlib/OutputVoltagePoint.hpp File Reference

```
#include <vector>
```

Data Structures

- class OutputVoltagePoint

  Class OutputVoltagePoint represents a single analog output data point, consisting of a D/A channel number and a voltage to output to that channel.
- class OutputVoltagePointArray

Namespaces

- AIOUSB

24.33 deprecated/classlib/README.doc File Reference

24.34 lib/wrappers/README.doc File Reference

24.35 Firmware/README.doc File Reference

24.36 samples/USB-AI16-16/README.doc File Reference

24.37 samples/USB-AO16-16/README.doc File Reference

24.38 samples/USB-DA12-8A/README.doc File Reference

24.39 samples/USB-DIO-16/README.doc File Reference

24.40 samples/USB-DIO-32/README.doc File Reference

24.41 samples/USB-IDIO-16_8/README.doc File Reference

24.42 samples/USB-IIRO-16_8/README.doc File Reference

24.43 deprecated/classlib/USB_AI16_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_AI16_Family.hpp"
```

Namespaces

- AIOUSB

24.44 deprecated/classlib/USB_AI16_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <AnalogInputSubsystem.hpp>
#include <DigitalIOSubsystem.hpp>
#include <CounterSubsystem.hpp>
```

Data Structures

- class USB_AI16_Family

Namespaces

- AIOUSB

24.45 deprecated/classlib/USB_AIO16_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_AIO16_Family.hpp"
```

Namespaces

- AIOUSB

24.46 deprecated/classlib/USB_AIO16_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <AnalogInputSubsystem.hpp>
#include <AnalogOutputSubsystem.hpp>
#include <DigitalIOSubsystem.hpp>
#include <CounterSubsystem.hpp>
```

Data Structures

- class USB_AIO16_Family

  Class USB_AIO16_Family represents a USB-AI16-family device, which encompasses the following product IDs: USB-_AI16_16A, USB_AI16_16E, USB_AI12_16A, USB_AI12_16, USB_AI12_16E, USB_AI16_64MA, USB_AI16_64ME, U-SB_AI12_64MA, USB_AI12_64M, USB_AI12_64ME, USB_AI16_32A, USB_AI16_32E, USB_AI12_32A, USB_AI12_32, USB_AI12_32E, USB_AI16_64A, USB_AI16_64E, USB_AI12_64A, USB_AI12_64, USB_AI12_64E, USB_AI16_96A, U-SB_AI16_96E, USB_AI12_96A, USB_AI12_96, USB_AI12_96E, USB_AI16_128A, USB_AI16_128E, USB_AI12_128A, USB_AI12_128, USB_AI12_128E.

Namespaces

- AIOUSB

24.47 deprecated/classlib/USB_AO16_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_AO16_Family.hpp"
```

Namespaces

- AIOUSB

24.48 deprecated/classlib/USB_AO16_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <AO16_AnalogOutputSubsystem.hpp>
#include <DigitalIOSubsystem.hpp>
```

Data Structures

- class USB_AO16_Family

  Class USB_AO16_Family represents a USB-AO16-family device, which encompasses the following product IDs: USB_-AO16_16A, USB_AO16_16, USB_AO16_12A, USB_AO16_12, USB_AO16_8A, USB_AO16_8, USB_AO16_4A, USB_-AO16_4, USB_AO12_16A, USB_AO12_16, USB_AO12_12A, USB_AO12_12, USB_AO12_8A, USB_AO12_8, USB_A-O12_4A, USB_AO12_4.

Namespaces

- AIOUSB

24.49 deprecated/classlib/USB_CTR_15_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_CTR_15_Family.hpp"
```

Namespaces

- AIOUSB

24.50 deprecated/classlib/USB_CTR_15_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <CounterSubsystem.hpp>
```

Data Structures

- class USB_CTR_15_Family

  Class USB_CTR_15_Family represents a USB-CTR-15-family device, which encompasses the following product IDs: U-SB_CTR_15.

Namespaces

- AIOUSB

24.51 deprecated/classlib/USB_DA12_8A_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_DA12_8A_Family.hpp"
```

Namespaces

- AIOUSB

24.52 deprecated/classlib/USB_DA12_8A_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <DA12_AnalogOutputSubsystem.hpp>
#include <DigitalIOSubsystem.hpp>
```

Data Structures

- class USB_DA12_8A_Family

    Class USB_DA12_8A_Family represents a USB-DA12-8A-family device, which encompasses the following product IDs: USB_DA12_8A_REV_A, USB_DA12_8A.

Namespaces

- AIOUSB

24.53 deprecated/classlib/USB_DA12_8E_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_DA12_8E_Family.hpp"
```

Namespaces

- AIOUSB

24.54 deprecated/classlib/USB_DA12_8E_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <DA12_AnalogOutputSubsystem.hpp>
#include <DigitalIOSubsystem.hpp>
```

Data Structures

- class USB_DA12_8E_Family

    Class USB_DA12_8E_Family represents a USB-DA12-8E-family device, which encompasses the following product IDs: USB_DA12_8E.

Namespaces

- AIOUSB

24.55 deprecated/classlib/USB_DIO_16_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_DIO_16_Family.hpp"
```

Namespaces

- AIOUSB

24.56 deprecated/classlib/USB_DIO_16_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <DigitalIOSubsystem.hpp>
#include <DIOStreamSubsystem.hpp>
```

Data Structures

- class USB_DIO_16_Family

    Class USB_DIO_16_Family represents a USB-DIO-16-family device, which encompasses the following product IDs: USB_DI16A_REV_A1, USB_DO16A_REV_A1, USB_DI16A_REV_A2, USB_DIO_16H, USB_DI16A, USB_DO16A, USB_DIO_16A.

Namespaces

- AIOUSB

24.57 deprecated/classlib/USB_DIO_32_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_DIO_32_Family.hpp"
```

Namespaces

- AIOUSB

24.58 deprecated/classlib/USB_DIO_32_Family.hpp File Reference

```
#include <USBDeviceBase.hpp>
#include <DigitalIOSubsystem.hpp>
#include <CounterSubsystem.hpp>
```

Data Structures

- class USB_DIO_32_Family

    Class USB_DIO_32_Family represents a USB-DIO-32-family device, which encompasses the following product IDs: USB_DIO_32.

Namespaces

- AIOUSB

24.59 deprecated/classlib/USB_DIO_Family.cpp File Reference

```
#include <iostream>
#include <bits/stl_algo.h>
#include <assert.h>
#include "USBDeviceManager.hpp"
#include "USB_DIO_Family.hpp"
```

Namespaces

- AIOUSB

24.60 deprecated/classlib/USB_DIO_Family.hpp File Reference

```
#include "CppCommon.h"
#include <USBDeviceBase.hpp>
#include <DigitalIOSubsystem.hpp>
```

Data Structures

- class USB_DIO_Family

  Class USB_DIO_Family represents a USB-DIO-family device, which performs basic digital I/O and encompasses the following product IDs: USB_DIO_48, USB_DIO_96, USB_IIRO_16, USB_II_16, USB_RO_16, USB_IIRO_8, USB_II_8, USB_IIRO_4, USB_IDIO_16, USB_II_16_OLD, USB_IDO_16, USB_IDIO_8, USB_II_8_OLD, USB_IDIO_4, USB_IIRO4-_2SM, USB_IIRO4_COM, USB_DIO16RO8, PICO_DIO16RO8.

Namespaces

- AIOUSB

24.61 deprecated/classlib/USBDeviceBase.cpp File Reference

```
#include "CppCommon.h"
#include "AIODeviceTable.h"
#include <iomanip>
#include <assert.h>
#include <AIOUSB_Core.h>
#include "USBDeviceBase.hpp"
#include "USBDeviceManager.hpp"
```

Namespaces

- AIOUSB

Functions

- ostream & operator<< (ostream &out, USBDeviceBase &device)
- ostream & operator<< (ostream &out, USBDeviceBase ∗device)

24.62 deprecated/classlib/USBDeviceBase.hpp File Reference

```
#include <vector>
#include <string>
#include <iostream>
#include <aiousb.h>
```

Data Structures

- class BoolArray
- class UCharArray
- class UShortArray
- class IntArray
- class DoubleArray
- class StringArray
- class USBDeviceArray
- class USBDeviceBase

  Class USBDeviceBase is the abstract super class of all USB device families.

Namespaces

- AIOUSB

Functions

- std::ostream & operator<< (std::ostream &out, USBDeviceBase &device)
- std::ostream & operator<< (std::ostream &out, USBDeviceBase ∗device)

24.63 deprecated/classlib/USBDeviceManager.cpp File Reference

class USBDeviceManager implementation

```
#include "CppCommon.h"
#include <iostream>
#include <iterator>
#include <bits/stl_algo.h>
#include <assert.h>
#include <AIOUSB_Core.h>
#include "AIODeviceTable.h"
#include "USBDeviceManager.hpp"
#include "USB_AI16_Family.hpp"
#include "USB_AO16_Family.hpp"
#include "USB_CTR_15_Family.hpp"
#include "USB_DA12_8A_Family.hpp"
#include "USB_DA12_8E_Family.hpp"
#include "USB_DIO_16_Family.hpp"
#include "USB_DIO_32_Family.hpp"
#include "USB_DIO_Family.hpp"
#include "USB_AIO16_Family.hpp"
```

Namespaces

- AIOUSB

24.63.1 Detailed Description

class USBDeviceManager implementation

Author

Format:

an <ae>

Date

Format:

ad

24.64 deprecated/classlib/USBDeviceManager.hpp File Reference

class USBDeviceManager, OperationFailedException, IllegalArgumentException declarations

```
#include "CppCommon.h"
#include <iostream>
#include <stdexcept>
#include <string>
#include <vector>
#include <aiousb.h>
#include <USBDeviceBase.hpp>
```

Data Structures

- class USBDeviceManager

  Class USBDeviceManager manages all the USB devices on the bus.
- class OperationFailedException

  Class OperationFailedException is thrown whenever an operation attempted on a device fails.
- class IllegalArgumentException

  Class IllegalArgumentException is thrown whenever an invalid argument is passed to a method.

Namespaces

- AIOUSB

24.64.1 Detailed Description

class USBDeviceManager, OperationFailedException, IllegalArgumentException declarations

Author

Format:

an <ae>

Date

Format:

ad

24.65 doc/aiousb.doc File Reference

24.66 doc/firmware.doc File Reference

24.67 doc/index.doc File Reference

24.68 doc/install.doc File Reference

24.69 doc/java.doc File Reference

24.70 doc/libusb.doc File Reference

24.71 doc/samples.doc File Reference

24.72 doc/wrappers.doc File Reference

24.73 lib/ADCConfigBlock.c File Reference

```
#include "ADCConfigBlock.h"
#include "AIOUSBDevice.h"
#include "AIOUSB_ADC.h"
#include "AIOUSB_Core.h"
#include "cJSON.h"
#include <ctype.h>
```

Functions

- AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock ∗to, ADCConfigBlock ∗from)
- AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock ∗config)
- AIOUSBDevice ∗ ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock ∗obj, AIORET_TYPE ∗result)
- AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock ∗obj, AIOUSBDevice ∗dev)
- AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock ∗obj, AIOUSBDevice ∗dev)
- AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock ∗config, AIOUSBDevice ∗dev)

    initializes an ADCConfigBlock using parameters from the AIOUSBDevice
- AIORET_TYPE ADCConfigBlockSetSize (ADCConfigBlock ∗obj, unsigned size)
- AIORET_TYPE ADCConfigBlockGetSize (const ADCConfigBlock ∗obj)

- AIORET_TYPE ADCConfigBlockSetTesting (ADCConfigBlock ∗obj, AIOUSB_BOOL testing)
- AIORET_TYPE ADCConfigBlockSetDebug (ADCConfigBlock ∗obj, AIOUSB_BOOL debug)
- AIORET_TYPE ADCConfigBlockSetRangeSingle (ADCConfigBlock ∗config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock ∗config, unsigned reg, unsigned char value)
- AIORET_TYPE ADCConfigBlockGetTesting (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockGetDebug (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock ∗config, AIOUSBDevice ∗deviceDesc, unsigned size)
- AIORET_TYPE ADCConfigBlockInitForCounterScan (ADCConfigBlock ∗config, AIOUSBDevice ∗deviceDesc)
- void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock ∗configBlock, AIOUSBDevice ∗deviceDesc)
- AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock ∗config, unsigned gainCode, A-IOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock ∗config, unsigned channel)
- AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock ∗config, unsigned channel, unsigned char gain-Code)
- AIORET_TYPE ADCConfigBlockSetEndChannel (ADCConfigBlock ∗config, unsigned char endChannel)
- AIORET_TYPE ADCConfigBlockSetChannelRange (ADCConfigBlock ∗config, unsigned startChannel, unsigned endChannel, unsigned gainCode)

  INTERNAL_DOCUMENTATION.
- AIORET_TYPE ADCConfigBlockSetStartChannel (ADCConfigBlock ∗config, unsigned char startChannel)
- AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock ∗config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE ADCConfigBlockSetCalMode (ADCConfigBlock ∗config, ADCalMode calMode)
- AIORET_TYPE ADCConfigBlockGetCalMode (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetStartChannel (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetEndChannel (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetOversample (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetOversample (ADCConfigBlock ∗config, unsigned overSample)
- AIORET_TYPE ADCConfigBlockGetTimeout (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetTimeout (ADCConfigBlock ∗config, unsigned timeout)
- AIORET_TYPE ADCConfigBlockGetTriggerMode (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetTriggerMode (ADCConfigBlock ∗config, unsigned triggerMode)
- AIORET_TYPE ADCConfigBlockSetDifferentialMode (ADCConfigBlock ∗config, unsigned channel, AIOUSB_B-OOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetReference (ADCConfigBlock ∗config, int ref)

  Sets the Timer reference.
- AIORET_TYPE ADCConfigBlockSetTriggerEdge (ADCConfigBlock ∗config, AIOUSB_BOOL val)
- const char ∗ get_gain_code (int code)
- const char ∗ get_cal_mode (int code)
- char ∗ ADCConfigBlockToYAML (ADCConfigBlock ∗config)
- char ∗ ADCConfigBlockToJSON (ADCConfigBlock ∗config)

  INTERNAL_DOCUMENTATION.
- AIORET_TYPE ADCConfigBlockSetScanAllChannels (ADCConfigBlock ∗config, AIOUSB_BOOL val)
- AIORET_TYPE ADCConfigBlockSetTriggerReference (ADCConfigBlock ∗config, int val)
- AIOUSB_BOOL is_all_digits (char ∗str)
- cJSON ∗ ADCConfigBlockGetJSONValueOrDefault (cJSON ∗config, char const ∗key, EnumStringLookup ∗lookup, size_t size)
- cJSON ∗ ADCConfigBlockGetJSONValueOrInt (cJSON ∗config, char const ∗key, int val)
- ADCConfigBlock ∗ NewADCConfigBlockFromJSON (const char ∗str)
- AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock ∗config, int clock_rate)
- AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock ∗config)

24.73.1 Function Documentation

AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock ∗ *to*, ADCConfigBlock ∗ *from*)

AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock ∗ *config*)

AIOUSBDevice∗ ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock ∗ *obj*, AIORET_TYPE ∗ *result*)

AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock ∗ *obj*, AIOUSBDevice ∗ *dev*)

AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock ∗ *obj*, AIOUSBDevice ∗ *dev*)

AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock ∗ *config*)

AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock ∗ *config*, AIOUSBDevice ∗ *dev*)

initializes an ADCConfigBlock using parameters from the AIOUSBDevice

AIORET_TYPE ADCConfigBlockSetSize (**ADCConfigBlock** ∗ *obj,* unsigned *size*)

AIORET_TYPE ADCConfigBlockGetSize (const **ADCConfigBlock** ∗ *obj*)

AIORET_TYPE ADCConfigBlockSetTesting (**ADCConfigBlock** ∗ *obj,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE ADCConfigBlockSetDebug (**ADCConfigBlock** ∗ *obj,* **AIOUSB_BOOL** *debug*)

AIORET_TYPE ADCConfigBlockSetRangeSingle (**ADCConfigBlock** ∗ *config,* unsigned long *channel,* unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetRegister (**ADCConfigBlock** ∗ *config,* unsigned *reg,* unsigned char *value*)

AIORET_TYPE ADCConfigBlockGetTesting (const **ADCConfigBlock** ∗ *obj*)

AIORET_TYPE ADCConfigBlockGetDebug (const **ADCConfigBlock** ∗ *obj*)

AIORET_TYPE ADCConfigBlockInit (**ADCConfigBlock** ∗ *config,* **AIOUSBDevice** ∗ *deviceDesc,* unsigned *size*)

Parameters

config	
deviceDesc	
size	

AIORET_TYPE ADCConfigBlockInitForCounterScan (**ADCConfigBlock** ∗ *config,* **AIOUSBDevice** ∗ *deviceDesc*)

Parameters

config	
deviceDesc	

Returns

void ADC_VerifyAndCorrectConfigBlock (**ADCConfigBlock** ∗ *configBlock,* **AIOUSBDevice** ∗ *deviceDesc*)

AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (**ADCConfigBlock** ∗ *config,* unsigned *gainCode,* **AIOUSB_BOOL** *differentialMode*)

AIORET_TYPE ADCConfigBlockGetGainCode (const **ADCConfigBlock** ∗ *config,* unsigned *channel*)

AIORET_TYPE ADCConfigBlockSetGainCode (**ADCConfigBlock** ∗ *config,* unsigned *channel,* unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetEndChannel (**ADCConfigBlock** ∗ *config,* unsigned char *endChannel*)

AIORET_TYPE ADCConfigBlockSetChannelRange (**ADCConfigBlock** ∗ *config,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

INTERNAL_DOCUMENTATION.

AIORET_TYPE ADCConfigBlockSetStartChannel (**ADCConfigBlock** ∗ *config,* unsigned char *startChannel*)

See Also

USB_SOFTWARE_MANUAL

AIORET_TYPE ADCConfigBlockSetScanRange (**ADCConfigBlock** ∗ *config,* unsigned *startChannel,* unsigned *endChannel*)

this board doesn't have a MUX, so support base number of channels

AIORET_TYPE ADCConfigBlockSetCalMode (**ADCConfigBlock** ∗ *config,* **ADCalMode** *calMode*)

AIORET_TYPE ADCConfigBlockGetCalMode (const **ADCConfigBlock** ∗ *config*)

AIORET_TYPE ADCConfigBlockGetStartChannel (const ADCConfigBlock ∗ *config*)

AIORET_TYPE ADCConfigBlockGetEndChannel (const ADCConfigBlock ∗ *config*)

AIORET_TYPE ADCConfigBlockGetOversample (const ADCConfigBlock ∗ *config*)

AIORET_TYPE ADCConfigBlockSetOversample (ADCConfigBlock ∗ *config,* unsigned *overSample*)

AIORET_TYPE ADCConfigBlockGetTimeout (const ADCConfigBlock ∗ *config*)

AIORET_TYPE ADCConfigBlockSetTimeout (ADCConfigBlock ∗ *config,* unsigned *timeout*)

AIORET_TYPE ADCConfigBlockGetTriggerMode (const ADCConfigBlock ∗ *config*)

AIORET_TYPE ADCConfigBlockSetTriggerMode (ADCConfigBlock ∗ *config,* unsigned *triggerMode*)

AIORET_TYPE ADCConfigBlockSetDifferentialMode (ADCConfigBlock ∗ *config,* unsigned *channel,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE ADCConfigBlockSetReference (ADCConfigBlock ∗ *config,* int *ref*)

Sets the Timer reference.

AIORET_TYPE ADCConfigBlockSetTriggerEdge (ADCConfigBlock ∗ *config,* AIOUSB_BOOL *val*)

const char∗ get_gain_code (int *code*)

const char∗ get_cal_mode (int *code*)

char∗ ADCConfigBlockToYAML (ADCConfigBlock ∗ *config*)

```
* ---
* adcconfig:
*   channels:
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   calibration: Normal
*   trigger:
*     edge: falling edge
*     refchannel: all-channels
*     reference: external
*   oversample: 201
*   clockrate: 1000
*
```

char∗ ADCConfigBlockToJSON (ADCConfigBlock ∗ *config*)

INTERNAL_DOCUMENTATION.

AIORET_TYPE ADCConfigBlockSetScanAllChannels (ADCConfigBlock ∗ *config,* AIOUSB_BOOL *val*)

AIORET_TYPE ADCConfigBlockSetTriggerReference (ADCConfigBlock ∗ *config,* int *val*)

AIOUSB_BOOL is_all_digits (char ∗ *str*)

cJSON∗ ADCConfigBlockGetJSONValueOrDefault (cJSON ∗ *config,* char const ∗ *key,* EnumStringLookup ∗ *lookup,* size_t *size*)

cJSON∗ **ADCConfigBlockGetJSONValueOrInt (cJSON** ∗ *config,* char const ∗ *key,* int *val*)

ADCConfigBlock∗ **NewADCConfigBlockFromJSON (** const char ∗ *str*)

AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock ∗ *config,* int *clock_rate*)

AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock ∗ *config*)

24.74 lib/ADCConfigBlock.h File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <stdlib.h>
#include <string.h>
```

Data Structures

- struct mux_settings
- struct ADCConfigBlock

Typedefs

- typedef struct AIOUSBDevice AIOUSBDevice
- typedef struct mux_settings ADCMuxSettings
- typedef struct ADCConfigBlock ADCConfigBlock
- typedef ADCConfigBlock ADConfigBlock

Functions

- AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock ∗, AIOUSBDevice ∗deviceDesc, unsigned int)
- AIORET_TYPE ADCConfigBlockInitForCounterScan (ADCConfigBlock ∗config, AIOUSBDevice ∗deviceDesc)
- AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock ∗config)
- void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock ∗configBlock, AIOUSBDevice ∗deviceDesc)
- AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock ∗config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock ∗config, unsigned reg, unsigned char value)
- AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock ∗config, unsigned channel)
- AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock ∗config, unsigned channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock ∗config, int clock_rate)
- AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock ∗config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE ADCConfigBlockSetStartChannel (ADCConfigBlock ∗config, unsigned char startChannel)
- AIORET_TYPE ADCConfigBlockSetEndChannel (ADCConfigBlock ∗config, unsigned char endChannel)
- AIORET_TYPE ADCConfigBlockSetChannelRange (ADCConfigBlock ∗config, unsigned startChannel, unsigned endChannel, unsigned gainCode)

  INTERNAL_DOCUMENTATION.
- AIORET_TYPE ADCConfigBlockSetCalMode (ADCConfigBlock ∗config, ADCalMode calMode)
- AIORET_TYPE ADCConfigBlockGetCalMode (const ADCConfigBlock ∗config)
- char ∗ ADCConfigBlockToYAML (ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetStartChannel (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetEndChannel (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockGetOversample (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetOversample (ADCConfigBlock ∗config, unsigned overSample)
- AIORET_TYPE ADCConfigBlockGetTimeout (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetTimeout (ADCConfigBlock ∗config, unsigned timeout)
- AIORET_TYPE ADCConfigBlockGetTriggerMode (const ADCConfigBlock ∗config)
- AIORET_TYPE ADCConfigBlockSetTriggerMode (ADCConfigBlock ∗config, unsigned triggerMode)
- AIORET_TYPE ADCConfigBlockSetReference (ADCConfigBlock ∗config, int ref)

  Sets the Timer reference.
- AIORET_TYPE ADCConfigBlockSetTriggerEdge (ADCConfigBlock ∗config, AIOUSB_BOOL val)
- AIORET_TYPE ADCConfigBlockSetDifferentialMode (ADCConfigBlock ∗config, unsigned channel, AIOUSB_BOOL differentialMode)

- AIORET_TYPE ADCConfigBlockSetRangeSingle (ADCConfigBlock ∗config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock ∗to, ADCConfigBlock ∗from)
- AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock ∗obj, AIOUSBDevice ∗dev)
- AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock ∗obj, AIOUSBDevice ∗dev)
- AIOUSBDevice ∗ ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock ∗obj, AIORET_TYPE ∗res)
- AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock ∗config, AIOUSBDevice ∗dev)

  initializes an ADCConfigBlock using parameters from the AIOUSBDevice
- AIORET_TYPE ADCConfigBlockSetTesting (ADCConfigBlock ∗obj, AIOUSB_BOOL testing)
- AIORET_TYPE ADCConfigBlockGetTesting (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockSetSize (ADCConfigBlock ∗obj, unsigned size)
- AIORET_TYPE ADCConfigBlockGetSize (const ADCConfigBlock ∗obj)
- AIORET_TYPE ADCConfigBlockSetDebug (ADCConfigBlock ∗obj, AIOUSB_BOOL debug)
- AIORET_TYPE ADCConfigBlockGetDebug (const ADCConfigBlock ∗obj)
- char ∗ ADCConfigBlockToJSON (ADCConfigBlock ∗config)

  INTERNAL_DOCUMENTATION.
- ADCConfigBlock ∗ NewADCConfigBlockFromJSON (const char ∗str)
- AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock ∗config)
- AIOUSB_BOOL is_all_digits (char ∗str)

24.74.1 Typedef Documentation

typedef struct AIOUSBDevice AIOUSBDevice

typedef struct mux_settings ADCMuxSettings

typedef struct ADCConfigBlock ADCConfigBlock

typedef ADCConfigBlock ADCConfigBlock

24.74.2 Function Documentation

AIORET_TYPE ADCConfigBlockInit (ADCConfigBlock ∗ *config,* AIOUSBDevice ∗ *deviceDesc,* unsigned *size*)

Parameters

config	
deviceDesc	
size	

AIORET_TYPE ADCConfigBlockInitForCounterScan (ADCConfigBlock ∗ *config,* AIOUSBDevice ∗ *deviceDesc*)

Parameters

config	
deviceDesc	

Returns

AIORET_TYPE ADCConfigBlockInitializeDefault (ADCConfigBlock ∗ *config*)

void ADC_VerifyAndCorrectConfigBlock (ADCConfigBlock ∗ *configBlock,* AIOUSBDevice ∗ *deviceDesc*)

AIORET_TYPE ADCConfigBlockSetAllGainCodeAndDiffMode (ADCConfigBlock ∗ *config,* unsigned *gainCode,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE ADCConfigBlockSetRegister (ADCConfigBlock ∗ *config,* unsigned *reg,* unsigned char *value*)

AIORET_TYPE ADCConfigBlockGetGainCode (const ADCConfigBlock ∗ *config,* unsigned *channel*)

AIORET_TYPE ADCConfigBlockSetGainCode (ADCConfigBlock ∗ *config,* unsigned *channel,* unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockSetClockRate (ADCConfigBlock ∗ *config,* int *clock_rate*)

AIORET_TYPE ADCConfigBlockGetClockRate (ADCConfigBlock ∗ ***config*** **)**

AIORET_TYPE ADCConfigBlockSetScanRange (ADCConfigBlock ∗ ***config,*** **unsigned** ***startChannel,*** **unsigned** ***endChannel*** **)**

this board doesn't have a MUX, so support base number of channels

AIORET_TYPE ADCConfigBlockSetStartChannel (ADCConfigBlock ∗ ***config,*** **unsigned char** ***startChannel*** **)**

See Also

USB_SOFTWARE_MANUAL

AIORET_TYPE ADCConfigBlockSetEndChannel (ADCConfigBlock ∗ ***config,*** **unsigned char** ***endChannel*** **)**

AIORET_TYPE ADCConfigBlockSetChannelRange (ADCConfigBlock ∗ ***config,*** **unsigned** ***startChannel,*** **unsigned** ***endChannel,*** **unsigned** ***gainCode*** **)**

INTERNAL_DOCUMENTATION.

AIORET_TYPE ADCConfigBlockSetCalMode (ADCConfigBlock ∗ ***config,*** **ADCalMode** ***calMode*** **)**

AIORET_TYPE ADCConfigBlockGetCalMode (const ADCConfigBlock ∗ ***config*** **)**

char∗ **ADCConfigBlockToYAML (ADCConfigBlock** ∗ ***config*** **)**

```
* ---
* adcconfig:
*   channels:
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   calibration: Normal
*   trigger:
*     edge: falling edge
*     refchannel: all-channels
*     reference: external
*   oversample: 201
*   clockrate: 1000
*
```

AIORET_TYPE ADCConfigBlockGetStartChannel (const ADCConfigBlock ∗ ***config*** **)**

AIORET_TYPE ADCConfigBlockGetEndChannel (const ADCConfigBlock ∗ ***config*** **)**

AIORET_TYPE ADCConfigBlockGetOversample (const ADCConfigBlock ∗ ***config*** **)**

AIORET_TYPE ADCConfigBlockSetOversample (ADCConfigBlock ∗ ***config,*** **unsigned** ***overSample*** **)**

AIORET_TYPE ADCConfigBlockGetTimeout (const ADCConfigBlock ∗ ***config*** **)**

AIORET_TYPE ADCConfigBlockSetTimeout (ADCConfigBlock ∗ ***config,*** **unsigned** ***timeout*** **)**

AIORET_TYPE ADCConfigBlockGetTriggerMode (const ADCConfigBlock ∗ ***config*** **)**

AIORET_TYPE ADCConfigBlockSetTriggerMode (ADCConfigBlock ∗ ***config,*** **unsigned** ***triggerMode*** **)**

AIORET_TYPE ADCConfigBlockSetReference (ADCConfigBlock ∗ ***config,*** **int** ***ref*** **)**

Sets the Timer reference.

AIORET_TYPE ADCConfigBlockSetTriggerEdge (ADCConfigBlock ∗ *config*, AIOUSB_BOOL *val*)

AIORET_TYPE ADCConfigBlockSetDifferentialMode (ADCConfigBlock ∗ *config*, unsigned *channel*, AIOUSB_BOOL *differentialMode*)

AIORET_TYPE ADCConfigBlockSetRangeSingle (ADCConfigBlock ∗ *config*, unsigned long *channel*, unsigned char *gainCode*)

AIORET_TYPE ADCConfigBlockCopy (ADCConfigBlock ∗ *to*, ADCConfigBlock ∗ *from*)

AIORET_TYPE ADCConfigBlockSetDevice (ADCConfigBlock ∗ *obj*, AIOUSBDevice ∗ *dev*)

AIORET_TYPE ADCConfigBlockSetAIOUSBDevice (ADCConfigBlock ∗ *obj*, AIOUSBDevice ∗ *dev*)

AIOUSBDevice∗ ADCConfigBlockGetAIOUSBDevice (ADCConfigBlock ∗ *obj*, AIORET_TYPE ∗ *res*)

AIORET_TYPE ADCConfigBlockInitializeFromAIOUSBDevice (ADCConfigBlock ∗ *config*, AIOUSBDevice ∗ *dev*)

initializes an ADCConfigBlock using parameters from the AIOUSBDevice

AIORET_TYPE ADCConfigBlockSetTesting (ADCConfigBlock ∗ *obj*, AIOUSB_BOOL *testing*)

AIORET_TYPE ADCConfigBlockGetTesting (const ADCConfigBlock ∗ *obj*)

AIORET_TYPE ADCConfigBlockSetSize (ADCConfigBlock ∗ *obj*, unsigned *size*)

AIORET_TYPE ADCConfigBlockGetSize (const ADCConfigBlock ∗ *obj*)

AIORET_TYPE ADCConfigBlockSetDebug (ADCConfigBlock ∗ *obj*, AIOUSB_BOOL *debug*)

AIORET_TYPE ADCConfigBlockGetDebug (const ADCConfigBlock ∗ *obj*)

char∗ ADCConfigBlockToJSON (ADCConfigBlock ∗ *config*)

INTERNAL_DOCUMENTATION.

ADCConfigBlock∗ NewADCConfigBlockFromJSON (const char ∗ *str*)

AIORET_TYPE DeleteADCConfigBlock (ADCConfigBlock ∗ *config*)

AIOUSB_BOOL is_all_digits (char ∗ *str*)

24.75 lib/AIOBuf.c File Reference

```
#include "AIOBuf.h"
#include <stdio.h>
#include <stdlib.h>
```

Functions

- AIOBuf ∗ NewAIOBuf (AIOBufType type, size_t size)
- AIORET_TYPE DeleteAIOBuf (AIOBuf ∗buf)
- AIORET_TYPE AIOBufGetSize (AIOBuf ∗buf)
- AIORET_TYPE AIOBufGetTotalSize (AIOBuf ∗buf)
- AIOBufType AIOBufGetType (AIOBuf ∗buf)
- void ∗ AIOBufGetRaw (AIOBuf ∗buf)
- AIORET_TYPE AIOBufGetTypeSize (AIOBuf ∗buf)
- AIORET_TYPE AIOBufRead (AIOBuf ∗buf, void ∗tobuf, size_t size_tobuf)
- AIORET_TYPE AIOBufWrite (AIOBuf ∗buf, void ∗frombuf, size_t size_frombuf)
- AIOBufIterator AIOBufGetIterator (AIOBuf ∗buf)
- AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator ∗biter)
- void AIOBufIteratorNext (AIOBufIterator ∗biter)
- AIOEither AIOBufIteratorGetValue (AIOBufIterator ∗biter)

    Returns a value from the current interator.

24.75.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.75.2 Function Documentation

AIOBuf∗ NewAIOBuf (**AIOBufType** *type,* size_t *size*)

AIORET_TYPE DeleteAIOBuf (**AIOBuf** ∗ *buf*)

AIORET_TYPE AIOBufGetSize (**AIOBuf** ∗ *buf*)

AIORET_TYPE AIOBufGetTotalSize (**AIOBuf** ∗ *buf*)

AIOBufType AIOBufGetType (**AIOBuf** ∗ *buf*)

void∗ AIOBufGetRaw (**AIOBuf** ∗ *buf*)

AIORET_TYPE AIOBufGetTypeSize (**AIOBuf** ∗ *buf*)

AIORET_TYPE AIOBufRead (**AIOBuf** ∗ *buf,* void ∗ *tobuf,* size_t *size_tobuf*)

AIORET_TYPE AIOBufWrite (**AIOBuf** ∗ *buf,* void ∗ *frombuf,* size_t *size_frombuf*)

AIOBufIterator AIOBufGetIterator (**AIOBuf** ∗ *buf*)

AIOUSB_BOOL AIOBufIteratorIsValid (**AIOBufIterator** ∗ *biter*)

void AIOBufIteratorNext (**AIOBufIterator** ∗ *biter*)

AIOEither AIOBufIteratorGetValue (**AIOBufIterator** ∗ *biter*)

Returns a value from the current interator.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

24.76 lib/AIOBuf.h File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
```

Data Structures

- struct AIOBuf
- struct aiobuf_iterator

Typedefs

- typedef struct AIOBuf AIOBuf
- typedef struct aiobuf_iterator AIOBufIterator

Enumerations

- enum AIOBufType { AIO_ERROR_BUF = -1, AIO_DEFAULT_BUF = 1, AIO_COUNTS_BUF = 2, AIO_VOLTS_-BUF = 8 }

Functions

- AIOBuf ∗ NewAIOBuf (AIOBufType type, size_t size)
- AIORET_TYPE DeleteAIOBuf (AIOBuf ∗type)
- AIORET_TYPE AIOBufGetSize (AIOBuf ∗buf)
- AIOBufType AIOBufGetType (AIOBuf ∗buf)
- void ∗ AIOBufGetRaw (AIOBuf ∗buf)
- AIORET_TYPE AIOBufRead (AIOBuf ∗buf, void ∗tobuf, size_t size_tobuf)
- AIORET_TYPE AIOBufWrite (AIOBuf ∗buf, void ∗frombuf, size_t size_frombuf)
- AIOBufIterator AIOBufGetIterator (AIOBuf ∗buf)
- AIOEither AIOBufIteratorGetValue (AIOBufIterator ∗biter)

  Returns a value from the current interator.
- AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator ∗biter)
- void AIOBufIteratorNext (AIOBufIterator ∗biter)

24.76.1 Typedef Documentation

typedef struct AIOBuf AIOBuf

typedef struct aiobuf_iterator AIOBufIterator

24.76.2 Enumeration Type Documentation

enum AIOBufType

Enumerator

AIO_ERROR_BUF

AIO_DEFAULT_BUF

AIO_COUNTS_BUF

AIO_VOLTS_BUF

24.76.3 Function Documentation

AIOBuf∗ NewAIOBuf (AIOBufType *type,* size_t *size*)

AIORET_TYPE DeleteAIOBuf (AIOBuf ∗ *type*)

AIORET_TYPE AIOBufGetSize (AIOBuf ∗ *buf*)

AIOBufType AIOBufGetType (AIOBuf ∗ *buf*)

void∗ AIOBufGetRaw (AIOBuf ∗ *buf*)

AIORET_TYPE AIOBufRead (AIOBuf ∗ *buf,* void ∗ *tobuf,* size_t *size_tobuf*)

AIORET_TYPE AIOBufWrite (AIOBuf ∗ *buf,* void ∗ *frombuf,* size_t *size_frombuf*)

AIOBufIterator AIOBufGetIterator (AIOBuf ∗ *buf*)

AIOEither AIOBufIteratorGetValue (AIOBufIterator ∗ *biter*)

Returns a value from the current interator.

Casts up to the largest number we have and then a user can Cast down to the number they wish to actually use.

Todo make this better instead of using memcpy, just cast directly

Parameters

biter	Iterator

Returns

AIO_NUMBER large precision number.

AIOUSB_BOOL AIOBufIteratorIsValid (AIOBufIterator ∗ *biter*)

void AIOBufIteratorNext (AIOBufIterator ∗ *biter*)

24.77 lib/AIOChannelMask.c File Reference

```
#include "AIOChannelMask.h"
#include "AIOTypes.h"
#include <stdio.h>
#include <string.h>
```

Functions

- AIOChannelMask ∗ NewAIOChannelMask (unsigned number_channels)
  Constructor AIOChannelMask bit mask object.
- void DeleteAIOChannelMask (AIOChannelMask ∗mask)
  Destructor for the AIOChannelMask object.
- AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗mask, int ∗pos)
  Returns an interator to the indices that are valid high (1).
- AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗mask, int ∗pos)
  Part of the iterator pair of functions for finding the indices where the mask has a 1.
- AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗obj, unsigned field)
  Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.
- AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗obj, char field, unsigned index)
  Sets the Bit Mask at specified index to the values contained in field.
- AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗obj, char ∗tmp, unsigned index)
  Retrieves the mask at offset index, and saves it to tmp.
- AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗obj)
  Gives the size of the given BitMask.
- AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗obj)
  Returns channels that are set to High (not low) ∗.
- AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗obj)
- AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask ∗obj, const char ∗bitfields)
  Rely on the base type to determine the sizes.
- AIOChannelMask ∗ NewAIOChannelMaskFromStr (const char ∗bitfields)
  Creates a new AIOChannelMask object from a character string of 1's and 0's.
- AIOChannelMask ∗ NewAIOChannelMaskFromChr (const char bits)
- char ∗ AIOChannelMaskToString (AIOChannelMask ∗obj)
  Returns a string representation for the AIOChannel Bit mask in question.
- char ∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗obj, unsigned index)
  Returns a mask for the index in question.
- char ∗ AIOChannelMaskGetMask (AIOChannelMask ∗obj)

24.77.1 Function Documentation

AIOChannelMask∗ NewAIOChannelMask (unsigned *number_channels*)

Constructor AIOChannelMask bit mask object.

Parameters

number_-channels	The number of bits in our Bit Mask

void DeleteAIOChannelMask (AIOChannelMask ∗ *mask*)

Destructor for the AIOChannelMask object.

Parameters

mask	AIOChannelMask to delete

AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗ *mask*, int ∗ *pos*)

Returns an interator to the indices that are valid high (1).

AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗ *mask*, int ∗ *pos*)

Part of the iterator pair of functions for finding the indices where the mask has a 1.

AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗ *obj*, unsigned *field*)

Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.

AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗ *obj*, char *field*, unsigned *index*)

Sets the Bit Mask at specified index to the values contained in field.

AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗ *obj*, char ∗ *tmp*, unsigned *index*)

Retrieves the mask at offset index, and saves it to tmp.

Parameters

obj	The AIOChannelMask bit mask object
∗*tmp*	The object we save the BitMask to
index	into the AIOChannelMask that we wish to retrieve the bitmask for

AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗ *obj*)

Gives the size of the given BitMask.

AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗ *obj*)

Returns channels that are set to High (not low) ∗.

Parameters

obj	

Returns

AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗ *obj*)

AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask ∗ *obj*, const char ∗ *bitfields*)

Rely on the base type to determine the sizes.

Parameters

obj	
bitfields	a character string that contains 0s and 1s.

AIOChannelMask∗ **NewAIOChannelMaskFromStr (const char** ∗ ***bitfields*** **)**

Creates a new AIOChannelMask object from a character string of 1's and 0's.

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

AIOChannelMask∗ **NewAIOChannelMaskFromChr (const char *bits*)**

char∗ **AIOChannelMaskToString (AIOChannelMask** ∗ ***obj*** **)**

Returns a string representation for the AIOChannel Bit mask in question.

Parameters

obj	AIOChannelMask to convert to string form

Todo Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

char∗ **AIOChannelMaskToStringAtIndex (AIOChannelMask** ∗ ***obj,*** **unsigned *index*** **)**

Returns a mask for the index in question.

Parameters

obj	AIOChannelMask bit mask object
index	into byte array that we wish to return a byte worth of bits from

Note

Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

char∗ **AIOChannelMaskGetMask (AIOChannelMask** ∗ ***obj*** **)**

24.78 lib/AIOChannelMask.h File Reference

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <assert.h>
#include <unistd.h>
```

Data Structures

- struct AIOChannelMask

Macros

- #define BIT_LENGTH(x) (sizeof(x) ∗ 8)

Typedefs

- typedef char aio_channel_obj

Functions

- AIOChannelMask ∗ NewAIOChannelMask (unsigned size)

    Constructor AIOChannelMask bit mask object.
- void DeleteAIOChannelMask (AIOChannelMask ∗mask)

    Destructor for the AIOChannelMask object.
- AIOChannelMask ∗ NewAIOChannelMaskFromStr (const char ∗bitfields)

    Creates a new AIOChannelMask object from a character string of 1's and 0's.
- AIOChannelMask ∗ NewAIOChannelMaskFromChr (const char bits)
- char ∗ AIOChannelMaskToString (AIOChannelMask ∗mask)

    Returns a string representation for the AIOChannel Bit mask in question.
- char ∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗obj, unsigned index)

    Returns a mask for the index in question.
- char ∗ AIOChannelMaskGetMask (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗mask, char ∗val, unsigned index)

    Retrieves the mask at offset index, and saves it to tmp.
- AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗mask)

    Returns channels that are set to High (not low) ∗.
- AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗mask)
- AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗mask)

    Gives the size of the given BitMask.
- AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗mask, int ∗pos)

    Returns an interator to the indices that are valid high (1).
- AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗mask, int ∗pos)

    Part of the iterator pair of functions for finding the indices where the mask has a 1.
- AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗mask, unsigned field)

    Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.
- AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗mask, char field, unsigned index)

    Sets the Bit Mask at specified index to the values contained in field.
- AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask ∗mask, const char ∗bitfields)

    Rely on the base type to determine the sizes.

24.78.1 Macro Definition Documentation

#define BIT_LENGTH(*x*) (sizeof(x) ∗ 8)

24.78.2 Typedef Documentation

typedef char aio_channel_obj

24.78.3 Function Documentation

AIOChannelMask∗ NewAIOChannelMask (unsigned *number_channels*)

Constructor AIOChannelMask bit mask object.

Parameters

number_-channels	The number of bits in our Bit Mask

void DeleteAIOChannelMask (AIOChannelMask ∗ *mask*)

Destructor for the AIOChannelMask object.

Parameters

mask	AIOChannelMask to delete

AIOChannelMask∗ NewAIOChannelMaskFromStr (const char ∗ *bitfields*)

Creates a new AIOChannelMask object from a character string of 1's and 0's.

Parameters

bitfields	

Returns

a new AIOChannelMask object

Todo Add smarter error checking

AIOChannelMask∗ NewAIOChannelMaskFromChr (const char *bits*)

char∗ AIOChannelMaskToString (AIOChannelMask ∗ *obj*)

Returns a string representation for the AIOChannel Bit mask in question.

Parameters

obj	AIOChannelMask to convert to string form

Todo Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

char∗ AIOChannelMaskToStringAtIndex (AIOChannelMask ∗ *obj,* unsigned *index*)

Returns a mask for the index in question.

Parameters

obj	AIOChannelMask bit mask object
index	into byte array that we wish to return a byte worth of bits from

Note

Check for the case where we have say 17 signals(non-integer multiple of BITS_PER_BYTE

char∗ AIOChannelMaskGetMask (AIOChannelMask ∗ *mask*)

AIORET_TYPE AIOChannelMaskGetMaskAtIndex (AIOChannelMask ∗ *obj,* char ∗ *tmp,* unsigned *index*)

Retrieves the mask at offset index, and saves it to tmp.

Parameters

obj	The AIOChannelMask bit mask object
∗tmp	The object we save the BitMask to
index	into the AIOChannelMask that we wish to retrieve the bitmask for

AIORET_TYPE AIOChannelMaskNumberChannels (AIOChannelMask ∗ *obj*)

Returns channels that are set to High (not low) ∗.

Parameters

obj	

Returns

AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask ∗ *mask*)

AIORET_TYPE AIOChannelMaskGetSize (AIOChannelMask ∗ *mask*)

Gives the size of the given BitMask.

AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask ∗ *mask,* int ∗ *pos*)

Returns an interator to the indices that are valid high (1).

AIORET_TYPE AIOChannelMaskNextIndex (AIOChannelMask ∗ *mask,* int ∗ *pos*)

Part of the iterator pair of functions for finding the indices where the mask has a 1.

AIORET_TYPE AIOChannelMaskSetMaskFromInt (AIOChannelMask ∗ *obj,* unsigned *field*)

Sets the AIOChannelMask using the regular notion of or'ing of shifted bytes,.

AIORET_TYPE AIOChannelMaskSetMaskAtIndex (AIOChannelMask ∗ *mask,* char *field,* unsigned *index*)

Sets the Bit Mask at specified index to the values contained in field.

AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask ∗ *obj,* const char ∗ *bitfields*)

Rely on the base type to determine the sizes.

Parameters

obj	
bitfields	a character string that contains 0s and 1s.

24.79 lib/AIOChannelRange.c File Reference

```
#include "AIOChannelRange.h"
```

Enumerations

- enum STATE {
  BEGIN, START_CHANNEL, END_CHANNEL, GAIN,
  END }

Functions

- AIOChannelRange ∗ NewAIOChannelRangeFromStr (const char ∗str)
- void DeleteAIOChannelRange (AIOChannelRange ∗range)
- const char ∗ lookup_voltage_range (ADGainCode code)
- char ∗ AIOChannelRangeToStr (AIOChannelRange ∗range)
- AIORET_TYPE AIOChannelRangeGetStart (AIOChannelRange ∗range)
- AIORET_TYPE AIOChannelRangeGetEnd (AIOChannelRange ∗range)
- AIORET_TYPE AIOChannelRangeGetGain (AIOChannelRange ∗range)

Variables

- int aio_channel_range_error = 0

24.79.1 Enumeration Type Documentation

enum STATE

Enumerator

BEGIN

START_CHANNEL

END_CHANNEL

GAIN

END

24.79.2 Function Documentation

AIOChannelRange∗ NewAIOChannelRangeFromStr (const char ∗ *str*)

void DeleteAIOChannelRange (AIOChannelRange ∗ *range*)

const char∗ lookup_voltage_range (ADGainCode *code*)

char∗ AIOChannelRangeToStr (AIOChannelRange ∗ *range*)

AIORET_TYPE AIOChannelRangeGetStart (AIOChannelRange ∗ *range*)

AIORET_TYPE AIOChannelRangeGetEnd (AIOChannelRange ∗ *range*)

AIORET_TYPE AIOChannelRangeGetGain (AIOChannelRange ∗ *range*)

24.79.3 Variable Documentation

int aio_channel_range_error = 0

24.80 lib/AIOChannelRange.h File Reference

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
```

Data Structures

- struct ad_gain_pairs
- struct aio_channel_range

Macros

- #define LENGTH_AD_GAIN_CODE_STRINGS ((int)(sizeof(AD_GAIN_CODE_STRINGS)/sizeof(struct ad_gain-_pairs) - 1))

Typedefs

- typedef struct aio_channel_range AIOChannelRange

Functions

- AIOChannelRange ∗ NewAIOChannelRangeFromStr (const char ∗str)
- void DeleteAIOChannelRange (AIOChannelRange ∗range)
- char ∗ AIOChannelRangeToStr (AIOChannelRange ∗range)
- AIORET_TYPE AIOChannelRangeGetStart (AIOChannelRange ∗range)
- AIORET_TYPE AIOChannelRangeGetEnd (AIOChannelRange ∗range)
- AIORET_TYPE AIOChannelRangeGetGain (AIOChannelRange ∗range)

Variables

- struct ad_gain_pairs AD_GAIN_CODE_STRINGS []
- int aio_channel_range_error

24.80.1 Macro Definition Documentation

#define LENGTH_AD_GAIN_CODE_STRINGS ((int)(sizeof(AD_GAIN_CODE_STRINGS)/sizeof(struct ad_gain_pairs) - 1))

24.80.2 Typedef Documentation

typedef struct aio_channel_range AIOChannelRange

24.80.3 Function Documentation

AIOChannelRange∗ NewAIOChannelRangeFromStr (const char ∗ *str*)

void DeleteAIOChannelRange (**AIOChannelRange** ∗ *range*)

char∗ AIOChannelRangeToStr (**AIOChannelRange** ∗ *range*)

AIORET_TYPE AIOChannelRangeGetStart (**AIOChannelRange** ∗ *range*)

AIORET_TYPE AIOChannelRangeGetEnd (**AIOChannelRange** ∗ *range*)

AIORET_TYPE AIOChannelRangeGetGain (**AIOChannelRange** ∗ *range*)

24.80.4 Variable Documentation

struct **ad_gain_pairs** AD_GAIN_CODE_STRINGS[ ]

Initial value:

```
= {
    {AD_GAIN_CODE_0_10V, "0-10"},
    {AD_GAIN_CODE_10V, "+-10"},
    {AD_GAIN_CODE_0_5V,"0-5"},
    {AD_GAIN_CODE_5V,"+-5"},
    {AD_GAIN_CODE_0_2V,"0-2"},
    {AD_GAIN_CODE_2V,"+-2"},
    {AD_GAIN_CODE_0_1V,"0-1"},
    {AD_GAIN_CODE_1V,"+-1"},
    {ADGainCode_end,0}
}
```

int aio_channel_range_error

24.81 lib/AIOCmd.c File Reference

```
#include "AIOTypes.h"
#include "AIOCmd.h"
#include "AIOEither.h"
```

Functions

- AIOCmd ∗ NewAIOCmdFromJSON (const char ∗str)
- AIOCmd ∗ NewAIOCmd ()
- AIORET_TYPE DeleteAIOCmd (AIOCmd ∗cmd)

24.81.1 Function Documentation

AIOCmd∗ NewAIOCmdFromJSON (const char ∗ *str*)

AIOCmd∗ NewAIOCmd ( )

AIORET_TYPE DeleteAIOCmd (**AIOCmd** ∗ *cmd*)

24.82 lib/AIOCmd.h File Reference

General structure for processing AIOUSB commands.

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

Data Structures

- struct AIOCmd

Typedefs

- typedef struct AIOCmd AIOCmd

Functions

- AIOCmd ∗ NewAIOCmdFromJSON (const char ∗str)
- AIOCmd ∗ NewAIOCmd ()
- AIORET_TYPE DeleteAIOCmd (AIOCmd ∗cmd)

24.82.1 Detailed Description

General structure for processing AIOUSB commands.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.82.2 Typedef Documentation

typedef struct AIOCmd AIOCmd

24.82.3 Function Documentation

AIOCmd∗ NewAIOCmdFromJSON (const char ∗ *str*)

AIOCmd∗ NewAIOCmd ( )

AIORET_TYPE DeleteAIOCmd (AIOCmd ∗ *cmd*)

24.83 lib/AIOCommandLine.c File Reference

```
#include "AIOCommandLine.h"
#include "AIOList.h"
```

Functions

- AIOCommandLineOptions ∗ AIO_CMDLINE_DEFAULT_OPTIONS ()
- AIOCommandLineOptions ∗ AIO_CMDLINE_SCRIPTING_OPTIONS ()
- AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions ∗opts)
- AIORET_TYPE AIOProcessCommandLine (AIOCommandLineOptions ∗options, int ∗argc, char ∗∗argv)
- AIORET_TYPE AIOProcessCmdline (AIOCommandLineOptions ∗options, int argc, char ∗∗argv)

  A simplified command line parsing library for.
- void AIOPrintUsage (int argc, char ∗∗argv, struct option ∗options)

  Shows the user the various options that this library is capable of parsing on the command line.
- AIOCommandLineOptions ∗ NewDefaultAIOCommandLineOptions ()

  Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

- AIOCommandLineOptions ∗ NewAIOCommandLineOptionsFromDefaultOptions (const AIOCommandLineOptions ∗orig)
- AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions ∗options)

  A Descructor for the allocated AIOCommandLineOptions.
- AIORET_TYPE AIOCommandLineListDevices (AIOCommandLineOptions ∗options, int ∗indices, int num_devices)
- AIORET_TYPE AIOCommandLineOptionsListDevices (AIOCommandLineOptions ∗options, intlist ∗indices)

  Lists any devices that were matched and then lists which index was specified.
- AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock ∗config, AIOCommandLineOptions ∗options)
- AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock ∗config, AIOCommandLineOptions ∗options)

  Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.
- AIOChannelRangeTmp ∗ AIOGetChannelRange (char ∗optarg)
- const AIOCommandLineOptions ∗ AIO_SCRIPTING_OPTIONS (void)
- const AIOCommandLineOptions ∗ AIO_CMDLINE_OPTIONS (void)

Variables

- int opterr
- int optind
- AIOCommandLineOptions AIO_DEFAULT_CMDLINE_OPTIONS

  The default settings for running various samples.
- AIOCommandLineOptions AIO_DEFAULT_SCRIPTING_OPTIONS

24.83.1 Function Documentation

AIOCommandLineOptions∗ AIO_CMDLINE_DEFAULT_OPTIONS ( )

AIOCommandLineOptions∗ AIO_CMDLINE_SCRIPTING_OPTIONS ( )

AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions ∗ *opts*)

AIORET_TYPE AIOProcessCommandLine (AIOCommandLineOptions ∗ *options,* int ∗ *argc,* char ∗∗ *argv*)

Parameters

options	
argc	Pointer to number of arguments in argv.
argv	An array of strings

Returns

AIORET_TYPE AIOProcessCmdline (AIOCommandLineOptions ∗ *options,* int *argc,* char ∗∗ *argv*)

A simplified command line parsing library for.

Parameters

options	AIOCommandLineOptions object that holds overridden cmd line options
argc	Number of command line arguments
argv	Array of strings to the command line arguments

Returns

void AIOPrintUsage (int *argc,* char ∗∗ *argv,* struct option ∗ *options*)

Shows the user the various options that this library is capable of parsing on the command line.

Parameters

argc	Number of command line arguments
argv	Array of strings to the command line arguments
options	

AIOCommandLineOptions∗ **NewDefaultAIOCommandLineOptions ( )**

Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

Returns

AIOCommandLineOptions ∗ a new object containing the default settings

AIOCommandLineOptions∗ **NewAIOCommandLineOptionsFromDefaultOptions (const AIOCommandLineOptions** ∗ ***orig*** **)**

AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions ∗ ***options*** **)**

A Descructor for the allocated AIOCommandLineOptions.

Parameters

options	

Returns

AIORET_TYPE AIOCommandLineListDevices (AIOCommandLineOptions ∗ ***options,*** **int** ∗ ***indices,*** **int** ***num_devices*** **)**

AIORET_TYPE AIOCommandLineOptionsListDevices (AIOCommandLineOptions ∗ ***options,*** **intlist** ∗ ***indices*** **)**

Lists any devices that were matched and then lists which index was specified.

Parameters

options	AIOCommandLineOptions object
indices	Array of devices found
num_devices	number of devices in the array

Returns

$>=$ AIOUSB_SUCCESS if devices have been found, $<$ 0 if no devices found

AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock ∗ ***config,*** **AIOCommandLineOptions** ∗ ***options*** **)**

AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock ∗ ***config,*** **AIOCommandLineOptions** ∗ ***options*** **)**

Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.

Parameters

config	ADCConfigBlock object that is written to the AIOUSB device
options	AIOCommandLineOptions object that represents the set of user parameters specified on the command line

Returns

$>=$ AIOUSB_SUCCESS is successful, $<$ 0 if there was an error

AIOChannelRangeTmp∗ **AIOGetChannelRange (char** ∗ ***optarg*** **)**

const AIOCommandLineOptions∗ **AIO_SCRIPTING_OPTIONS (void)**

const AIOCommandLineOptions∗ **AIO_CMDLINE_OPTIONS (void)**

24.83.2 Variable Documentation

int opterr

int optind

AIOCommandLineOptions AIO_DEFAULT_CMDLINE_OPTIONS

The default settings for running various samples.

THis makes it easier to just get a sample up and running and then tweak certain parameters for ones own needs. For instance, if the user wanted to perform an simple ADC_GetScan, s/he could just use the settings provided in the AIO_DEFAULT_CMDLINE_OPTIONS variable to get

- 16 channels per scan
- Each channel sampling at AD_GAIN_CODE_0_5V (0 to 5 volts)
- 0 Oversamples.
- 1000 ms timeout.

AIOCommandLineOptions AIO_DEFAULT_SCRIPTING_OPTIONS

24.84 lib/AIOCommandLine.h File Reference

```
#include "AIOTypes.h"
#include "ADCConfigBlock.h"
#include "AIOContinuousBuffer.h"
#include "AIOConfiguration.h"
#include "AIOUSB_Core.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Properties.h"
#include "AIOUSB_Log.h"
#include <getopt.h>
#include <ctype.h>
#include <stdlib.h>
```

Data Structures

- struct AIOChannelRangeTmp
- struct AIOCommandLineOptions

Macros

- #define DUMP 0x1000
- #define CNTS 0x1001
- #define JCONF 0x1002
- #define REPEAT 0x1003

Typedefs

- typedef struct AIOChannelRangeTmp AIOChannelRangeTmp
- typedef struct
  AIOCommandLineOptions AIOCommandLineOptions

Enumerations

- enum DeviceEnum {
  INDEX_NUM = 0, ADCCONFIG_OPT, TIMEOUT_OPT, DEBUG_OPT,
  SETCAL_OPT, COUNT_OPT, SAMPLE_OPT, FILE_OPT,
  CHANNEL_OPT }

Functions

- AIOCommandLineOptions ∗ NewDefaultAIOCommandLineOptions ()

    Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.
- AIOCommandLineOptions ∗ NewAIOCommandLineOptionsFromDefaultOptions (const AIOCommandLineOptions ∗orig)
- AIOCommandLineOptions ∗ AIO_CMDLINE_DEFAULT_OPTIONS ()
- AIOCommandLineOptions ∗ AIO_CMDLINE_SCRIPTING_OPTIONS ()
- AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions ∗opts)
- AIORET_TYPE AIOProcessCmdline (AIOCommandLineOptions ∗options, int argc, char ∗∗argv)

    A simplified command line parsing library for.
- AIORET_TYPE AIOProcessCommandLine (AIOCommandLineOptions ∗options, int ∗argc, char ∗∗argv)
- AIOChannelRangeTmp ∗ AIOGetChannelRange (char ∗optarg)
- void AIOPrintUsage (int argc, char ∗∗argv, struct option ∗options)

    Shows the user the various options that this library is capable of parsing on the command line.
- AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions ∗options)

    A Descructor for the allocated AIOCommandLineOptions.
- AIORET_TYPE AIOCommandLineOptionsListDevices (AIOCommandLineOptions ∗options, intlist ∗indices)

    Lists any devices that were matched and then lists which index was specified.
- AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock ∗config, AIOCommandLineOptions ∗options)

    Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.
- const AIOCommandLineOptions ∗ AIO_SCRIPTING_OPTIONS (void)
- const AIOCommandLineOptions ∗ AIO_CMDLINE_OPTIONS (void)
- AIORET_TYPE AIOCommandLineListDevices (AIOCommandLineOptions ∗options, int ∗indices, int num_devices) ACCES_DEPRECATED("Please use AIOCommandLineOptionsListDevices")
- AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock ∗config, AIOCommandLineOptions ∗options) ACCES_DEPRECATED("Please use AIOCommandLineOptionsOverrideADCConfigBlock")

Variables

- AIOCommandLineOptions AIO_DEFAULT_CMDLINE_OPTIONS

    The default settings for running various samples.
- AIOCommandLineOptions AIO_DEFAULT_SCRIPTING_OPTIONS

24.84.1 Macro Definition Documentation

#define DUMP 0x1000

#define CNTS 0x1001

#define JCONF 0x1002

#define REPEAT 0x1003

24.84.2 Typedef Documentation

typedef struct AIOChannelRangeTmp AIOChannelRangeTmp

typedef struct AIOCommandLineOptions AIOCommandLineOptions

24.84.3 Enumeration Type Documentation

enum DeviceEnum

Enumerator

INDEX_NUM

ADCCONFIG_OPT

TIMEOUT_OPT

DEBUG_OPT

SETCAL_OPT

COUNT_OPT

SAMPLE_OPT

FILE_OPT

CHANNEL_OPT

24.84.4 Function Documentation

AIOCommandLineOptions∗ NewDefaultAIOCommandLineOptions ( )

Creates a new command line option object for performing comparisons with the default settings for AIOUSB devices.

Returns

AIOCommandLineOptions ∗ a new object containing the default settings

AIOCommandLineOptions∗ NewAIOCommandLineOptionsFromDefaultOptions (const AIOCommandLineOptions ∗ *orig*)

AIOCommandLineOptions∗ AIO_CMDLINE_DEFAULT_OPTIONS ( )

AIOCommandLineOptions∗ AIO_CMDLINE_SCRIPTING_OPTIONS ( )

AIORET_TYPE AIO_CMDLINE_CLEAR_OPTIONS (AIOCommandLineOptions ∗ *opts*)

AIORET_TYPE AIOProcessCmdline (AIOCommandLineOptions ∗ *options,* int *argc,* char ∗∗ *argv*)

A simplified command line parsing library for.

Parameters

options	AIOCommandLineOptions object that holds overridden cmd line options
argc	Number of command line arguments
argv	Array of strings to the command line arguments

Returns

AIORET_TYPE AIOProcessCommandLine (AIOCommandLineOptions ∗ *options,* int ∗ *argc,* char ∗∗ *argv*)

Parameters

options	
argc	Pointer to number of arguments in argv.
argv	An array of strings

Returns

AIOChannelRangeTmp∗ AIOGetChannelRange (char ∗ *optarg*)

void AIOPrintUsage (int *argc,* char ∗∗ *argv,* struct option ∗ *options*)

Shows the user the various options that this library is capable of parsing on the command line.

Parameters

argc	Number of command line arguments
argv	Array of strings to the command line arguments
options	

AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions ∗ *options*)

A Descructor for the allocated AIOCommandLineOptions.

Parameters

options	

Returns

AIORET_TYPE AIOCommandLineOptionsListDevices (AIOCommandLineOptions ∗ *options,* intlist ∗ *indices*)

Lists any devices that were matched and then lists which index was specified.

Parameters

options	AIOCommandLineOptions object
indices	Array of devices found
num_devices	number of devices in the array

Returns

$>=$ AIOUSB_SUCCESS if devices have been found, $<$ 0 if no devices found

AIORET_TYPE AIOCommandLineOptionsOverrideADCConfigBlock (ADCConfigBlock ∗ *config*, AIOCommandLineOptions ∗ *options*)

Allows the AIOCommandLineOptions options to override the settings in the ADCConfigBlock.

Parameters

config	ADCConfigBlock object that is written to the AIOUSB device
options	AIOCommandLineOptions object that represents the set of user parameters specified on the command line

Returns

$>=$ AIOUSB_SUCCESS is successful, $<$ 0 if there was an error

const AIOCommandLineOptions∗ AIO_SCRIPTING_OPTIONS (void)

const AIOCommandLineOptions∗ AIO_CMDLINE_OPTIONS (void)

AIORET_TYPE AIOCommandLineListDevices (AIOCommandLineOptions ∗ *options*, int ∗ *indices*, int *num_devices*)

AIORET_TYPE AIOCommandLineOverrideADCConfigBlock (ADCConfigBlock ∗ *config*, AIOCommandLineOptions ∗ *options*)

24.84.5 Variable Documentation

AIOCommandLineOptions AIO_DEFAULT_CMDLINE_OPTIONS

The default settings for running various samples.

THis makes it easier to just get a sample up and running and then tweak certain parameters for ones own needs. For instance, if the user wanted to perform an simple ADC_GetScan, s/he could just use the settings provided in the AIO_DEFAULT_CMDLINE_OPTIONS variable to get

- 16 channels per scan
- Each channel sampling at AD_GAIN_CODE_0_5V (0 to 5 volts)
- 0 Oversamples.
- 1000 ms timeout.

AIOCommandLineOptions AIO_DEFAULT_SCRIPTING_OPTIONS

24.85 lib/AIOConfiguration.c File Reference

```
#include "AIOConfiguration.h"
#include "AIOContinuousBuffer.h"
#include "ADCConfigBlock.h"
```

Functions

- AIOConfiguration ∗ NewAIOConfiguration ()
- void DeleteAIOConfiguration (AIOConfiguration ∗config)
- AIORET_TYPE AIOConfigurationInitialize (AIOConfiguration ∗config)
- AIORET_TYPE AIOArgumentInitialize (AIOArgument ∗arg)
- AIORET_TYPE AIOConfigurationSetTimeout (AIOConfiguration ∗config, unsigned timeout)
- AIORET_TYPE AIOConfigurationSetDebug (AIOConfiguration ∗config, AIOUSB_BOOL debug)

24.85.1 Function Documentation

AIOConfiguration∗ NewAIOConfiguration ( )

void DeleteAIOConfiguration (**AIOConfiguration** ∗ *config*)

AIORET_TYPE AIOConfigurationInitialize (**AIOConfiguration** ∗ *config*)

AIORET_TYPE AIOArgumentInitialize (**AIOArgument** ∗ *arg*)

AIORET_TYPE AIOConfigurationSetTimeout (**AIOConfiguration** ∗ *config,* unsigned *timeout*)

AIORET_TYPE AIOConfigurationSetDebug (**AIOConfiguration** ∗ *config,* **AIOUSB_BOOL** *debug*)

24.86 lib/AIOConfiguration.h File Reference

```
#include "AIOTypes.h"
#include "AIOContinuousBuffer.h"
#include "ADCConfigBlock.h"
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
```

Data Structures

- struct configuration
- struct AIOArgument
- struct AIOArguments

Typedefs

- typedef struct configuration AIOConfiguration
- typedef AIOConfiguration ADCConfiguration
- typedef AIOConfiguration AIOContinousBufConfiguration

Enumerations

- enum ConfigurationType { NO_CONFIG = -1, ADCCONIGBLOCK_CONFIG = 0, AIOCONTINUOUSBUF_CONFIG = 1 }
- enum ADCScanType {
  AD_SCAN_GETSCAN = 0, AD_SCAN_GETSCANV, AD_SCAN_GETCHANNEL, AD_SCAN_GETCHANNELV,
  AD_SCAN_CONTINUOUS, AD_SCAN_BULKACQUIRE }
- enum ADCSetCalFunction { AD_NO_SET_CAL = -1, AD_SET_CAL_AUTO = 0, AD_SET_CAL_NORMAL, AD_SET_CAL_MANUAL }

Functions

- AIORET_TYPE AIOConfigurationSetDebug (AIOConfiguration ∗config, AIOUSB_BOOL debug)
- AIORET_TYPE AIOConfigurationSetTimeout (AIOConfiguration ∗config, unsigned timeout)
- AIOConfiguration ∗ NewAIOConfiguration ()
- AIORET_TYPE AIOConfigurationInitialize (AIOConfiguration ∗config)
- AIORET_TYPE AIOArgumentInitialize (AIOArgument ∗arg)

24.86.1 Typedef Documentation

typedef struct **configuration AIOConfiguration**

typedef **AIOConfiguration ADCConfiguration**

typedef **AIOConfiguration AIOContinousBufConfiguration**

24.86.2 Enumeration Type Documentation

enum ConfigurationType

Enumerator

NO_CONFIG
ADCCONIGBLOCK_CONFIG
AIOCONTINUOUSBUF_CONFIG

enum ADCScanType

Enumerator

AD_SCAN_GETSCAN
AD_SCAN_GETSCANV
AD_SCAN_GETCHANNEL
AD_SCAN_GETCHANNELV
AD_SCAN_CONTINUOUS
AD_SCAN_BULKACQUIRE

enum ADCSetCalFunction

Enumerator

AD_NO_SET_CAL
AD_SET_CAL_AUTO
AD_SET_CAL_NORMAL
AD_SET_CAL_MANUAL

24.86.3 Function Documentation

AIORET_TYPE AIOConfigurationSetDebug (AIOConfiguration ∗ *config*, AIOUSB_BOOL *debug*)

AIORET_TYPE AIOConfigurationSetTimeout (AIOConfiguration ∗ *config*, unsigned *timeout*)

AIOConfiguration∗ NewAIOConfiguration ( )

AIORET_TYPE AIOConfigurationInitialize (AIOConfiguration ∗ *config*)

AIORET_TYPE AIOArgumentInitialize (AIOArgument ∗ *arg*)

24.87 lib/AIOContinuousBuffer.c File Reference

This file contains the required structures for performing the continuous streaming buffers that talk to ACCES USB-AI∗ cards. The functionality in this file was wrapped up to provide a more unified interface for continuous streaming of acquisition data and to provide the user with a simplified system of reads for actually getting the streaming data. The role of the continuous mode is to just create a thread in the background that handles the low level USB transactions for collecting samples. This thread will fill up a data structure known as the AIOContinuousBuf that is implemented as a fifo.

```
#include "AIOUSB_Log.h"
#include "AIOContinuousBuffer.h"
#include "AIOBuf.h"
#include "ADCConfigBlock.h"
#include "AIOChannelMask.h"
#include "AIOUSB_CTR.h"
#include "AIOUSB_Core.h"
#include "AIODeviceTable.h"
#include "AIOFifo.h"
#include "AIOCountsConverter.h"
#include "AIOCmd.h"
#include "cJSON.h"
#include <ctype.h>
```

Data Structures

- struct rangelookup

Typedefs

- typedef struct rangelookup RangeValueLookup

Functions

- void ∗ ConvertCountsToVoltsFunction (void ∗object)

  Main work function for collecting data.
- void ∗ RawCountsWorkFunction (void ∗object)
- AIORET_TYPE AIOContinuousBufForceTerminateAcqusitionOverrun (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufForceTerminateAcqusition (AIOContinuousBuf ∗buf)
- AIOContinuousBuf ∗ NewAIOContinuousBufForCounts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels)
- AIOContinuousBuf ∗ NewAIOContinuousBuf (unsigned long deviceIndex, unsigned num_channels, unsigned num_oversamples, unsigned base_size)

  Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).
- AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf ∗buf, unsigned num_channels)

  will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized
- AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf ∗buf, size_t newbase)
- AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf ∗buf)
- AIOContinuousBuf ∗ NewAIOContinuousBufForVolts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels, unsigned num_oversamples)
- AIORET_TYPE AIOContinuousBuf_InitConfiguration (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf ∗buf, void ∗frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf ∗buf, void ∗frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf ∗buf, unsigned size, ADGainCode gainCode, AIOUSB_BOOL diffMode, unsigned char os, AIOUSB_BOOL dfs)
- AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf ∗buf)

  Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.
- AIORET_TYPE AIOContinuousBuf_SendPreConfig (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf ∗buf)
- AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf ∗buf)

  Destructor for AIOContinuousBuf object.
- AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf ∗buf, unsigned blksize)
- AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf ∗buf)
- ADCConfigBlock ∗ AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_SetCallback (AIOContinuousBuf ∗buf, void ∗(∗work)(void ∗object))
- AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf ∗buf, void ∗(∗work)(void ∗object))
- AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf ∗buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_BufSizeForCounts (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf ∗buf, uint16_t new_unit_size)
- AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetSizeNumElements (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf ∗buf)
- THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf ∗buf)

  returns the number of Scans accross all channels that still remain in the buffer
- AIORET_TYPE AIOContinuousBufCountsAvailable (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetDataAvailable (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf ∗buf, unsigned short ∗read_buf, unsigned tmpbuffer_size, unsigned size)

  will read in an integer number of scan counts if there is room.

- AIORET_TYPE AIOContinuousBufGetNumberOfScansToRead (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetNumberOfScansToRead (AIOContinuousBuf ∗buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf ∗buf, unsigned short ∗read-_buf, unsigned tmpbuffer_size, int64_t num_scans)

  will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufReadSingle (AIOContinuousBuf ∗buf, AIOBuf ∗tobuf, size_t size_to_read)
- AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf ∗buf, unsigned short ∗read_-buf, unsigned read_buf_size)
- AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf ∗buf)

  Returns.
- AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf ∗buf, unsigned int hz)
- AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufStart (AIOContinuousBuf ∗buf)

  Starts the thread that acquires data from USB bus.
- AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf ∗buf, AIOChannelMask ∗mask)

  Sets the channel mask.
- AIORET_TYPE AIOContinuousBuf_NumberSignals (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_NumberChannels (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf ∗buf, AIOBufferType ∗writebuf, unsigned wrbufsize, unsigned size, AIOContinuousBufMode flag)

  Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.
- AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf ∗buf, unsigned short ∗data, unsigned data-size, unsigned size, AIOContinuousBufMode flag)
- AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf ∗buf)
- AIORET_TYPE StartStreaming (AIOContinuousBuf ∗buf)
- AIORET_TYPE SetConfig (AIOContinuousBuf ∗buf)
- AIORET_TYPE ResetCounters (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufLoadCounters (AIOContinuousBuf ∗buf, unsigned countera, unsigned coun-terb)
- AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufPreSetup (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufReadNSamples (AIOContinuousBuf ∗buf, void ∗tobuf, size_t n_to_read)
- AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf ∗buf)
- unsigned long number_to_read (AIOContinuousBuf ∗buf, AIOCmd ∗cmd)
- AIOUSB_BOOL continue_running (AIOContinuousBuf ∗buf, AIOCmd ∗cmd)
- AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf ∗buf, AIO-Cmd ∗cmd, AIORET_TYPE(∗callback)(AIOContinuousBuf ∗buf))

  Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd ∗cmd.
- AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf ∗buf)

  Setups the Automated runs for continuous mode runs.
- AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf ∗buf, AIOBufferType ∗readbuf, unsigned readbuf-size, unsigned size)

  Reads the current available amount of data from buf, into the readbuf datastructure ∗.
- AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf ∗buf, ADGainCode gainCode)
- AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_SetTesting (AIOContinuousBuf ∗buf, AIOUSB_BOOL testing)
- AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf ∗buf, AIOUSB_BOOL testing)
- AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetDefaultModeForCounterScan (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf ∗buf, AIOUSB_BOOL debug)
- AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_SetDeviceIndex (AIOContinuousBuf ∗buf, unsigned long DeviceIndex)
- AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf ∗buf, unsigned long DeviceIndex)
- AIORET_TYPE AIOContinuousBuf_SaveConfig (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSaveConfig (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_SetStartAndEndChannel (AIOContinuousBuf ∗buf, unsigned startChannel, unsigned endChannel)

- AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (AIOContinuousBuf ∗buf, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOContinuousBuf_SetChannelRangeGain (AIOContinuousBuf ∗buf, unsigned startChannel, unsigned endChannel, unsigned gainCode)
- AIORET_TYPE AIOContinuousBuf_SetChannelRange (AIOContinuousBuf ∗buf, unsigned startChannel, unsigned endChannel, unsigned gainCode)
- AIORET_TYPE AIOContinuousBufSetChannelRange (AIOContinuousBuf ∗buf, unsigned startChannel, unsigned endChannel, unsigned gainCode)
- AIORET_TYPE AIOContinuousBufSetTimeout (AIOContinuousBuf ∗buf, unsigned timeout)
- AIORET_TYPE AIOContinuousBufGetTimeout (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_SetOversample (AIOContinuousBuf ∗buf, unsigned os)
- AIORET_TYPE AIOContinuousBufSetOversample (AIOContinuousBuf ∗buf, unsigned os)
- AIORET_TYPE AIOContinuousBufSetOverSample (AIOContinuousBuf ∗buf, size_t os)
- AIORET_TYPE AIOContinuousBuf_GetOverSample (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetOversample (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBuf_SetAllGainCodeAndDiffMode (AIOContinuousBuf ∗buf, ADGainCode gain, AIOUSB_BOOL diff)
- AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (AIOContinuousBuf ∗buf, ADGainCode gain, A-IOUSB_BOOL diff)
- AIORET_TYPE AIOContinuousBuf_SetDiscardFirstSample (AIOContinuousBuf ∗buf, AIOUSB_BOOL discard)
- AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (AIOContinuousBuf ∗buf, AIOUSB_BOOL discard)
- AIORET_TYPE AIOContinuousBuf_GetDeviceIndex (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetDeviceIndex (AIOContinuousBuf ∗buf)
- cJSON ∗ GetJSONValueOrDefault (cJSON ∗config, char const ∗key, EnumStringLookup ∗lookup, size_t size)
- AIOContinuousBuf ∗ NewAIOContinuousBufFromJSON (const char ∗str)
- char ∗ AIOContinuousBufToJSON (AIOContinuousBuf ∗buf)

Variables

- EnumStringLookup TrueFalse [ ]
- RangeValueLookup BaseSizeRange [ ]

24.87.1 Detailed Description

This file contains the required structures for performing the continuous streaming buffers that talk to ACCES USB-AI∗ cards. The functionality in this file was wrapped up to provide a more unified interface for continuous streaming of acquisition data and to provide the user with a simplified system of reads for actually getting the streaming data. The role of the continuous mode is to just create a thread in the background that handles the low level USB transactions for collecting samples. This thread will fill up a data structure known as the AIOContinuousBuf that is implemented as a fifo. AIOUSB sample program.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

Todo Make the number of channels in the ContinuousBuffer match the number of channels in the config object

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

Todo Make the number of channels in the ContinuousBuffer match the number of channels in the config object

All the API functions that DO NOT begin "AIOUSB_" are standard API functions, largely documented in http-://accesio.com/MANUALS/USB%20Software%20Reference.pdf. The functions that DO begin with "-AIOUSB_" are "extended" API functions added to the Linux implementation. Source code lines in this sample program that are prefixed with the comment "/ ∗ API ∗ /" highlight calls to the AIOUSB API.

LIBUSB (http://www.libusb.org/) must be installed on the Linux box (the AIOUSB code was developed using libusb version 1.0.3). After installing libusb, it may also be necessary to set an environment variable so that the libusb and aiousb header files can be located:

```
export CPATH=/usr/local/include/libusb-1.0/:/usr/local/include/aiousb/
```

Once libusb is installed properly, and you have sourced sourceme.sh you can compile the sample program using the following command.

```
make sample AIOUSBLIBDIR=${AIO_LIB_DIR} AIOUSBCLASSLIBDIR=${AIO_CLASSLIB_DIR} DEBUG=1

* Alternatively, one can "manually" compile the sample program using the command:
*
*     g++ sample.cpp -laiousb -lusb-1.0 -o sample
*
* or, to enable debug features
*
*     g++ -ggdb sample.cpp -laiousbdbg -lusb-1.0 -o sample
*
```

24.87.2 Typedef Documentation

typedef struct rangelookup RangeValueLookup

24.87.3 Function Documentation

void ∗ ConvertCountsToVoltsFunction (void ∗ *object*)

Main work function for collecting data.

Also performs copies from the raw acquiring buffer into the AIOContinuousBuf

Parameters

object	

Returns

Todo Ensure that copying matches the actual size of the data

create temporary buffer and then Load the fifo with values

Modification, allow the count to keep going... stop if

1. count >= number we are supposed to read
2. we don't have enough space

void ∗ RawCountsWorkFunction (void ∗ *object*)

Modification, allow the count to keep going... stop if

1. count >= number we are supposed to read
2. we don't have enough space

AIORET_TYPE AIOContinuousBufForceTerminateAcqusitionOverrun (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufForceTerminateAcqusition (AIOContinuousBuf ∗ *buf*)

AIOContinuousBuf∗ NewAIOContinuousBufForCounts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels*)

AIOContinuousBuf∗ NewAIOContinuousBuf (unsigned long *deviceIndex,* unsigned *num_channels,* unsigned *num_oversamples,* unsigned *base_size*)

Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).

Parameters

deviceIndex	
num_channels	
num_-oversamples	
base_size	

Returns

AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf ∗ *buf,* unsigned *num_channels*)

will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf ∗ *buf,* size_t *newbase*)

AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf ∗ *buf*)

AIOContinuousBuf∗ NewAIOContinuousBufForVolts (unsigned long *DeviceIndex,* unsigned *scancounts,* unsigned *num_channels,* unsigned *num_oversamples*)

AIORET_TYPE AIOContinuousBuf_InitConfiguration (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf ∗ *buf,* void ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf ∗ *buf,* void ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf ∗ *buf,* unsigned *size,* ADGainCode *gainCode,* AIOUSB_BOOL *diffMode,* unsigned char *os,* AIOUSB_BOOL *dfs*)

AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf ∗ *buf*)

Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.

Parameters

buf	Our AIOContinuousBuffer

Returns

AIOUSB_SUCCESS if successful, value < 0 if not.

AIORET_TYPE AIOContinuousBuf_SendPreConfig (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf ∗ *buf*)

Destructor for AIOContinuousBuf object.

AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf ∗ *buf,* unsigned *blksize*)

AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf ∗ *buf*)

ADCConfigBlock∗ AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetCallback (AIOContinuousBuf ∗ *buf,* void ∗(∗)(void ∗object) *work*)

AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf ∗ *buf,* void ∗(∗)(void ∗object) *work*)

AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf ∗ *buf,* int64_t *num_scans*)

AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_BufSizeForCounts (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf ∗ *buf,* uint16_t *new_unit_size*)

AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf ∗ *buf*)

Todo Fix this to use condition variable

AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetSizeNumElements (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf ∗ *buf*)

THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf ∗ *buf*)

returns the number of Scans accross all channels that still remain in the buffer

AIORET_TYPE AIOContinuousBufCountsAvailable (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetDataAvailable (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf ∗ *buf,* unsigned short ∗ *read_buf,* unsigned *tmpbuffer_size,* unsigned *size*)

will read in an integer number of scan counts if there is room.

-

Parameters

buf	
read_buf	
tmpbuffer_size	
size	The size of the tmp buffer

Returns

AIORET_TYPE AIOContinuousBufGetNumberOfScansToRead (AIOContinuousBuf ∗ ***buf*** **)**

AIORET_TYPE AIOContinuousBufSetNumberOfScansToRead (AIOContinuousBuf ∗ ***buf,*** **int64_t *num_scans*)**

AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf ∗ ***buf,*** **unsigned short** ∗ ***read_buf,*** **unsigned *tmpbuffer_size,* int64_t *num_scans*)**

will read in an integer number of scan counts if there is room.

- **Parameters**

buf	
read_buf	
tmpbuffer_size	
num_scans	

Returns

AIORET_TYPE AIOContinuousBufReadSingle (AIOContinuousBuf ∗ ***buf,*** **AIOBuf** ∗ ***tobuf,*** **size_t *size_to_read*)**

AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf ∗ ***buf,*** **unsigned short** ∗ ***read_buf,*** **unsigned *read_buf_size*)**

AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf ∗ ***buf*** **)**

Returns.

Parameters

buf	

Returns

Pointer to our work function

AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf ∗ ***buf,*** **unsigned int *hz*)**

AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf ∗ ***buf*** **)**

AIORET_TYPE AIOContinuousBufStart (AIOContinuousBuf ∗ ***buf*** **)**

Starts the thread that acquires data from USB bus.

- **Parameters**

buf	

Returns

status code of start.

AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf ∗ ***buf*** **)**

AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf ∗ ***buf,*** **AIOChannelMask** ∗ ***mask*** **)**

Sets the channel mask.

Parameters

buf	
mask	

Returns

AIORET_TYPE AIOContinuousBuf_NumberSignals (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_NumberChannels (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf ∗ *buf,* AIOBufferType ∗ *writebuf,* unsigned *wrbufsize,* unsigned *size,* AIOContinuousBufMode *flag*)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

Parameters

buf	
writebuf	
wrbufsize	
size	
flag	

Returns

Status of whether the write was successful , if so returning the number of bytes written or if there was insufficient space, it returns negative error code. If the number is $>=$ 0, then this corresponds to the number of bytes that were written into the buffer.

AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf ∗ *buf,* unsigned short ∗ *data,* unsigned *datasize,* unsigned *size,* AIOContinuousBufMode *flag*)

AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE StartStreaming (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE SetConfig (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE ResetCounters (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufLoadCounters (AIOContinuousBuf ∗ *buf,* unsigned *countera,* unsigned *counterb*)

AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufPreSetup (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufReadNSamples (AIOContinuousBuf ∗ *buf,* void ∗ *tobuf,* size_t *n_to_read*)

AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf ∗ *buf*)

unsigned long number_to_read (AIOContinuousBuf ∗ *buf,* AIOCmd ∗ *cmd*)

AIOUSB_BOOL continue_running (AIOContinuousBuf ∗ *buf,* AIOCmd ∗ *cmd*)

AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf ∗ *buf*, AIOCmd ∗ *cmd*, AIORET_TYPE(∗)(AIOContinuousBuf ∗buf) *callback*)

Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd ∗cmd.

The user can specify that the callback is called after each oversample, full chanell, full scan, or N number of scans.

Parameters

buf	
cmd	
callback	

Returns

$>=$ 0 if successful, $<$ 0 if failure

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf ∗ *buf*)

Setups the Automated runs for continuous mode runs.

Parameters

buf	

Returns

Note

Setup counters see reference in `USB AIO documentation`

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf ∗ *buf*, AIOBufferType ∗ *readbuf*, unsigned *readbufsize*, unsigned *size*)

Reads the current available amount of data from buf, into the readbuf datastructure ∗.

Parameters

buf	
readbuf	
readbufsize	
size	

Returns

If number is positive, it is the number of bytes that have been read.

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf ∗ *buf*)

Parameters

buf	

Returns

AIORET_TYPE AIOContinuousBufUnlock (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSimpleSetupConfig (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gainCode*)

AIORET_TYPE AIOContinuousBufEnd (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetTesting (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE AIOContinuousBufSetTesting (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *testing*)

AIORET_TYPE AIOContinuousBufGetTesting (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDefaultModeForCounterScan (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDebug (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *debug*)

AIORET_TYPE AIOContinuousBufGetDebug (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetDeviceIndex (**AIOContinuousBuf** ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBufSetDeviceIndex (**AIOContinuousBuf** ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBuf_SaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufSaveConfig (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBuf_SetChannelRangeGain (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBuf_SetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSetChannelRange (**AIOContinuousBuf** ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSetTimeout (**AIOContinuousBuf** ∗ *buf,* unsigned *timeout*)

AIORET_TYPE AIOContinuousBufGetTimeout (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetOversample (**AIOContinuousBuf** ∗ *buf,* unsigned *os*)

AIORET_TYPE AIOContinuousBufSetOversample (**AIOContinuousBuf** ∗ *buf,* unsigned *os*)

AIORET_TYPE AIOContinuousBufSetOverSample (**AIOContinuousBuf** ∗ *buf,* size_t *os*)

AIORET_TYPE AIOContinuousBuf_GetOverSample (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetOversample (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBuf_SetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (**AIOContinuousBuf** ∗ *buf,* **ADGainCode** *gain,* **AIOUSB_BOOL** *diff*)

AIORET_TYPE AIOContinuousBuf_SetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (**AIOContinuousBuf** ∗ *buf,* **AIOUSB_BOOL** *discard*)

AIORET_TYPE AIOContinuousBuf_GetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetDeviceIndex (**AIOContinuousBuf** ∗ *buf*)

cJSON∗ **GetJSONValueOrDefault (** **cJSON** ∗ *config,* **char const** ∗ *key,* **EnumStringLookup** ∗ *lookup,* **size_t** *size* **)**

AIOContinuousBuf∗ **NewAIOContinuousBufFromJSON (** **const char** ∗ *str* **)**

char∗ **AIOContinuousBufToJSON (** **AIOContinuousBuf** ∗ *buf* **)**

24.87.4 Variable Documentation

EnumStringLookup TrueFalse[ ]

Initial value:

```
= {
    { AIOUSB_TRUE , (char *)"true"  , (char *)AIO_STRINGIFY(true)  },
    { AIOUSB_FALSE, (char *)"false" , (char *)AIO_STRINGIFY(false)  },
}
```

RangeValueLookup BaseSizeRange[ ]

Initial value:

```
= {
    { 0 , 100000000 }
}
```

24.88 lib/AIOContinuousBuffer.h File Reference

```
#include "AIOTypes.h"
#include "AIOChannelMask.h"
#include "AIOUSB_ADC.h"
#include "AIOFifo.h"
#include "AIOUSB_Core.h"
#include "AIOBuf.h"
#include "AIOCmd.h"
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <libusb.h>
#include <math.h>
```

Data Structures

- struct AIOContinuousBuf

  AIOContinuousBuf provides a buffer that is used with the AIOUSB highspeed data acquisition API.

Macros

- #define ROOTCLOCK 10000000

Typedefs

- typedef void ∗(∗ AIOUSB_WorkFn)(void ∗obj)
- typedef struct AIOContinuousBuf AIOContinuousBuf

  AIOContinuousBuf provides a buffer that is used with the AIOUSB highspeed data acquisition API.

Enumerations

- enum AIO_CONT_BUF_TYPE { AIO_CONT_BUF_TYPE_COUNTS = 2, AIO_CONT_BUF_TYPE_VOLTS = 8 }

Functions

- AIOContinuousBuf ∗ NewAIOContinuousBuf (unsigned long DeviceIndex, unsigned num_channels, unsigned num_oversamples, unsigned base_size)

    Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).
- AIOContinuousBuf ∗ NewAIOContinuousBufForCounts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels)
- AIOContinuousBuf ∗ NewAIOContinuousBufForVolts (unsigned long DeviceIndex, unsigned scancounts, unsigned num_channels, unsigned num_oversamples)
- AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf ∗buf)

    Destructor for AIOContinuousBuf object.
- AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf ∗buf)

    Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.
- AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf ∗buf, unsigned size, ADGainCode gainCode, AIOUSB_BOOL diffMode, unsigned char os, AIOUSB_BOOL dfs)
- AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf ∗buf)

    Returns.
- AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf ∗buf, void ∗(∗work)(void ∗object))
- AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf ∗buf, unsigned sblksize)
- AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf ∗buf)
- ADCConfigBlock ∗ AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf ∗buf, unsigned num_channels)

    will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized
- AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetOversample (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetOversample (AIOContinuousBuf ∗buf, unsigned num_oversamples)
- AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf ∗buf, size_t newbase)
- AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf ∗buf, uint16_t new_unit_size)
- AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf ∗buf, AIOUSB_BOOL testing)
- AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (AIOContinuousBuf ∗buf, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (AIOContinuousBuf ∗buf, ADGainCode gain, AIOUSB_BOOL diff)
- AIORET_TYPE AIOContinuousBufGetDeviceIndex (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (AIOContinuousBuf ∗buf, AIOUSB_BOOL discard)
- AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf ∗buf, AIOChannelMask ∗mask)

    Sets the channel mask.
- AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetChannelRange (AIOContinuousBuf ∗buf, unsigned startChannel, unsigned endChannel, unsigned gainCode)
- AIORET_TYPE AIOContinuousBufSaveConfig (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf ∗buf, unsigned long DeviceIndex)
- AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetTimeout (AIOContinuousBuf ∗buf, unsigned timeout)
- AIORET_TYPE AIOContinuousBufGetTimeout (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf ∗buf, AIOUSB_BOOL debug)
- AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf ∗buf, int64_t num_scans)
- AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf ∗buf, void ∗frombuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf ∗buf, void ∗tobuf, unsigned int N)
- AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf ∗buf)

Setups the Automated runs for continuous mode runs.

- AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf ∗buf, AIOCmd ∗cmd, AIORET_TYPE(∗callback)(AIOContinuousBuf ∗buf))

    Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd ∗cmd.
- AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf ∗buf)
- THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf ∗buf, unsigned short ∗tmp, unsigned tmpsize, unsigned size)

    will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf ∗buf, unsigned short ∗read_buf, unsigned read_buf_size)
- AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf ∗buf, unsigned short ∗read_buf, unsigned tmpbuffer_size, int64_t num_scans)

    will read in an integer number of scan counts if there is room.
- AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf ∗buf)

    returns the number of Scans accross all channels that still remain in the buffer
- AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf ∗buf, unsigned int hz)
- AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf ∗buf)
- AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf ∗buf, ADGainCode gainCode)
- AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf ∗buf, AIOBufferType ∗readbuf, unsigned readbufsize, unsigned size)

    Reads the current available amount of data from buf, into the readbuf datastructure ∗.
- AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf ∗buf, AIOBufferType ∗writebuf, unsigned wrbufsize, unsigned size, AIOContinuousBufMode flag)

    Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.
- AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf ∗buf, unsigned short ∗data, unsigned datasize, unsigned size, AIOContinuousBufMode flag)
- AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf ∗buf)
- char ∗ AIOContinuousBufToJSON (AIOContinuousBuf ∗buf)
- AIOContinuousBuf ∗ NewAIOContinuousBufFromJSON (const char ∗json_string)

24.88.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.88.2 Macro Definition Documentation

#define ROOTCLOCK 10000000

24.88.3 Typedef Documentation

typedef void∗(∗ AIOUSB_WorkFn)(void ∗obj)

typedef struct AIOContinuousBuf AIOContinuousBuf

AIOContinuousBuf provides a buffer that is used with the AIOUSB highspeed data acquisition API.

It is designed to provide an ease of use with getting these acquisitions running with as little fuss as possible. They key flow for using this buffer is the following:

- Create a new AIOContinuousBuf of a certain size that is large enough to handle the running clock rate ∗ number-_of_oversamples ∗
- Assign a device index to the AIOContinuousBuf
- Start am acquisition by calling AIOContinuousBufInitiateCallbackAcquisition;
- Process the input data using either a simple while loop burst_test.c

  or using the callback function as in

24.88.4 Enumeration Type Documentation

enum AIO_CONT_BUF_TYPE

Enumerator

AIO_CONT_BUF_TYPE_COUNTS

AIO_CONT_BUF_TYPE_VOLTS

24.88.5 Function Documentation

AIOContinuousBuf∗ NewAIOContinuousBuf (unsigned long *deviceIndex*, unsigned *num_channels*, unsigned *num_oversamples*, unsigned *base_size*)

Simplest constructor for the continuous mode buffer. It will by default use counts (uint16_t) as the fundamental size/type (AIO_CONT_BUF_TYPE_COUNTS).

Parameters

deviceIndex	
num_channels	
num_-oversamples	
base_size	

Returns

AIOContinuousBuf∗ NewAIOContinuousBufForCounts (unsigned long *DeviceIndex*, unsigned *scancounts*, unsigned *num_channels*)

AIOContinuousBuf∗ NewAIOContinuousBufForVolts (unsigned long *DeviceIndex*, unsigned *scancounts*, unsigned *num_channels*, unsigned *num_oversamples*)

AIORET_TYPE DeleteAIOContinuousBuf (AIOContinuousBuf ∗ *buf*)

Destructor for AIOContinuousBuf object.

AIORET_TYPE AIOContinuousBufInitConfiguration (AIOContinuousBuf ∗ *buf*)

Sets up an AIOContinuousBuffer to perform Internal , counter based scanning.

Parameters

buf	Our AIOContinuousBuffer

Returns

AIOUSB_SUCCESS if successful, value $<$ 0 if not.

AIORET_TYPE AIOContinuousBufInitADCConfigBlock (AIOContinuousBuf ∗ *buf,* unsigned *size,* ADGainCode *gainCode,* AIOUSB_BOOL *diffMode,* unsigned char *os,* AIOUSB_BOOL *dfs*)

AIOUSB_WorkFn AIOContinuousBufGetCallback (AIOContinuousBuf ∗ *buf*)

Returns.

Parameters

buf	

Returns

Pointer to our work function

AIORET_TYPE AIOContinuousBufSetCallback (AIOContinuousBuf ∗ *buf,* void ∗(∗)(void ∗object) *work*)

AIORET_TYPE AIOContinuousBufSetStreamingBlockSize (AIOContinuousBuf ∗ *buf,* unsigned *sblksize*)

AIORET_TYPE AIOContinuousBufGetStreamingBlockSize (AIOContinuousBuf ∗ *buf*)

ADCConfigBlock∗ AIOContinuousBufGetADCConfigBlock (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetNumberChannels (AIOContinuousBuf ∗ *buf,* unsigned *num_channels*)

will set the number of channels that this AIOcontinuousbuf watches and if the number isn't divisibly into the total size of the fifo, the fifo gets resized

AIORET_TYPE AIOContinuousBufGetNumberChannels (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetOversample (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetOversample (AIOContinuousBuf ∗ *buf,* unsigned *num_oversamples*)

AIORET_TYPE AIOContinuousBufNumberChannels (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetBaseSize (AIOContinuousBuf ∗ *buf,* size_t *newbase*)

AIORET_TYPE AIOContinuousBufGetBaseSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetBufferSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetUnitSize (AIOContinuousBuf ∗ *buf,* uint16_t *new_unit_size*)

AIORET_TYPE AIOContinuousBufGetUnitSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetTesting (AIOContinuousBuf ∗ *buf,* AIOUSB_BOOL *testing*)

AIORET_TYPE AIOContinuousBufGetTesting (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSendPreConfig (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetStartAndEndChannel (AIOContinuousBuf ∗ *buf,* unsigned *startChannel,* unsigned *endChannel*)

AIORET_TYPE AIOContinuousBufSetAllGainCodeAndDiffMode (AIOContinuousBuf ∗ *buf,* ADGainCode *gain,* AIOUSB_BOOL *diff*)

AIORET_TYPE AIOContinuousBufGetDeviceIndex (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDiscardFirstSample (AIOContinuousBuf * *buf,* AIOUSB_BOOL *discard*)

AIORET_TYPE AIOContinuousBufSetChannelMask (AIOContinuousBuf * *buf,* AIOChannelMask * *mask*)

Sets the channel mask.

Parameters

buf	
mask	

Returns

AIORET_TYPE AIOContinuousBufNumberSignals (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetChannelRange (AIOContinuousBuf ∗ *buf,* unsigned *startChannel,* unsigned *endChannel,* unsigned *gainCode*)

AIORET_TYPE AIOContinuousBufSaveConfig (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDeviceIndex (AIOContinuousBuf ∗ *buf,* unsigned long *DeviceIndex*)

AIORET_TYPE AIOContinuousBufResetDevice (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetTimeout (AIOContinuousBuf ∗ *buf,* unsigned *timeout*)

AIORET_TYPE AIOContinuousBufGetTimeout (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetDebug (AIOContinuousBuf ∗ *buf,* AIOUSB_BOOL *debug*)

AIORET_TYPE AIOContinuousBufGetDebug (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetNumberScans (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetNumberScans (AIOContinuousBuf ∗ *buf,* int64_t *num_scans*)

AIORET_TYPE AIOContinuousBufNumberWriteSamplesRemaining (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufNumberSamplesAvailable (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetNumberSamplesPerScan (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetTotalSamplesExpected (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufReset (AIOContinuousBuf ∗ *buf*)

Todo Fix this to use condition variable

AIORET_TYPE AIOContinuousBufPushN (AIOContinuousBuf ∗ *buf,* void ∗ *frombuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufPopN (AIOContinuousBuf ∗ *buf,* void ∗ *tobuf,* unsigned int *N*)

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf ∗ *buf*)

Parameters

buf	

Returns

AIORET_TYPE AIOContinuousBufUnlock (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf ∗ *buf*)

Setups the Automated runs for continuous mode runs.

Parameters

buf	

Returns

Note

Setup counters see reference in `USB AIO documentation`

Note

BufStart (or bulk read) must occur before loading the counters

Allow the other command to be run

AIORET_TYPE AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction (AIOContinuousBuf ∗ *buf,* AIOCmd ∗ *cmd,* AIORET_TYPE(∗)(AIOContinuousBuf ∗buf) *callback*)

Sets up a smart continuos mode acquisition allowing the user to specify a callback function that is called based on the arguments constructed in AIOCmd ∗cmd.

The user can specify that the callback is called after each oversample, full chanell, full scan, or N number of scans.

Parameters

buf	
cmd	
callback	

Returns

>= 0 if successful, < 0 if failure

AIORET_TYPE AIOContinuousBufStopAcquisition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufInitiateCallbackAcquisition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetReadPosition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetWritePosition (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetRemainingSize (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetStatus (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetExitCode (AIOContinuousBuf ∗ *buf*)

THREAD_STATUS AIOContinuousBufGetRunStatus (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufPending (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufGetScansRead (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufReadIntegerScanCounts (AIOContinuousBuf ∗ *buf,* unsigned short ∗ *read_buf,* unsigned *tmpbuffer_size,* unsigned *size*)

will read in an integer number of scan counts if there is room.

- **Parameters**

buf	
read_buf	
tmpbuffer_size	

size	The size of the tmp buffer

Returns

AIORET_TYPE AIOContinuousBufReadCompleteScanCounts (AIOContinuousBuf ∗ *buf*, unsigned short ∗ *read_buf*, unsigned *read_buf_size*)

AIORET_TYPE AIOContinuousBufReadIntegerNumberOfScans (AIOContinuousBuf ∗ *buf*, unsigned short ∗ *read_buf*, unsigned *tmpbuffer_size*, int64_t *num_scans*)

will read in an integer number of scan counts if there is room.

Parameters

•

buf	
read_buf	
tmpbuffer_size	
num_scans	

Returns

AIORET_TYPE AIOContinuousBufSetCountsBuffer (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSetVoltsBuffer (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufCountScansAvailable (AIOContinuousBuf ∗ *buf*)

returns the number of Scans accross all channels that still remain in the buffer

AIORET_TYPE AIOContinuousBufSetClock (AIOContinuousBuf ∗ *buf*, unsigned int *hz*)

AIORET_TYPE AIOContinuousBufGetClock (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufEnd (AIOContinuousBuf ∗ *buf*)

AIORET_TYPE AIOContinuousBufSimpleSetupConfig (AIOContinuousBuf ∗ *buf*, ADGainCode *gainCode*)

AIORET_TYPE AIOContinuousBufRead (AIOContinuousBuf ∗ *buf*, AIOBufferType ∗ *readbuf*, unsigned *readbufsize*, unsigned *size*)

Reads the current available amount of data from buf, into the readbuf datastructure ∗.

Parameters

buf	
readbuf	
readbufsize	
size	

Returns

If number is positive, it is the number of bytes that have been read.

AIORET_TYPE AIOContinuousBufWrite (AIOContinuousBuf ∗ *buf*, AIOBufferType ∗ *writebuf*, unsigned *wrbufsize*, unsigned *size*, AIOContinuousBufMode *flag*)

Allows one to write into the AIOContinuousBuf buffer a given amount (size) of data.

Parameters

buf	
writebuf	
wrbufsize	
size	
flag	

Returns

Status of whether the write was successful , if so returning the number of bytes written or if there was insufficient space, it returns negative error code. If the number is >= 0, then this corresponds to the number of bytes that were written into the buffer.

AIORET_TYPE AIOContinuousBufWriteCounts (AIOContinuousBuf ∗ *buf*, unsigned short ∗ *data*, unsigned *datasize*, unsigned *size*, AIOContinuousBufMode *flag*)

AIORET_TYPE AIOContinuousBufCleanup (AIOContinuousBuf ∗ *buf*)

char∗ AIOContinuousBufToJSON (AIOContinuousBuf ∗ *buf*)

AIOContinuousBuf∗ NewAIOContinuousBufFromJSON (const char ∗ *json_string*)

24.89 lib/AIOCountsConverter.c File Reference

General header files for the AIOUSB library.

```
#include "AIOTypes.h"
#include "AIOUSB_Core.h"
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include <pthread.h>
```

Functions

- int default_out (AIOCountsConverter ∗cc, unsigned rounded_num_counts)
- int enhanced_out (AIOCountsConverter ∗cc, unsigned rounded_num_counts)
- AIOCountsConverter ∗ NewAIOCountsConverterWithBuffer (void ∗buf, unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter ∗ NewAIOCountsConverterWithScanLimiter (void ∗buf, unsigned num_scans, unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter ∗ NewAIOCountsConverter (unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- void DeleteAIOCountsConverter (AIOCountsConverter ∗ccv)
- void AIOCountsConverterReset (AIOCountsConverter ∗cc)
- AIORET_TYPE AIOCountsConverterConvertNScans (AIOCountsConverter ∗ccv, int num_scans)
- AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (AIOCountsConverter ∗ccv)
- double Convert (AIOGainRange range, unsigned short sum)
- AIORET_TYPE AIOCountsConverterConvertFifo (AIOCountsConverter ∗cc, void ∗tobufptr, void ∗frombufptr, unsigned num_counts)
- AIORET_TYPE AIOCountsConverterConvert (AIOCountsConverter ∗cc, void ∗to_buf, void ∗from_buf, unsigned num_bytes)
- AIOGainRange ∗ NewAIOGainRangeFromADCConfigBlock (ADCConfigBlock ∗adc)
- void DeleteAIOGainRange (AIOGainRange ∗agr)

24.89.1 Detailed Description

General header files for the AIOUSB library.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.89.2 Function Documentation

int default_out (**AIOCountsConverter** ∗ *cc,* unsigned *rounded_num_counts*)

int enhanced_out (**AIOCountsConverter** ∗ *cc,* unsigned *rounded_num_counts*)

AIOCountsConverter∗ NewAIOCountsConverterWithBuffer (void ∗ *buf,* unsigned *num_channels,* **AIOGainRange** ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter∗ NewAIOCountsConverterWithScanLimiter (void ∗ *buf,* unsigned *num_scans,* unsigned *num_channels,* **AIOGainRange** ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter∗ NewAIOCountsConverter (unsigned *num_channels,* **AIOGainRange** ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

void DeleteAIOCountsConverter (**AIOCountsConverter** ∗ *ccv*)

void AIOCountsConverterReset (**AIOCountsConverter** ∗ *cc*)

AIORET_TYPE AIOCountsConverterConvertNScans (**AIOCountsConverter** ∗ *ccv,* int *num_scans*)

AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (**AIOCountsConverter** ∗ *ccv*)

double Convert (**AIOGainRange** *range,* unsigned short *sum*)

AIORET_TYPE AIOCountsConverterConvertFifo (**AIOCountsConverter** ∗ *cc,* void ∗ *tobufptr,* void ∗ *frombufptr,* unsigned *num_counts*)

Parameters

cc	Counts converter object
tobufptr	ToFifo (double)
frombufptr	From Fifo (unsigned short)
num_counts	number of counts to convert

Returns

Number of tobufptr objects that have been created

AIORET_TYPE AIOCountsConverterConvert (**AIOCountsConverter** ∗ *cc,* void ∗ *to_buf,* void ∗ *from_buf,* unsigned *num_bytes*)

AIOGainRange∗ NewAIOGainRangeFromADCConfigBlock (**ADCConfigBlock** ∗ *adc*)

void DeleteAIOGainRange (**AIOGainRange** ∗ *agr*)

24.90 lib/AIOCountsConverter.h File Reference

```
#include "AIOTypes.h"
#include "ADCConfigBlock.h"
#include "AIOContinuousBuffer.h"
#include "AIOFifo.h"
```

Data Structures

- struct AIOGainRange
- struct aio_counts_converter

Typedefs

- typedef struct aio_counts_converter AIOCountsConverter

Functions

- AIOCountsConverter ∗ NewAIOCountsConverterWithBuffer (void ∗buf, unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter ∗ NewAIOCountsConverter (unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- AIOCountsConverter ∗ NewAIOCountsConverterFromAIOContinuousBuf (void ∗buf)
- AIOCountsConverter ∗ NewAIOCountsConverterWithScanLimiter (void ∗buf, unsigned num_scans, unsigned num_channels, AIOGainRange ∗ranges, unsigned num_oversamples, unsigned unit_size)
- void AIOCountsConverterReset (AIOCountsConverter ∗cc)
- void DeleteAIOCountsConverter (AIOCountsConverter ∗ccv)
- AIORET_TYPE AIOCountsConverterConvertNScans (AIOCountsConverter ∗cc, int num_scans)
- AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (AIOCountsConverter ∗cc)
- AIORET_TYPE AIOCountsConverterConvert (AIOCountsConverter ∗cc, void ∗tobuf, void ∗frombuf, unsigned num_bytes)
- AIORET_TYPE AIOCountsConverterConvertFifo (AIOCountsConverter ∗cc, void ∗tobuf, void ∗frombuf, unsigned num_bytes)
- AIOGainRange ∗ NewAIOGainRangeFromADCConfigBlock (ADCConfigBlock ∗adc)
- void DeleteAIOGainRange (AIOGainRange ∗)

24.90.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.90.2 Typedef Documentation

typedef struct aio_counts_converter AIOCountsConverter

24.90.3 Function Documentation

AIOCountsConverter∗ NewAIOCountsConverterWithBuffer (void ∗ *buf,* unsigned *num_channels,* AIOGainRange ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter∗ NewAIOCountsConverter (unsigned *num_channels,* AIOGainRange ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

AIOCountsConverter∗ NewAIOCountsConverterFromAIOContinuousBuf (void ∗ *buf*)

AIOCountsConverter∗ NewAIOCountsConverterWithScanLimiter (void ∗ *buf,* unsigned *num_scans,* unsigned *num_channels,* **AIOGainRange** ∗ *ranges,* unsigned *num_oversamples,* unsigned *unit_size*)

void AIOCountsConverterReset (**AIOCountsConverter** ∗ *cc*)

void DeleteAIOCountsConverter (**AIOCountsConverter** ∗ *ccv*)

AIORET_TYPE AIOCountsConverterConvertNScans (**AIOCountsConverter** ∗ *cc,* int *num_scans*)

AIORET_TYPE AIOCountsConverterConvertAllAvailableScans (**AIOCountsConverter** ∗ *cc*)

AIORET_TYPE AIOCountsConverterConvert (**AIOCountsConverter** ∗ *cc,* void ∗ *tobuf,* void ∗ *frombuf,* unsigned *num_bytes*)

AIORET_TYPE AIOCountsConverterConvertFifo (**AIOCountsConverter** ∗ *cc,* void ∗ *tobufptr,* void ∗ *frombufptr,* unsigned *num_counts*)

Parameters

cc	Counts converter object
tobufptr	ToFifo (double)
frombufptr	From Fifo (unsigned short)
num_counts	number of counts to convert

Returns

Number of tobufptr objects that have been created

AIOGainRange∗ NewAIOGainRangeFromADCConfigBlock (**ADCConfigBlock** ∗ *adc*)

void DeleteAIOGainRange (**AIOGainRange** ∗)

24.91 lib/AIODeviceInfo.c File Reference

```
#include "AIODeviceInfo.h"
#include "AIODeviceTable.h"
#include "AIOUSBDevice.h"
#include "AIOUSB_Core.h"
```

Functions

- AIODeviceInfo ∗ NewAIODeviceInfo ()
- void DeleteAIODeviceInfo (AIODeviceInfo ∗di)
- const char ∗ AIODeviceInfoGetName (AIODeviceInfo ∗di)
- AIORET_TYPE AIODeviceInfoGetCounters (AIODeviceInfo ∗di)
- AIORET_TYPE AIODeviceInfoGetDIOBytes (AIODeviceInfo ∗di)
- AIODeviceInfo ∗ AIODeviceInfoGet (unsigned long DeviceIndex)

24.91.1 Function Documentation

AIODeviceInfo∗ NewAIODeviceInfo ( )

void DeleteAIODeviceInfo (**AIODeviceInfo** ∗ *di*)

const char∗ AIODeviceInfoGetName (**AIODeviceInfo** ∗ *di*)

AIORET_TYPE AIODeviceInfoGetCounters (**AIODeviceInfo** ∗ *di*)

AIORET_TYPE AIODeviceInfoGetDIOBytes (**AIODeviceInfo** ∗ *di*)

AIODeviceInfo ∗ AIODeviceInfoGet (unsigned long *DeviceIndex*)

24.92 lib/AIODeviceInfo.h File Reference

```
#include "AIOTypes.h"
#include "AIOUSB_Core.h"
#include <stdlib.h>
#include <string.h>
```

Data Structures

- struct AIODeviceInfo

Typedefs

- typedef struct AIODeviceInfo AIODeviceInfo

Functions

- AIODeviceInfo ∗ NewAIODeviceInfo ()
- void DeleteAIODeviceInfo (AIODeviceInfo ∗di)
- const char ∗ AIODeviceInfoGetName (AIODeviceInfo ∗di)
- AIODeviceInfo ∗ AIODeviceInfoGet (unsigned long DeviceIndex)
- AIORET_TYPE AIODeviceInfoGetCounters (AIODeviceInfo ∗di)
- AIORET_TYPE AIODeviceInfoGetDIOBytes (AIODeviceInfo ∗di)

24.92.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.92.2 Typedef Documentation

typedef struct AIODeviceInfo AIODeviceInfo

24.92.3 Function Documentation

AIODeviceInfo∗ NewAIODeviceInfo ( )

void DeleteAIODeviceInfo (AIODeviceInfo ∗ *di*)

const char∗ AIODeviceInfoGetName (AIODeviceInfo ∗ *di*)

AIODeviceInfo∗ AIODeviceInfoGet (unsigned long *DeviceIndex*)

AIORET_TYPE AIODeviceInfoGetCounters (AIODeviceInfo ∗ *di*)

AIORET_TYPE AIODeviceInfoGetDIOBytes (AIODeviceInfo ∗ *di*)

24.93 lib/AIODeviceQuery.c File Reference

A simple structure for querying a USB card . This provides a simpler interface for more complicated queries going forward.

```
#include "AIODeviceQuery.h"
#include "AIODeviceTable.h"
```

Functions

- AIODeviceQuery ∗ NewAIODeviceQuery (unsigned long DeviceIndex)

  Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.
- AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery ∗devq)

  Destructor for AIODeviceQuery ∗.
- char ∗ AIODeviceQueryToStr (AIODeviceQuery ∗devq)

  Converts the AIODeviceQuery into a string representation.
- char ∗ AIODeviceQueryToRepr (AIODeviceQuery ∗devq)

  Repr version of this product.
- AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery ∗devq)

  Returns the Product ID of the device in question.
- AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery ∗devq)

  Returns the Index associated with the AIODeviceQuery.
- AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery ∗devq)

  Returns the strlenght of the Device name of the device in question.
- char ∗ AIODeviceQueryGetName (AIODeviceQuery ∗devq)

  Returns the name of the Device at the index in question.
- AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery ∗devq)

  Returns number of Digital bytes for the device in question.
- AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery ∗devq)

  Returns number of Counters for the device in question.

24.93.1 Detailed Description

A simple structure for querying a USB card . This provides a simpler interface for more complicated queries going forward.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.93.2 Function Documentation

AIODeviceQuery∗ NewAIODeviceQuery (unsigned long *DeviceIndex*)

Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.

Parameters

DeviceIndex	

Returns

AIODeviceQuery ∗ object

AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery ∗ *devq*)

Destructor for AIODeviceQuery ∗.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, success, otherwise error

char∗ AIODeviceQueryToStr (AIODeviceQuery ∗ *devq*)

Converts the AIODeviceQuery into a string representation.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

char∗ AIODeviceQueryToRepr (AIODeviceQuery ∗ *devq*)

Repr version of this product.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery ∗ *devq*)

Returns the Product ID of the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, the product ID in question, otherwise error

AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery ∗ *devq*)

Returns the Index associated with the AIODeviceQuery.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0 index , otherwise error

AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery ∗ *devq*)

Returns the strlenght of the Device name of the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, the name length in question, otherwise error

char∗ AIODeviceQueryGetName (AIODeviceQuery ∗ *devq*)

Returns the name of the Device at the index in question.

Parameters

devq	AIODeviceQuery ∗

Returns

!= 0 the name of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery ∗ *devq*)

Returns number of Digital bytes for the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0 the number of dio bytes of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery ∗ *devq*)

Returns number of Counters for the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

!= 0 the name of the card, otherwise an error

24.94 lib/AIODeviceQuery.h File Reference

```
#include "AIOTypes.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <libusb.h>
#include <math.h>
```

Data Structures

- struct AIODeviceQuery

Typedefs

- typedef struct AIODeviceQuery AIODeviceQuery

Functions

- AIODeviceQuery ∗ NewAIODeviceQuery (unsigned long DeviceIndex)

  Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.
- AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery ∗devq)

  Destructor for AIODeviceQuery ∗.
- char ∗ AIODeviceQueryToStr (AIODeviceQuery ∗devq)

  Converts the AIODeviceQuery into a string representation.
- char ∗ AIODeviceQueryToRepr (AIODeviceQuery ∗devq)

  Repr version of this product.
- AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery ∗devq)

  Returns the Product ID of the device in question.
- AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery ∗devq)

  Returns the strlenght of the Device name of the device in question.
- char ∗ AIODeviceQueryGetName (AIODeviceQuery ∗devq)

  Returns the name of the Device at the index in question.
- AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery ∗devq)

  Returns number of Digital bytes for the device in question.
- AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery ∗devq)

  Returns number of Counters for the device in question.
- AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery ∗devq)

  Returns the Index associated with the AIODeviceQuery.

24.94.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.94.2 Typedef Documentation

typedef struct AIODeviceQuery AIODeviceQuery

24.94.3 Function Documentation

AIODeviceQuery∗ NewAIODeviceQuery (unsigned long *DeviceIndex*)

Constructor of a AIODeviceQuery, and using the DeviceIndex , queries the device at that index.

Parameters

DeviceIndex	

Returns

AIODeviceQuery ∗ object

AIORET_TYPE DeleteAIODeviceQuery (AIODeviceQuery ∗ *devq*)

Destructor for AIODeviceQuery ∗.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, success, otherwise error

char∗ AIODeviceQueryToStr (AIODeviceQuery ∗ *devq*)

Converts the AIODeviceQuery into a string representation.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

char∗ AIODeviceQueryToRepr (AIODeviceQuery ∗ *devq*)

Repr version of this product.

Parameters

devq	AIODeviceQuery ∗

Returns

String representing the Device query, NULL if not defined

AIORET_TYPE AIODeviceQueryGetProductID (AIODeviceQuery ∗ *devq*)

Returns the Product ID of the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, the product ID in question, otherwise error

AIORET_TYPE AIODeviceQueryNameSize (AIODeviceQuery ∗ *devq*)

Returns the strlenght of the Device name of the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0, the name length in question, otherwise error

char∗ AIODeviceQueryGetName (AIODeviceQuery ∗ *devq*)

Returns the name of the Device at the index in question.

Parameters

devq	AIODeviceQuery ∗

Returns

!= 0 the name of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumDIOBytes (AIODeviceQuery ∗ *devq*)

Returns number of Digital bytes for the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0 the number of dio bytes of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetNumCounters (AIODeviceQuery ∗ *devq*)

Returns number of Counters for the device in question.

Parameters

devq	AIODeviceQuery ∗

Returns

!= 0 the name of the card, otherwise an error

AIORET_TYPE AIODeviceQueryGetIndex (AIODeviceQuery ∗ *devq*)

Returns the Index associated with the AIODeviceQuery.

Parameters

devq	AIODeviceQuery ∗

Returns

>= 0 index , otherwise error

24.95 lib/AIODeviceTable.c File Reference

```
#include "AIODeviceTable.h"
#include <string.h>
#include <errno.h>
```

Macros

- #define NUM_PROD_NAMES (sizeof(productIDNameTable) / sizeof(productIDNameTable[0]))

Functions

- AIOUSB_BOOL AIOUSB_SetInit ()
- void AIODeviceTableInit (void)
- AIOUSB_BOOL AIOUSB_IsInit ()
- unsigned long AIOUSB_InitTest (void)
- AIOUSB_BOOL AIOUSB_Cleanup ()
- unsigned long QueryDeviceInfo (unsigned long DeviceIndex, unsigned long ∗pPID, unsigned long ∗pNameSize, char ∗pName, unsigned long ∗pDIOBytes, unsigned long ∗pCounters)

  Identifying devices on the USB bus.
- PRIVATE char ∗ ProductIDToName (unsigned int productID)

  this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()
- PRIVATE AIORET_TYPE ProductNameToID (const char ∗name)
- AIORET_TYPE GetDevices (void)
- USBDevice ∗ AIODeviceTableGetUSBDeviceAtIndex (unsigned long DeviceIndex, AIORESULT ∗res)
- char ∗ GetSafeDeviceName (unsigned long DeviceIndex)

  GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.
- AIORET_TYPE AIOUSB_EnsureOpen (unsigned long DeviceIndex)
- AIOUSBDevice ∗ AIODeviceTableGetDeviceAtIndex (unsigned long DeviceIndex, AIORESULT ∗res)
- AIOUSBDevice ∗ AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSBGetError ()

- AIORESULT AIODeviceTableAddDeviceToDeviceTable (int ∗numAccesDevices, unsigned long productID)

  A mock function that can set up the DeviceTable with any type of devices.
- AIORESULT AIODeviceTableAddDeviceToDeviceTableWithUSBDevice (int ∗numAccesDevices, unsigned long productID, USBDevice ∗usb_dev)
- AIORET_TYPE ClearAIODeviceTable (int numDevices)

  cleans up the AIODeviceTable and frees any memory associated with it.
- AIORESULT AIODeviceTableSetDeviceID (int index, AIOUSBDevice ∗dev)
- AIORESULT AIOUSB_GetAllDevices ()
- AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long ∗products, int length)
- void CloseAllDevices (void)
- unsigned long AIODeviceTableClearDevices (void)
- unsigned long ClearDevices (void)
- AIORET_TYPE AIODeviceTablePopulateTable (void)

  populate device table with ACCES devices found on USB bus
- AIORET_TYPE AIOUSB_Init (void)

  AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.
- AIORET_TYPE AIOUSB_Exit ()
- AIORET_TYPE AIOUSB_Reset (unsigned long DeviceIndex)

Variables

- AIOUSBDevice deviceTable [MAX_USB_DEVICES]
- unsigned long AIOUSB_INIT_PATTERN = 0x9b6773adul
- unsigned long aiousbInit = 0

24.95.1 Macro Definition Documentation

#define NUM_PROD_NAMES (sizeof(productIDNameTable) / sizeof(productIDNameTable[0]))

24.95.2 Function Documentation

AIOUSB_BOOL AIOUSB_SetInit ( )

void AIODeviceTableInit (void)

AIOUSB_BOOL AIOUSB_IsInit ( )

unsigned long AIOUSB_InitTest (void)

AIOUSB_BOOL AIOUSB_Cleanup ( )

unsigned long QueryDeviceInfo (unsigned long *DeviceIndex*, unsigned long ∗ *pPID*, unsigned long ∗ *pNameSize*, char ∗ *pName*, unsigned long ∗ *pDIOBytes*, unsigned long ∗ *pCounters*)

Identifying devices on the USB bus.

Parameters

DeviceIndex	
pPID	
pNameSize	
pName	
pDIOBytes	
pCounters	

Returns

PRIVATE char∗ ProductIDToName (unsigned int *productID*)

this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()

productIDIndex[] represents an index into productIDNameTable[], sorted by product ID; specifically, it contains pointers into productIDNameTable[]; to get the actual product ID, the pointer in productIDIndex[] must be dereferenced; using a separate index table instead of sorting productIDNameTable[] directly permits us to create multiple indexes, in particular, a second index sorted by product name

PRIVATE AIORET_TYPE ProductNameToID (const char ∗ *name*)

This function is the complement of ProductIDToName() and returns the product ID for a given name; this function should be used with care; it will work reliably if passed a name obtained from ProductIDToName(); however, if passed a name obtained from the device itself it may not work; the reason is that devices contain their own name strings, which are most likely identical to the names defined in this module, but not guaranteed to be so; that's not as big a problem as it sounds, however, because if one has the means to obtain the name from the device, then they also have access to the device's product ID, so calling this function is unnecessary; this function is mainly for performing simple conversions between product names and IDs, primarily to support user interfaces

Parameters

name	

Returns

productNameIndex[] represents an index into productIDNameTable[], sorted by product name (see notes for ProductID-ToName())

<index of product names in productIDNameTable[]

random pattern

== INIT_PATTERN if index has been created

AIORET_TYPE GetDevices (void)

Note

Will call AIOUSB_Init() in case the AIOUSB API has not been initialized with the AIOUSB_Init() function. This is a convenience function.

Returns

if $<$ 0 Error else SUCCESS

Note

we clear the device table to erase references to devices which may have been unplugged; any device indexes to devices that have not been unplugged, which the user may be using, *should* still be valid

USBDevice∗ AIODeviceTableGetUSBDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT ∗ *res*)

Parameters

	DeviceIndex	Device index we are probing
`out`	*res*	Error code if unable to find USB device

Returns

USBDevice ∗ A Usb handle that can be used for USB transactions

char∗ GetSafeDeviceName (unsigned long *DeviceIndex*)

GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.

AIORET_TYPE AIOUSB_EnsureOpen (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSBDevice∗ AIODeviceTableGetDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT ∗ *res*)

AIOUSBDevice∗ AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long *DeviceIndex*)

AIORET_TYPE AIOUSBGetError ( )

AIORESULT AIODeviceTableAddDeviceToDeviceTable (int ∗ *numAccesDevices,* unsigned long *productID*)

A mock function that can set up the DeviceTable with any type of devices.

AIORESULT AIODeviceTableAddDeviceToDeviceTableWithUSBDevice (int ∗ *numAccesDevices,* unsigned long *productID,* USBDevice ∗ *usb_dev*)

AIORET_TYPE ClearAIODeviceTable (int *numDevices*)

cleans up the AIODeviceTable and frees any memory associated with it.

Parameters

numDevices	

Returns

AIORESULT AIODeviceTableSetDeviceID (int *index,* AIOUSBDevice ∗ *dev*)

AIORESULT AIOUSB_GetAllDevices ( )

AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long ∗ *products,* int *length*)

void CloseAllDevices (void)

unsigned long AIODeviceTableClearDevices (void)

unsigned long ClearDevices (void)

AIORET_TYPE AIODeviceTablePopulateTable (void)

populate device table with ACCES devices found on USB bus

Todo Rely on Global Header files for the functionality of devices / cards as opposed to hard coding

Note

populate device table so users can use diFirst and diOnly immediately; be sure to call PopulateDeviceTable() after 'aiousbInit = AIOUSB_INIT_PATTERN;'

AIORET_TYPE AIOUSB_Init (void)

AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.

Hence you should just run AIOUSB_Init() once at the beginning and then the AIOUSB_Exit() once at the end after every thread acquiring data has been stopped.

AIORET_TYPE AIOUSB_Exit ( )

AIORET_TYPE AIOUSB_Reset (unsigned long *DeviceIndex*)

24.95.3 Variable Documentation

AIOUSBDevice deviceTable[MAX_USB_DEVICES]

unsigned long AIOUSB_INIT_PATTERN = 0x9b6773adul

unsigned long aiousbInit = 0

24.96 lib/AIODeviceTable.h File Reference

```
#include "AIOTypes.h"
#include "AIOUSBDevice.h"
#include "AIOUSB_Core.h"
#include <string.h>
#include "libusb.h"
#include <stdlib.h>
#include <errno.h>
```

Functions

- AIORESULT AIODeviceTableAddDeviceToDeviceTable (int ∗numAccesDevices, unsigned long productID)

  A mock function that can set up the DeviceTable with any type of devices.
- AIORESULT AIODeviceTableAddDeviceToDeviceTableWithUSBDevice (int ∗numAccesDevices, unsigned long productID, USBDevice ∗usb_dev)
- AIORET_TYPE AIODeviceTablePopulateTable (void)

  populate device table with ACCES devices found on USB bus
- AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long ∗products, int length)
- AIORESULT AIODeviceTableClearDevices (void)
- AIORESULT ClearDevices (void)
- AIOUSBDevice ∗ AIODeviceTableGetDeviceAtIndex (unsigned long DeviceIndex, AIORESULT ∗res)
- AIOUSBDevice ∗ AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long DeviceIndex)
- USBDevice ∗ AIODeviceTableGetUSBDeviceAtIndex (unsigned long DeviceIndex, AIORESULT ∗res)
- unsigned long QueryDeviceInfo (unsigned long DeviceIndex, unsigned long ∗pPID, unsigned long ∗pNameSize, char ∗pName, unsigned long ∗pDIOBytes, unsigned long ∗pCounters)

  Identifying devices on the USB bus.
- AIORET_TYPE GetDevices (void)
- char ∗ GetSafeDeviceName (unsigned long DeviceIndex)

  GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.
- char ∗ ProductIDToName (unsigned int productID)

  this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()
- AIORET_TYPE ProductNameToID (const char ∗name)
- AIORET_TYPE AIOUSB_Init (void)

  AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.
- AIORET_TYPE AIOUSB_EnsureOpen (unsigned long DeviceIndex)
- AIOUSB_BOOL AIOUSB_IsInit ()
- AIORET_TYPE AIOUSB_Exit ()
- AIORET_TYPE AIOUSB_Reset (unsigned long DeviceIndex)
- void AIODeviceTableInit (void)
- AIORET_TYPE ClearAIODeviceTable (int numDevices)

  cleans up the AIODeviceTable and frees any memory associated with it.
- void CloseAllDevices (void)
- AIORESULT AIOUSB_GetAllDevices ()
- AIORET_TYPE AIOUSBGetError ()

Variables

- AIOUSBDevice deviceTable [MAX_USB_DEVICES]
- unsigned long AIOUSB_INIT_PATTERN

24.96.1 Function Documentation

AIORESULT AIODeviceTableAddDeviceToDeviceTable (int ∗ *numAccesDevices,* unsigned long *productID*)

A mock function that can set up the DeviceTable with any type of devices.

AIORESULT AIODeviceTableAddDeviceToDeviceTableWithUSBDevice (int ∗ *numAccesDevices,* unsigned long *productID,* USBDevice ∗ *usb_dev*)

AIORET_TYPE AIODeviceTablePopulateTable (void)

populate device table with ACCES devices found on USB bus

Todo Rely on Global Header files for the functionality of devices / cards as opposed to hard coding

Note

populate device table so users can use diFirst and diOnly immediately; be sure to call PopulateDeviceTable() after 'aiousbInit = AIOUSB_INIT_PATTERN;'

AIORET_TYPE AIODeviceTablePopulateTableTest (unsigned long ∗ *products,* int *length*)

AIORESULT AIODeviceTableClearDevices (void)

AIORESULT ClearDevices (void)

AIOUSBDevice∗ AIODeviceTableGetDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT ∗ *res*)

AIOUSBDevice∗ AIODeviceTableGetAIOUSBDeviceAtIndex (unsigned long *DeviceIndex*)

USBDevice∗ AIODeviceTableGetUSBDeviceAtIndex (unsigned long *DeviceIndex,* AIORESULT ∗ *res*)

Parameters

	DeviceIndex	Device index we are probing
out	*res*	Error code if unable to find USB device

Returns

USBDevice ∗ A Usb handle that can be used for USB transactions

unsigned long QueryDeviceInfo (unsigned long *DeviceIndex,* unsigned long ∗ *pPID,* unsigned long ∗ *pNameSize,* char ∗ *pName,* unsigned long ∗ *pDIOBytes,* unsigned long ∗ *pCounters*)

Identifying devices on the USB bus.

Parameters

DeviceIndex	
pPID	
pNameSize	
pName	
pDIOBytes	
pCounters	

Returns

AIORET_TYPE GetDevices (void)

Note

Will call AIOUSB_Init() in case the AIOUSB API has not been initialized with the AIOUSB_Init() function. This is a convenience function.

Returns

if $<$ 0 Error else SUCCESS

Note

we clear the device table to erase references to devices which may have been unplugged; any device indexes to devices that have not been unplugged, which the user may be using, *should* still be valid

char∗ GetSafeDeviceName (unsigned long *DeviceIndex*)

GetSafeDeviceName() returns a null-terminated device name; if GetSafeDeviceName() is unable to obtain a legitimate device name it returns something like "UNKNOWN" or 0.

char∗ ProductIDToName (unsigned int *productID*)

this function returns the name of a product ID; generally, it's best to use this only as a last resort, since most devices return their name when asked in QueryDeviceInfo()

productIDIndex[] represents an index into productIDNameTable[], sorted by product ID; specifically, it contains pointers into productIDNameTable[]; to get the actual product ID, the pointer in productIDIndex[] must be dereferenced; using a separate index table instead of sorting productIDNameTable[] directly permits us to create multiple indexes, in particular, a second index sorted by product name

AIORET_TYPE ProductNameToID (const char ∗ *name*)

This function is the complement of ProductIDToName() and returns the product ID for a given name; this function should be used with care; it will work reliably if passed a name obtained from ProductIDToName(); however, if passed a name obtained from the device itself it may not work; the reason is that devices contain their own name strings, which are most likely identical to the names defined in this module, but not guaranteed to be so; that's not as big a problem as it sounds, however, because if one has the means to obtain the name from the device, then they also have access to the device's product ID, so calling this function is unnecessary; this function is mainly for performing simple conversions between product names and IDs, primarily to support user interfaces

Parameters

name	

Returns

productNameIndex[] represents an index into productIDNameTable[], sorted by product name (see notes for ProductID-ToName())

<index of product names in productIDNameTable[]

random pattern

== INIT_PATTERN if index has been created

AIORET_TYPE AIOUSB_Init (void)

AIOUSB_Init() and AIOUSB_Exit() are not thread-safe and should not be called while other threads might be accessing global variables.

Hence you should just run AIOUSB_Init() once at the beginning and then the AIOUSB_Exit() once at the end after every thread acquiring data has been stopped.

AIORET_TYPE AIOUSB_EnsureOpen (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSB_BOOL AIOUSB_IsInit ( )

AIORET_TYPE AIOUSB_Exit ( )

AIORET_TYPE AIOUSB_Reset (unsigned long *DeviceIndex*)

void AIODeviceTableInit (void)

AIORET_TYPE ClearAIODeviceTable (int *numDevices*)

cleans up the AIODeviceTable and frees any memory associated with it.

Parameters

numDevices	

Returns

void CloseAllDevices (void)

AIORESULT AIOUSB_GetAllDevices ( )

AIORET_TYPE AIOUSBGetError ( )

24.96.2 Variable Documentation

AIOUSBDevice deviceTable[MAX_USB_DEVICES]

unsigned long AIOUSB_INIT_PATTERN

24.97 lib/AIOEither.c File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <assert.h>
#include <stdarg.h>
#include <stdio.h>
```

Macros

- #define LOOKUP(T) aioret_value_ ## T
- #define AIO_EITHER_CHECK_VALUE(RETVAL, TYPE)
- #define AIO_EITHER_GET_VALUE(RETVAL, TYPE)

Functions

- AIORET_TYPE AIOEitherClear (AIOEither ∗retval)
- AIORET_TYPE AIOEitherSetRight (AIOEither ∗retval, AIO_EITHER_TYPE val, void ∗tmp,...)
- AIORET_TYPE AIOEitherGetRight (AIOEither ∗retval, void ∗tmp,...)
- AIORET_TYPE AIOEitherSetLeft (AIOEither ∗retval, int val)
- AIORET_TYPE AIOEitherGetLeft (AIOEither ∗retval)
- AIOUSB_BOOL AIOEitherHasError (AIOEither ∗retval)
- char ∗ AIOEitherToString (AIOEither ∗retval, AIORET_TYPE ∗result)
- int AIOEitherToInt (AIOEither retval)
- short AIOEitherToShort (AIOEither ∗retval, AIORET_TYPE ∗result)
- unsigned AIOEitherToUnsigned (AIOEither ∗retval, AIORET_TYPE ∗result)
- double AIOEitherToDouble (AIOEither ∗retval, AIORET_TYPE ∗result)
- AIO_NUMBER AIOEitherToAIONumber (AIOEither ∗retval, AIORET_TYPE ∗result)
- AIORET_TYPE AIOEitherToAIORetType (AIOEither either)

24.97.1 Macro Definition Documentation

#define LOOKUP(*T*) aioret_value_ ## T

#define AIO_EITHER_CHECK_VALUE(*RETVAL,* TYPE)

Value:

```
if ( RETVAL->left ) {                                    \
        *result = RETVAL->left;                              \
    } else {                                                 \
        *result = AIOUSB_SUCCESS;                            \
        return *(TYPE *)&(RETVAL->right.number);             \
    }                                                        \
    return (TYPE)AIO_ERROR_VALUE;
```

#define AIO_EITHER_GET_VALUE(*RETVAL,* TYPE)

Value:

```
({                \
            int tmp;                                             \
            if(RETVAL.left) {                                    \
                errno=RETVAL.left;                               \
                tmp=(TYPE)AIO_ERROR_VALUE;                       \
            } else {                                             \
                tmp=*(TYPE *)&(RETVAL.right.number);             \
            };                                                   \
            tmp;})
```

24.97.2 Function Documentation

AIORET_TYPE AIOEitherClear (AIOEither ∗ *retval*)

AIORET_TYPE AIOEitherSetRight (AIOEither ∗ *retval,* AIO_EITHER_TYPE *val,* void ∗ *tmp,* ...)

AIORET_TYPE AIOEitherGetRight (AIOEither ∗ *retval,* void ∗ *tmp,* ...)

AIORET_TYPE AIOEitherSetLeft (AIOEither ∗ *retval,* int *val*)

AIORET_TYPE AIOEitherGetLeft (AIOEither ∗ *retval*)

AIOUSB_BOOL AIOEitherHasError (AIOEither ∗ *retval*)

char∗ AIOEitherToString (AIOEither ∗ *retval,* AIORET_TYPE ∗ *result*)

int AIOEitherToInt (AIOEither *retval*)

short AIOEitherToShort (AIOEither ∗ *retval,* AIORET_TYPE ∗ *result*)

unsigned AIOEitherToUnsigned (AIOEither ∗ *retval,* AIORET_TYPE ∗ *result*)

double AIOEitherToDouble (AIOEither ∗ *retval,* AIORET_TYPE ∗ *result*)

AIO_NUMBER AIOEitherToAIONumber (AIOEither ∗ *retval,* AIORET_TYPE ∗ *result*)

AIORET_TYPE AIOEitherToAIORetType (AIOEither *either*)

24.98 lib/AIOEither.h File Reference

General structure for AIOUSB Fifo.

```
#include "AIOTypes.h"
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
```

Data Structures

- struct aio_either_val
- struct aio_ret_value

Macros

- #define AIO_ERROR_VALUE 0xffffffffffffffff

Typedefs

- typedef struct aio_either_val AIO_EITHER_VALUE_ITEM
- typedef struct aio_ret_value AIOEither

Enumerations

- enum AIO_EITHER_TYPE {
  aioeither_value_int = 1, aioeither_value_int32_t = 1, aioeither_value_uint32_t = 2, aioeither_value_unsigned = 2,
  aioeither_value_uint16_t = 3, aioeither_vlaue_int16_t = 4, aioeither_value_double_t = 5, aioeither_value_double = 5,
  aioeither_value_uint8_t, aioeither_value_string, aioeither_value_longdouble_t, aioeither_value_obj }

Functions

- AIORET_TYPE AIOEitherClear (AIOEither ∗retval)
- AIORET_TYPE AIOEitherSetRight (AIOEither ∗retval, AIO_EITHER_TYPE val, void ∗tmp,...)
- AIORET_TYPE AIOEitherGetRight (AIOEither ∗retval, void ∗tmp,...)
- AIORET_TYPE AIOEitherSetLeft (AIOEither ∗retval, int val)
- AIORET_TYPE AIOEitherGetLeft (AIOEither ∗retval)
- AIOUSB_BOOL AIOEitherHasError (AIOEither ∗retval)
- char ∗ AIOEitherToString (AIOEither ∗retval, AIORET_TYPE ∗result)
- int AIOEitherToInt (AIOEither retval)
- short AIOEitherToShort (AIOEither ∗retval, AIORET_TYPE ∗result)
- unsigned AIOEitherToUnsigned (AIOEither ∗retval, AIORET_TYPE ∗result)
- double AIOEitherToDouble (AIOEither ∗retval, AIORET_TYPE ∗result)
- AIO_NUMBER AIOEitherToAIONumber (AIOEither ∗retval, AIORET_TYPE ∗result)
- AIORET_TYPE AIOEitherToAIORetType (AIOEither either)

24.98.1 Detailed Description

General structure for AIOUSB Fifo. General structure for returning results from routines.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.98.2 Macro Definition Documentation

#define AIO_ERROR_VALUE 0xffffffffffffffff

24.98.3 Typedef Documentation

typedef struct aio_either_val AIO_EITHER_VALUE_ITEM

typedef struct aio_ret_value AIOEither

24.98.4 Enumeration Type Documentation

enum AIO_EITHER_TYPE

Enumerator

aioeither_value_int

aioeither_value_int32_t

aioeither_value_uint32_t

aioeither_value_unsigned

aioeither_value_uint16_t

aioeither_vlaue_int16_t

aioeither_value_double_t

aioeither_value_double

aioeither_value_uint8_t

aioeither_value_string

aioeither_value_longdouble_t

aioeither_value_obj

24.98.5 Function Documentation

AIORET_TYPE AIOEitherClear (**AIOEither** ∗ *retval*)

AIORET_TYPE AIOEitherSetRight (**AIOEither** ∗ *retval,* **AIO_EITHER_TYPE** *val,* void ∗ *tmp,* *...*)

AIORET_TYPE AIOEitherGetRight (**AIOEither** ∗ *retval,* void ∗ *tmp,* *...*)

AIORET_TYPE AIOEitherSetLeft (**AIOEither** ∗ *retval,* int *val*)

AIORET_TYPE AIOEitherGetLeft (**AIOEither** ∗ *retval*)

AIOUSB_BOOL AIOEitherHasError (**AIOEither** ∗ *retval*)

char∗ AIOEitherToString (**AIOEither** ∗ *retval,* **AIORET_TYPE** ∗ *result*)

int AIOEitherToInt (**AIOEither** *retval*)

short AIOEitherToShort (**AIOEither** ∗ *retval,* **AIORET_TYPE** ∗ *result*)

unsigned AIOEitherToUnsigned (**AIOEither** ∗ *retval,* **AIORET_TYPE** ∗ *result*)

double AIOEitherToDouble (**AIOEither** ∗ *retval,* **AIORET_TYPE** ∗ *result*)

AIO_NUMBER AIOEitherToAIONumber (**AIOEither** ∗ *retval,* **AIORET_TYPE** ∗ *result*)

AIORET_TYPE AIOEitherToAIORetType (**AIOEither** *either*)

24.99 lib/AIOFifo.c File Reference

General structure for AIOUSB Fifo.

```
#include "AIOTypes.h"
#include "AIOFifo.h"
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <assert.h>
#include <stdarg.h>
```

Macros

- #define LOOKUP(T) aioeither_value_ ## T

Functions

- size_t delta (AIOFifo ∗fifo)
- AIORET_TYPE AIOFifoWriteSizeRemaining (void ∗tmpfifo)
- AIORET_TYPE AIOFifoWriteSizeRemainingNumElements (void ∗tmpfifo)
- AIORET_TYPE AIOFifoGetSize (void ∗tmpfifo)
- AIORET_TYPE AIOFifoGetSizeNumElements (void ∗tmpfifo)
- size_t rdelta (AIOFifo ∗fifo)
- AIORET_TYPE AIOFifoReadSize (void ∗tmpfifo)

- AIORET_TYPE AIOFifoReadSizeNumElements (void ∗tmpfifo)
- AIORET_TYPE AIOFifoResize (AIOFifo ∗fifo, size_t newsize)
- void AIOFifoInitialize (AIOFifo ∗nfifo, unsigned int size, unsigned refsize)
- AIOFifo ∗ NewAIOFifo (unsigned int size, unsigned refsize)
- void AIOFifoAllOrNoneInitialize (AIOFifo ∗nfifo, unsigned int size, unsigned refsize)
- AIOFifo ∗ NewAIOFifoAllOrNone (unsigned int size, unsigned refsize)
- void AIOFifoReset (void ∗tmpfifo)
- AIORET_TYPE AIOFifoGetRefSize (void ∗tmpfifo)
- AIORET_TYPE Push (AIOFifoTYPE ∗fifo, TYPE a)
- AIORET_TYPE PushN (AIOFifoTYPE ∗fifo, INPUT_TYPE ∗a, unsigned N)
- AIOEither Pop (AIOFifoTYPE ∗fifo)
- AIORET_TYPE PopN (AIOFifoTYPE ∗fifo, INPUT_TYPE ∗in, unsigned N)
- AIOFifoTYPE ∗ NewAIOFifoTYPE (unsigned int size)
- void DeleteAIOFifoTYPE (AIOFifoTYPE ∗fifo)
- void DeleteAIOFifo (AIOFifo ∗fifo)
- size_t increment (AIOFifo ∗fifo, size_t idx)
- AIORET_TYPE AIOFifoRead (AIOFifo ∗fifo, void ∗tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWrite (AIOFifo ∗fifo, void ∗frombuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWriteAllOrNone (AIOFifo ∗fifo, void ∗frombuf, unsigned maxsize)

  for AllOrNoneTesting
- AIORET_TYPE AIOFifoReadAllOrNone (AIOFifo ∗fifo, void ∗tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoReadPosition (void ∗nfifo)
- AIORET_TYPE AIOFifoWritePosition (void ∗nfifo)
- TEMPLATE_AIOFIFO_API (Counts, uint16_t)
- TEMPLATE_AIOFIFO_API (Volts, double)

24.99.1 Detailed Description

General structure for AIOUSB Fifo.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.99.2 Macro Definition Documentation

#define LOOKUP(*T*) aioeither_value_ ## T

24.99.3 Function Documentation

size_t delta (AIOFifo ∗ *fifo*)

AIORET_TYPE AIOFifoWriteSizeRemaining (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoWriteSizeRemainingNumElements (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoGetSize (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoGetSizeNumElements (void ∗ *tmpfifo*)

size_t rdelta (AIOFifo ∗ *fifo*)

AIORET_TYPE AIOFifoReadSize (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoReadSizeNumElements (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoResize (AIOFifo ∗ *fifo*, size_t *newsize*)

void AIOFifoInitialize (AIOFifo ∗ *nfifo*, unsigned int *size*, unsigned *refsize*)

AIOFifo∗ NewAIOFifo (unsigned int *size*, unsigned *refsize*)

void AIOFifoAllOrNoneInitialize (AIOFifo ∗ *nfifo*, unsigned int *size*, unsigned *refsize*)

AIOFifo∗ NewAIOFifoAllOrNone (unsigned int *size*, unsigned *refsize*)

void AIOFifoReset (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoGetRefSize (void ∗ *tmpfifo*)

AIORET_TYPE Push (AIOFifoTYPE ∗ *fifo*, TYPE *a*)

AIORET_TYPE PushN (AIOFifoTYPE ∗ *fifo*, INPUT_TYPE ∗ *a*, unsigned *N*)

AIOEither Pop (AIOFifoTYPE ∗ *fifo*)

AIORET_TYPE PopN (AIOFifoTYPE ∗ *fifo*, INPUT_TYPE ∗ *in*, unsigned *N*)

AIOFifoTYPE∗ NewAIOFifoTYPE (unsigned int *size*)

void DeleteAIOFifoTYPE (AIOFifoTYPE ∗ *fifo*)

void DeleteAIOFifo (AIOFifo ∗ *fifo*)

size_t increment (AIOFifo ∗ *fifo*, size_t *idx*)

AIORET_TYPE AIOFifoRead (AIOFifo ∗ *fifo*, void ∗ *tobuf*, unsigned *maxsize*)

AIORET_TYPE AIOFifoWrite (AIOFifo ∗ *fifo*, void ∗ *frombuf*, unsigned *maxsize*)

AIORET_TYPE AIOFifoWriteAllOrNone (AIOFifo ∗ *fifo*, void ∗ *frombuf*, unsigned *maxsize*)

for AllOrNoneTesting

AIORET_TYPE AIOFifoReadAllOrNone (AIOFifo ∗ *fifo*, void ∗ *tobuf*, unsigned *maxsize*)

AIORET_TYPE AIOFifoReadPosition (void ∗ *nfifo*)

AIORET_TYPE AIOFifoWritePosition (void ∗ *nfifo*)

TEMPLATE_AIOFIFO_API (Counts , uint16_t)

TEMPLATE_AIOFIFO_API (Volts , double)

24.100 lib/AIOFifo.h File Reference

```
#include "AIOTypes.h"
#include "AIOEither.h"
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
```

Data Structures

- struct AIOFifo

  AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition.
- struct new_aio_fifo

Macros

- #define LOCKING_MECHANISM ;
- #define GRAB_RESOURCE(obj) ;
- #define RELEASE_RESOURCE(obj) ;
- #define AIO_FIFO_INTERFACE
- #define TEMPLATE_AIOFIFO_INTERFACE(NAME, TYPE)
- #define TEMPLATE_AIOFIFO_API(NAME, TYPE)

Typedefs

- typedef struct AIOFifo AIOFifo

  AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition.
- typedef uint32_t TYPE
- typedef void INPUT_TYPE
- typedef struct new_aio_fifo AIOFifoTYPE

Functions

- TEMPLATE_AIOFIFO_INTERFACE (Counts, uint16_t)

  Counts Fifo definition that is a Fifo of 2 byte count values.
- TEMPLATE_AIOFIFO_INTERFACE (Volts, double)

  Volts Fifo definition that is a Fifo of 8 byte double values that will be analog voltage readings.
- AIOFifo ∗ NewAIOFifo (unsigned int size, unsigned int refsize)
- void DeleteAIOFifo (AIOFifo ∗fifo)
- void AIOFifoReset (void ∗fifo)
- AIORET_TYPE AIOFifoRead (AIOFifo ∗fifo, void ∗tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWrite (AIOFifo ∗fifo, void ∗frombuf, unsigned maxsize)
- AIORET_TYPE AIOFifoWriteAllOrNone (AIOFifo ∗fifo, void ∗frombuf, unsigned maxsize)

  for AllOrNoneTesting
- AIORET_TYPE AIOFifoReadAllOrNone (AIOFifo ∗fifo, void ∗tobuf, unsigned maxsize)
- AIORET_TYPE AIOFifoGetRefSize (void ∗fifo)
- AIOFifoTYPE ∗ NewAIOFifoTYPE (unsigned int size)
- AIORET_TYPE Push (AIOFifoTYPE ∗fifo, TYPE a)
- AIORET_TYPE PushN (AIOFifoTYPE ∗fifo, INPUT_TYPE ∗a, unsigned N)
- AIORET_TYPE PopN (AIOFifoTYPE ∗fifo, INPUT_TYPE ∗a, unsigned N)
- AIORET_TYPE AIOFifoWriteSizeRemaining (void ∗fifo)
- AIORET_TYPE AIOFifoWriteSizeRemainingNumElements (void ∗fifo)
- AIORET_TYPE AIOFifoReadSize (void ∗tmpfifo)
- AIORET_TYPE AIOFifoReadSizeNumElements (void ∗tmpfifo)
- AIORET_TYPE AIOFifoGetSize (void ∗fifo)
- AIORET_TYPE AIOFifoGetSizeNumElements (void ∗tmpfifo)
- AIORET_TYPE AIOFifoResize (AIOFifo ∗fifo, size_t newsize)
- AIORET_TYPE AIOFifoReadPosition (void ∗nfifo)
- AIORET_TYPE AIOFifoWritePosition (void ∗nfifo)

24.100.1 Macro Definition Documentation

#define LOCKING_MECHANISM ;

#define GRAB_RESOURCE(*obj*) ;

#define RELEASE_RESOURCE(*obj*) ;

#define AIO_FIFO_INTERFACE

Value:

```
void *data;                                                                   \
    unsigned int refsize;                                                         \
    unsigned int size;                                                            \
    volatile unsigned int read_pos;                                               \
    volatile unsigned int write_pos;
      \
    AIO_EITHER_TYPE kind;
      \
    AIORET_TYPE (*Read)( struct AIOFifo *fifo, void *tobuf, unsigned maxsize );
      \
    AIORET_TYPE (*Write)( struct AIOFifo *fifo, void *tobuf, unsigned maxsize );
      \
    void (*Reset)( void *fifo );                                              \
    size_t (*delta)( struct AIOFifo *fifo  );                                 \
    size_t (*rdelta)( struct AIOFifo *fifo  );                                \
    size_t (*_calculate_size_write)( struct AIOFifo *fifo, unsigned maxsize );        \
    size_t (*_calculate_size_read)( struct AIOFifo *fifo, unsigned maxsize );
```

#define TEMPLATE_AIOFIFO_INTERFACE(***NAME,*** **TYPE)**

Value:

```
typedef struct new_aio_fifo_##NAME {
      \
        AIO_FIFO_INTERFACE;
                          \
        LOCKING_MECHANISM;
                         \
        AIORET_TYPE (*Push)( struct new_aio_fifo_##NAME *fifo,
      TYPE a );                                   \
        AIORET_TYPE (*PushN)( struct new_aio_fifo_##NAME *fifo,
      TYPE *a, unsigned N );                     \
        AIOEither (*Pop)( struct new_aio_fifo_##NAME *fifo );
                   \
        AIORET_TYPE (*PopN)( struct new_aio_fifo_##NAME *fifo ,
      TYPE *a, unsigned N );                     \
    } AIOFifo##NAME;
      \
    AIOFifo##NAME *NewAIOFifo##NAME( unsigned int size );
                    \
    void DeleteAIOFifo##NAME( AIOFifo##NAME *fifo );
                   \
    AIORET_TYPE AIOFifo##NAME ##Initialize( AIOFifo##NAME *nfifo );
```

#define TEMPLATE_AIOFIFO_API(***NAME,*** **TYPE)**

24.100.2 Typedef Documentation

typedef struct AIOFifo AIOFifo

AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition.

The definition of the structure is comprised of the base interface (created wtih a #define) in AIO_FIFO_INTERFACE which handles the basic read and writing to the fifo. In addition , it also includes the Interface called LOCKING_MECHANISM, that makes sure that a write access to the FIFO is atomic.

typedef uint32_t TYPE

typedef void INPUT_TYPE

typedef struct new_aio_fifo AIOFifoTYPE

24.100.3 Function Documentation

TEMPLATE_AIOFIFO_INTERFACE (Counts , uint16_t)

Counts Fifo definition that is a Fifo of 2 byte count values.

TEMPLATE_AIOFIFO_INTERFACE (Volts , double)

Volts Fifo definition that is a Fifo of 8 byte double values that will be analog voltage readings.

AIOFifo∗ NewAIOFifo (unsigned int *size,* unsigned int *refsize*)

void DeleteAIOFifo (AIOFifo ∗ *fifo*)

void AIOFifoReset (void ∗ *fifo*)

AIORET_TYPE AIOFifoRead (AIOFifo ∗ *fifo,* void ∗ *tobuf,* unsigned *maxsize*)

AIORET_TYPE AIOFifoWrite (AIOFifo ∗ *fifo,* void ∗ *frombuf,* unsigned *maxsize*)

AIORET_TYPE AIOFifoWriteAllOrNone (AIOFifo ∗ *fifo,* void ∗ *frombuf,* unsigned *maxsize*)

for AllOrNoneTesting

AIORET_TYPE AIOFifoReadAllOrNone (AIOFifo ∗ *fifo,* void ∗ *tobuf,* unsigned *maxsize*)

AIORET_TYPE AIOFifoGetRefSize (void ∗ *fifo*)

AIOFifoTYPE∗ NewAIOFifoTYPE (unsigned int *size*)

AIORET_TYPE Push (AIOFifoTYPE ∗ *fifo,* TYPE *a*)

AIORET_TYPE PushN (AIOFifoTYPE ∗ *fifo,* INPUT_TYPE ∗ *a,* unsigned *N*)

AIORET_TYPE PopN (AIOFifoTYPE ∗ *fifo,* INPUT_TYPE ∗ *a,* unsigned *N*)

AIORET_TYPE AIOFifoWriteSizeRemaining (void ∗ *fifo*)

AIORET_TYPE AIOFifoWriteSizeRemainingNumElements (void ∗ *fifo*)

AIORET_TYPE AIOFifoReadSize (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoReadSizeNumElements (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoGetSize (void ∗ *fifo*)

AIORET_TYPE AIOFifoGetSizeNumElements (void ∗ *tmpfifo*)

AIORET_TYPE AIOFifoResize (AIOFifo ∗ *fifo,* size_t *newsize*)

AIORET_TYPE AIOFifoReadPosition (void ∗ *nfifo*)

AIORET_TYPE AIOFifoWritePosition (void ∗ *nfifo*)

24.101 lib/AIOList.c File Reference

```
#include "AIOTypes.h"
#include <sys/queue.h>
#include <stdio.h>
#include <string.h>
#include "AIOList.h"
```

Functions

- char ∗ intToString (int val)
- AIORET_TYPE Deleteint (int val)

  Dummy function to make tail q list work.
- TAIL_Q_LIST_IMPLEMENTATION (CStringArray ∗, CStringArray_p)
- TAIL_Q_LIST_IMPLEMENTATION (int, int)
- int intlistFirst (intlist ∗list)
- intlist ∗ Newintlist ()
- AIORET_TYPE Deleteintlist (intlist ∗list)
- char ∗ intlistToString (intlist ∗list)
- int intlistSize (intlist ∗list)
- AIORET_TYPE intlistInsert (intlist ∗list, int tmpval)

Variables

- int newone

24.101.1 Function Documentation

char∗ intToString (int *val*)

AIORET_TYPE Deleteint (int *val*)

Dummy function to make tail q list work.

TAIL_Q_LIST_IMPLEMENTATION (CStringArray ∗ , CStringArray_p)

TAIL_Q_LIST_IMPLEMENTATION (int , int)

int intlistFirst (intlist ∗ *list*)

intlist∗ Newintlist ( )

AIORET_TYPE Deleteintlist (intlist ∗ *list*)

char∗ intlistToString (intlist ∗ *list*)

int intlistSize (intlist ∗ *list*)

AIORET_TYPE intlistInsert (intlist ∗ *list*, int *tmpval*)

24.101.2 Variable Documentation

int newone

24.102 lib/AIOList.h File Reference

```
#include "AIOTypes.h"
#include "CStringArray.h"
#include <sys/queue.h>
```

Macros

- #define TAIL_Q_LIST_TYPE(PRETTYNAME) TailQList ## PRETTYNAME
- #define TAIL_Q_LIST_ENTRY_TYPE(PRETTYNAME) TailQListEntry ## PRETTYNAME
- #define TAIL_Q_LIST(TYPE, PRETTYNAME)
- #define TAIL_Q_LIST_IMPLEMENTATION(TYPE, PRETTYNAME)
- #define foreach_int(J, ILIST) for (intlistentry ∗_ ## IVAL = TailQListintFirst(ILIST) ; _ ## IVAL && (J = _##IVAL->_value); _ ## IVAL = _ ## IVAL->entries.tqe_next)
- #define foreach_CStringArray_p(J, ILIST) for (CStringArray_plistentry ∗_ ## IVAL = TailQListCStringArray_pFirst(ILIST) ; _ ## IVAL && (J = _##IVAL->_value); _ ## IVAL = _ ## IVAL->entries.tqe_next)

Functions

- TAIL_Q_LIST (int, int)
- TAIL_Q_LIST (CStringArray ∗, CStringArray_p)
- typedef TAIL_Q_LIST_TYPE (int) intlist
- typedef TAIL_Q_LIST_ENTRY_TYPE (int) intlistentry
- intlist ∗ Newintlist ()
- AIORET_TYPE Deleteintlist (intlist ∗list)
- char ∗ intlistToString (intlist ∗list)
- int intlistSize (intlist ∗list)
- int intlistFirst (intlist ∗list)
- AIORET_TYPE intlistInsert (intlist ∗list, int tmpval)

24.102.1 Macro Definition Documentation

#define TAIL_Q_LIST_TYPE(*PRETTYNAME*) TailQList ## PRETTYNAME

#define TAIL_Q_LIST_ENTRY_TYPE(*PRETTYNAME*) TailQListEntry ## PRETTYNAME

#define TAIL_Q_LIST(TYPE, *PRETTYNAME*)

Value:

```
typedef struct TailQListEntry ## PRETTYNAME {
      \
        TYPE _value;
                  \
        TAILQ_ENTRY(TailQListEntry ## PRETTYNAME) entries;                                   \
    } TailQListEntry ##PRETTYNAME;                                                           \
    struct tailhead ## PRETTYNAME { struct TailQListEntry ## PRETTYNAME *tqh_first;          \
                                         struct TailQListEntry ## PRETTYNAME **tqh_last; };  \
                                                                                              \
    typedef struct TailQList ## PRETTYNAME {                                                  \
        struct TailQListEntry ## PRETTYNAME _list;                                            \
        int _size;                                                                            \
        /* New stuff */                                                                       \
        struct tailhead ## PRETTYNAME head;                                                   \
        struct tailhead ## PRETTYNAME *headp;                                                 \
    } TailQList ##PRETTYNAME;
      \
    PUBLIC_EXTERN TailQList ## PRETTYNAME  *NewTailQList ## PRETTYNAME();
                     \
    TailQListEntry ## PRETTYNAME *NewTailQListEntry ## PRETTYNAME( TYPE value );
        \
    AIORET_TYPE DeleteTailQListEntry ## PRETTYNAME( TailQListEntry ## PRETTYNAME *entry );
                   \
    int TailQList ## PRETTYNAME ## Size( TailQList ## PRETTYNAME  *list );                    \
    TailQListEntry ## PRETTYNAME * TailQList ## PRETTYNAME ## First( TailQList ## PRETTYNAME *list );    \
    TailQListEntry ## PRETTYNAME * TailQList ## PRETTYNAME ## Last( TailQList ## PRETTYNAME *list );
      \
    TYPE TailQList ## PRETTYNAME ## LastValue( TailQList ## PRETTYNAME *list );
            \
    TYPE TailQListEntry ## PRETTYNAME ## To ##PRETTYNAME( TailQListEntry ## PRETTYNAME *entry );
            \
    char *TailQListEntry ## PRETTYNAME ## ToString( TailQListEntry ## PRETTYNAME *entry );              \
    char *TailQList ## PRETTYNAME ## ToString( TailQList ## PRETTYNAME  *list );
      \
    AIORET_TYPE DeleteTailQList ## PRETTYNAME ( TailQList ## PRETTYNAME  *list );
                   \
    AIORET_TYPE TailQList ## PRETTYNAME ## Insert( TailQList ## PRETTYNAME  *list,
                   \
                                                   TailQListEntry ## PRETTYNAME *nnode );
```

#define TAIL_Q_LIST_IMPLEMENTATION(TYPE, *PRETTYNAME*)

#define foreach_int(*J*, *ILIST*) for (intlistentry ∗_ ## IVAL = TailQListintFirst(ILIST) ; _ ## IVAL && (J = _##IVAL->_value); _ ## IVAL = _ ## IVAL->entries.tqe_next)

#define foreach_CStringArray_p(*J*, *ILIST*) for (CStringArray_plistentry ∗_ ## IVAL = TailQListCStringArray_pFirst(ILIST) ; _ ## IVAL && (J = _##IVAL->_value); _ ## IVAL = _ ## IVAL->entries.tqe_next)

24.102.2 Function Documentation

TAIL_Q_LIST (int , int)

TAIL_Q_LIST (CStringArray ∗ , CStringArray_p)

typedef TAIL_Q_LIST_TYPE (int)

typedef TAIL_Q_LIST_ENTRY_TYPE (int)

intlist∗ Newintlist ( )

AIORET_TYPE Deleteintlist (intlist ∗ *list*)

char∗ intlistToString (intlist ∗ *list*)

int intlistSize (intlist ∗ *list*)

int intlistFirst (intlist ∗ *list*)

AIORET_TYPE intlistInsert (intlist ∗ *list*, int *tmpval*)

24.103 lib/AIOProductTypes.c File Reference

```
#include "AIOProductTypes.h"
#include <stdarg.h>
#include <string.h>
```

Functions

- AIO_PRODUCT_CONSTANT (AIO_ANALOG_INPUT_OBJ, AIO_ANALOG_INPUT_GROUP, AIO_ANALOG_INPUT, 2, AIO_RANGE(USB_AI16_16A, USB_AI12_128E),)
- AIO_PRODUCT_CONSTANT (AIO_ANALOG_OUTPUT_OBJ, AIO_ANALOG_OUTPUT_GROUP, AIO_ANALOG_OUTPUT, 2, AIO_RANGE(USB_AO16_16A, USB_AIO12_128E),)
- AIO_PRODUCT_CONSTANT (AIO_DIGITAL_HIGHSPEED_OBJ, AIO_DIGITAL_HIGHSPEED_GROUP, AIO_DIGITAL_HIGHSPEED, 1,)

- AIO_PRODUCT_CONSTANT (AIO_ANALOG_CLOCK_100KHZ_OBJ, AIO_ANALOG_CLOCK_100KHZ_GRO-UP, AIO_ANALOG_CLOCK_100KHZ, 2, AIO_RANGE(USB_AI12_16E, USB_AI12_16E),)
- AIO_PRODUCT_CONSTANT (AIO_ANALOG_CLOCK_250KHZ_OBJ, AIO_ANALOG_CLOCK_250KHZ_GRO-UP, AIO_ANALOG_CLOCK_250KHZ, 2, AIO_RANGE(USB_AI12_16E, USB_AI12_16E),)
- AIO_PRODUCT_CONSTANT (AIO_ANALOG_CLOCK_500KHZ_OBJ, AIO_ANALOG_CLOCK_500KHZ_GRO-UP, AIO_ANALOG_CLOCK_500KHZ, 2, AIO_RANGE(USB_AI12_16A, USB_AI12_16E),)
- AIO_PRODUCT_CONSTANT (AIO_ANALOG_12BIT_A2D_OBJ, AIO_ANALOG_12BIT_A2D_GROUP, AIO_AI-O_ANALOG_12BIT_A2D, 4, AIO_RANGE(USB_AI12_16A, USB_AI12_16E), AIO_RANGE(USB_AO12_16A, U-SB_AO12_16A), AIO_RANGE(USB_AO12_8A, USB_AO12_8A),)
- AIO_PRODUCT_CONSTANT (AIO_ANALOG_16BIT_A2D_OBJ, AIO_ANALOG_16BIT_A2D_GROUP, AIO_AI-O_ANALOG_16BIT_A2D, 5, AIO_RANGE(USB_AI16_16A, USB_AI12_16), AIO_RANGE(USB_AO16_16A, U-SB_AO16_16A), AIO_RANGE(USB_AO16_8A, USB_AO16_8A), AIO_RANGE(USB_AO16_4A, USB_AO16_4-A),)
- AIOProductRange ∗ NewAIOProductRange (unsigned long start, unsigned long end)
- AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗pr)
- AIOProductGroup ∗ NewAIOProductGroup (size_t numbergroups,...)
- AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗pg)
- AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗g, unsigned long val)
- AIOProductGroup ∗ groupcpy (const AIOProductGroup ∗g)

24.103.1 Function Documentation

AIO_PRODUCT_CONSTANT (AIO_ANALOG_INPUT_OBJ , AIO_ANALOG_INPUT_GROUP , AIO_ANALOG_INPUT , 2 , AIO_RANGE(USB_AI16_16A, USB_AI12_128E))

AIO_PRODUCT_CONSTANT (AIO_ANALOG_OUTPUT_OBJ , AIO_ANALOG_OUTPUT_GROUP , AIO_ANALOG_OUTPUT , 2 , AIO_RANGE(USB_AO16_16A, USB_AIO12_128E))

AIO_PRODUCT_CONSTANT (AIO_DIGITAL_HIGHSPEED_OBJ , AIO_DIGITAL_HIGHSPEED_GROUP , AIO_DIGITAL_HIGHSPEED , 1)

AIO_PRODUCT_CONSTANT (AIO_ANALOG_CLOCK_100KHZ_OBJ , AIO_ANALOG_CLOCK_100KHZ_GROUP , AIO_ANALOG_CLOCK_100KHZ , 2 , AIO_RANGE(USB_AI12_16E, USB_AI12_16E))

AIO_PRODUCT_CONSTANT (AIO_ANALOG_CLOCK_250KHZ_OBJ , AIO_ANALOG_CLOCK_250KHZ_GROUP , AIO_ANALOG_CLOCK_250KHZ , 2 , AIO_RANGE(USB_AI12_16E, USB_AI12_16E))

AIO_PRODUCT_CONSTANT (AIO_ANALOG_CLOCK_500KHZ_OBJ , AIO_ANALOG_CLOCK_500KHZ_GROUP , AIO_ANALOG_CLOCK_500KHZ , 2 , AIO_RANGE(USB_AI12_16A, USB_AI12_16E))

AIO_PRODUCT_CONSTANT (AIO_ANALOG_12BIT_A2D_OBJ , AIO_ANALOG_12BIT_A2D_GROUP , AIO_AIO_ANALOG_12BIT_A2D , 4 , AIO_RANGE(USB_AI12_16A, USB_AI12_16E) , AIO_RANGE(USB_AO12_16A, USB_AO12_16A) , AIO_RANGE(USB_AO12_8A, USB_AO12_8A))

AIO_PRODUCT_CONSTANT (AIO_ANALOG_16BIT_A2D_OBJ , AIO_ANALOG_16BIT_A2D_GROUP , AIO_AIO_ANALOG_16BIT_A2D , 5 , AIO_RANGE(USB_AI16_16A, USB_AI12_16) , AIO_RANGE(USB_AO16_16A, USB_AO16_16A) , AIO_RANGE(USB_AO16_8A, USB_AO16_8A) , AIO_RANGE(USB_AO16_4A, USB_AO16_4A))

AIOProductRange∗ NewAIOProductRange (unsigned long *start,* unsigned long *end*)

AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗ *pr*)

AIOProductGroup∗ NewAIOProductGroup (size_t *numbergroups,* ...)

AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗ *pg*)

AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗ *g,* unsigned long *val*)

AIOProductGroup∗ groupcpy (const AIOProductGroup ∗ *g*)

24.104 lib/AIOProductTypes.h File Reference

```
#include "AIOTypes.h"
```

Data Structures

- struct AIOProductRange

    A simplified range of Products based off of device ids.
- struct AIOProductGroup

    A smart product group that marks a range of ACCES I/O Products.

Macros

- #define NUMARGS(...) (sizeof((void∗[ ]){__VA_ARGS__})/sizeof(void∗))
- #define AIO_RANGE(start, stop) (&(AIOProductRange){ ._start=start, ._end =stop })
- #define AIO_PRODUCT_GROUP(NAME, N,...) const AIOProductGroup NAME = { ._num_groups =N, ._groups = (AIOProductRange ∗∗)&(AIOProductRange ∗[N]){ __VA_ARGS__ } } ;
- #define AIO_PRODUCT_CONSTANT(NAME, NAMEPTR, NAMEFN, N,...)
- #define AIO_PRODUCT_EXTERN(NAME, NAMEPTR, NAMEFN)

Typedefs

- typedef struct AIOProductRange AIOProductRange

    A simplified range of Products based off of device ids.
- typedef struct AIOProductGroup AIOProductGroup

    A smart product group that marks a range of ACCES I/O Products.

Functions

- AIOProductRange ∗ NewAIOProductRange (unsigned long start, unsigned long end)
- AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗pr)
- AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗pr)
- AIOProductGroup ∗ NewAIOProductGroup (size_t numgroups,...)
- AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗)
- AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗g, unsigned long val)
- AIOProductGroup ∗ groupcpy (const AIOProductGroup ∗g)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_OUTPUT_OBJ, AIO_ANALOG_OUTPUT_GROUP, AIO_ANALOG_-OUTPUT)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_INPUT_OBJ, AIO_ANALOG_INPUT_GROUP, AIO_ANALOG_INPU-T)
- AIO_PRODUCT_EXTERN (AIO_DIGITAL_HIGHSPEED_OBJ, AIO_DIGITAL_HIGHSPEED_GROUP, AIO_DIG-ITAL_HIGHSPEED)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_100KHZ_OBJ, AIO_ANALOG_CLOCK_100KHZ_GROUP, AIO_ANALOG_CLOCK_100KHZ)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_250KHZ_OBJ, AIO_ANALOG_CLOCK_250KHZ_GROUP, AIO_ANALOG_CLOCK_250KHZ)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_500KHZ_OBJ, AIO_ANALOG_CLOCK_500KHZ_GROUP, AIO_ANALOG_CLOCK_500KHZ)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_12BIT_A2D_OBJ, AIO_ANALOG_12BIT_A2D_GROUP, AIO_AIO_-ANALOG_12BIT_A2D)
- AIO_PRODUCT_EXTERN (AIO_ANALOG_16BIT_A2D_OBJ, AIO_ANALOG_16BIT_A2D_GROUP, AIO_AIO_-ANALOG_16BIT_A2D)

24.104.1 Macro Definition Documentation

#define NUMARGS(*...*) (sizeof((void∗[ ]){__VA_ARGS__})/sizeof(void∗))

#define AIO_RANGE(*start, stop*) (&(AIOProductRange){ ._start=start, ._end =stop })

#define AIO_PRODUCT_GROUP(*NAME, N, ...*) const AIOProductGroup NAME = { ._num_groups =N, ._groups = (AIOProductRange ∗∗)&(AIOProductRange ∗[N]){ __VA_ARGS__ } } ;

#define AIO_PRODUCT_CONSTANT(*NAME, NAMEPTR, NAMEFN, N, ...*)

Value:

```
AIO_PRODUCT_GROUP(NAME,N, __VA_ARGS__); \
                                                          const
      AIOProductGroup *NAMEPTR = &NAME; \
                                                          AIOProductGroup
       *NAMEFN() { return groupcpy( NAMEPTR );}
```

#define AIO_PRODUCT_EXTERN(***NAME,*** ***NAMEPTR,*** ***NAMEFN*** **)**

Value:

```
extern const AIOProductGroup NAME; \
                                                    extern const
      AIOProductGroup *NAMEPTR;\
                                                    extern AIOProductGroup *NAMEFN();
```

24.104.2 Typedef Documentation

typedef struct AIOProductRange AIOProductRange

A simplified range of Products based off of device ids.

typedef struct AIOProductGroup AIOProductGroup

A smart product group that marks a range of ACCES I/O Products.

24.104.3 Function Documentation

AIOProductRange∗ NewAIOProductRange (unsigned long *start*, unsigned long *end*)

AIORET_TYPE DeleteAIOProductRange (AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeStart (const AIOProductRange ∗ *pr*)

AIORET_TYPE AIOProductRangeEnd (const AIOProductRange ∗ *pr*)

AIOProductGroup∗ NewAIOProductGroup (size_t *numgroups*, ...)

AIORET_TYPE DeleteAIOProductGroup (AIOProductGroup ∗)

AIORET_TYPE AIOProductGroupContains (const AIOProductGroup ∗ *g*, unsigned long *val*)

AIOProductGroup∗ groupcpy (const AIOProductGroup ∗ *g*)

AIO_PRODUCT_EXTERN (AIO_ANALOG_OUTPUT_OBJ , AIO_ANALOG_OUTPUT_GROUP , AIO_ANALOG_OUTPUT)

AIO_PRODUCT_EXTERN (AIO_ANALOG_INPUT_OBJ , AIO_ANALOG_INPUT_GROUP , AIO_ANALOG_INPUT)

AIO_PRODUCT_EXTERN (AIO_DIGITAL_HIGHSPEED_OBJ , AIO_DIGITAL_HIGHSPEED_GROUP , AIO_DIGITAL_HIGHSPEED)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_100KHZ_OBJ , AIO_ANALOG_CLOCK_100KHZ_GROUP , AIO_ANALOG_CLOCK_100KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_250KHZ_OBJ , AIO_ANALOG_CLOCK_250KHZ_GROUP , AIO_ANALOG_CLOCK_250KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_CLOCK_500KHZ_OBJ , AIO_ANALOG_CLOCK_500KHZ_GROUP , AIO_ANALOG_CLOCK_500KHZ)

AIO_PRODUCT_EXTERN (AIO_ANALOG_12BIT_A2D_OBJ , AIO_ANALOG_12BIT_A2D_GROUP , AIO_AIO_ANALOG_12BIT_A2D)

AIO_PRODUCT_EXTERN (AIO_ANALOG_16BIT_A2D_OBJ , AIO_ANALOG_16BIT_A2D_GROUP , AIO_AIO_ANALOG_16BIT_A2D)

24.105 lib/AIOTuple.c File Reference

```
#include "AIOTuple.h"
#include <stdarg.h>
#include <string.h>
```

24.106 lib/AIOTuple.h File Reference

```
#include "AIOTypes.h"
#include "CStringArray.h"
#include <stdio.h>
```

Macros

- #define AIOTUPLE2_TYPE(NAME, T1, T2)
- #define AIO_CHAR_ARRAY(N,...) (char ∗∗)&(char ∗[N]){ __VA_ARGS__ }
- #define AIOTUPLE2_PTR(NAME, T1, T2) NAME ∗
- #define AIOTUPLE2(NAME, T1, T2) NAME
- #define AIOTUPLE2_TO_STR(TYPE, T) TYPE ##ToString(T)

Functions

- AIOTUPLE2_TYPE (AIOTuple2_AIORET_TYPE__CStringArray, AIORET_TYPE, CStringArray)
- AIOTUPLE2_TYPE (AIOTuple2_AIORET_TYPE__CStringArray_p, AIORET_TYPE, CStringArray ∗)
- char ∗ AIOTuple2_AIORET_TYPE__CStringArray_pToString (AIOTuple2_AIORET_TYPE__CStringArray_p ∗type)
- AIORET_TYPE DeleteAIOTuple2_AIORET_TYPE__CStringArray_p (AIOTuple2_AIORET_TYPE__CString-Array_p ∗type)

24.106.1 Macro Definition Documentation

#define AIOTUPLE2_TYPE(*NAME, T1, T2*)

Value:

```
typedef struct NAME {                                      \
        T1 _1;                                                 \
        T2 _2;                                                 \
    } NAME;                                                    \
    T2 NAME ## get_2( NAME *obj ) { return obj->_2 ; };        \
    T1 NAME ## get_1( NAME *obj ) { return obj->_1 ; };
```

#define AIO_CHAR_ARRAY(*N, ...*) (char ∗∗)&(char ∗[N]){ __VA_ARGS__ }

#define AIOTUPLE2_PTR(*NAME, T1, T2*) NAME ∗

#define AIOTUPLE2(*NAME, T1, T2*) NAME

#define AIOTUPLE2_TO_STR(TYPE, *T*) TYPE ##ToString(T)

24.106.2 Function Documentation

AIOTUPLE2_TYPE (AIOTuple2_AIORET_TYPE__CStringArray , AIORET_TYPE , CStringArray)

AIOTUPLE2_TYPE (AIOTuple2_AIORET_TYPE__CStringArray_p , AIORET_TYPE , CStringArray ∗)

char∗ AIOTuple2_AIORET_TYPE__CStringArray_pToString (AIOTuple2_AIORET_TYPE__CStringArray_p ∗ *type*) `[inline]`

AIORET_TYPE DeleteAIOTuple2_AIORET_TYPE__CStringArray_p (AIOTuple2_AIORET_TYPE__CStringArray_p ∗ *type*) `[inline]`

24.107 lib/AIOTypes.h File Reference

```
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>
#include <errno.h>
```

Data Structures

- struct ushort_array
- struct lookup
- struct DeviceProperties

  Allows us to keep track of streaming (bulk) acquires without making the user keep track of the memory management.

Macros

- #define HAS_PTHREAD 1
- #define EXPORTED_FUNCTION
- #define CREATE_ENUM(name,...) typedef enum { name ## _begin, __VA_ARGS__, name ## _end } name;
- #define CREATE_ENUM_W_START(name, num,...) typedef enum { name ## _begin = (num-1), __VA_ARGS__, name ## _end } name;
- #define LAST_ENUM(name) (name ## _end-1)
- #define FIRST_ENUM(name) (name ## _begin+1)
- #define MIN_VALUE(name) (name ## _begin+1)
- #define MAX_VALUE(name) (name ## _end-1)
- #define VALID_ENUM(name, value) (value >= FIRST_ENUM(name) && value <= LAST_ENUM(name))
- #define ERR_UNLESS_VALID_ENUM(name, value) assert((value >= FIRST_ENUM(name) && value <= LAS-T_ENUM(name)))
- #define VALID_PRODUCT(product) (VALID_ENUM(ProductIDS, product))
- #define GCC_VERSION
- #define ACCES_DEPRECATED(FOO) __attribute__ ((deprecated))
- #define LAMBDA(return_type, header, function_body)
- #define MIN(a, b) (((a)<(b))?(a):(b))
- #define MAX(a, b) (((a)>(b))?(a):(b))
- #define AUR_CBUF_SETUP 0x01000007
- #define AUR_CBUF_EXIT 0x00020002
- #define NUMBER_CHANNELS 16

  Simple macro for iterating over objects.
- #define foreach_array(i, ary, size)
- #define AIO_MAKE_ERROR(N) -1∗abs(N)
- #define AIOUSB_ERROR_VALUE(N) -1∗abs(N)
- #define AIO_ASSERT(...) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { errno = -AIOUSB_ERROR_INVALI-D_PARAMETER; return -AIOUSB_ERROR_INVALID_PARAMETER; }
- #define AIO_ASSERT_RET(ret,...) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { return ret; }
- #define AIO_ASSERT_AIORET_TYPE(ret,...) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { errno = -abs(ret); return -abs(ret); }
- #define AIO_ASSERT_NO_RETURN(...) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { return; }
- #define AIO_ASSERT_EXIT(...) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { exit(-AIOUSB_ERROR_INV-ALID_PARAMETER); }
- #define AIO_ASSERT_ERR_NO_RETURN(err,...) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { exit(-err); }
- #define AIO_ASSERT_VALID_DATA(err,...) assert(__VA_ARGS__) ; if (!(__VA_ARGS__)) { return err; }
- #define AIO_ASSERT_USB(...) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_USBDEVICE, __V-A_ARGS__)
- #define AIO_ASSERT_DIOBUF(...) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_DIOBUF, __VA-_ARGS__)
- #define AIO_ASSERT_CHANNELMASK(...) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_AIOC-HANNELMASK, __VA_ARGS__)
- #define AIO_ASSERT_CONFIG(...) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_ADCCONFIG , -__VA_ARGS__)
- #define AIO_ASSERT_AIOCONTBUF(...) AIO_ASSERT_VALID_DATA(-AIOUSB_ERROR_INVALID_AIOCON-TINUOUS_BUFFER, __VA_ARGS__);
- #define AIO_ASSERT_AIOEITHER(err, msg,...) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { AIOEither tmp; tmp.left = err; tmp.errmsg=strdup(msg); return tmp; }
- #define AIO_ERROR(X) (-abs(X))

  AIO_ERROR ∗ are just like the regular ASSERTIONS meaning that they argument should evaluate to true otherwise it will fail.
- #define AIO_ERROR_VALID_DATA(err,...) if (!(__VA_ARGS__)) { return err; }
- #define AIO_ERROR_VALID_DATA_RETVAL(err,...) if (!(__VA_ARGS__)) { return -abs(err); }
- #define AIO_ERROR_VALID_AIORET_TYPE(err,...) if (!(__VA_ARGS__)) { return -abs(err); }
- #define AIO_ERROR_AIOEITHER_VALID_DATA(err,...)
- #define AIO_ERROR_VALID_DATA_W_CODE(err, code,...) if (!(__VA_ARGS__)) { { code; }; return err; }
- #define AIO_ERROR_VALID_DATA_WITH_CODE(retval, err, code) if (! (code)) { errno = -abs(err); return retval; }
- #define G_STMT_START do
- #define G_STMT_END while (0)

- #define G_BREAKPOINT() G_STMT_START{ raise (SIGTRAP); }G_STMT_END
- #define EXIT_FN_IF_NO_VALID_USB(d, r, f, u, g)
- #define AIOUSB_ERROR_OFFSET 100
- #define LIBUSB_RESULT_TO_AIOUSB_RESULT(code) (unsigned long)(AIOUSB_ERROR_OFFSET + -(int)(code))
- #define AIOUSB_RESULT_TO_LIBUSB_RESULT(code) (-((int)(code) - AIOUSB_ERROR_OFFSET))
- #define ROOTCLOCK 10000000

Typedefs

- typedef int64_t AIORET_TYPE
- typedef unsigned long AIORESULT
- typedef unsigned short ∗ COUNTS
- typedef struct ushort_array Ushort_Array
- typedef uint16_t AIOBufferType
- typedef enum AIOUSB_BOOL_VAL AIOUSB_BOOL
- typedef long double AIO_NUMBER
- typedef struct lookup EnumStringLookup

Enumerations

- enum AIO_SCAN_TYPE { AIO_PER_OVERSAMPLE = 1, AIO_PER_CHANNEL, AIO_PER_SCANS }
- enum THREAD_STATUS {
THREAD_STATUS_begin = (-1 -1), INVALID_OBJECT = -2, NOT_STARTED = 0, RUNNING = 1,
WITH_DATA = 2, TERMINATED = 4, RUNNING_OR_WITH_DATA = RUNNING | WITH_DATA, JOINED = 8,
TERMINATED_OVERRUN = 16, TERMINATING = 32, THREAD_STATUS_end }
- enum AIOContinuousBufMode {
AIOContinuousBufMode_begin = (0 -1), AIOCONTINUOUS_BUF_ALLORNONE, AIOCONTINUOUS_BUF_NO-RMAL, AIOCONTINUOUS_BUF_OVERRIDE,
AIOContinuousBufMode_end }
- enum { MAX_USB_DEVICES = 32 }
- enum AIOUSB_BOOL_VAL { AIOUSB_FALSE = 0, AIOUSB_TRUE = 1 }

    other libraries often declare BOOL, TRUE and FALSE, and worse, they declare these using #define; so we sidestep that potential conflict by declaring the same types prefixed with AIOUSB_; it's ugly, but if people want to use the shorter names and they are certain they won't conflict with anything else, they can define the ENABLE_BOOL_TYPE macro
- enum ProductIDS {
ProductIDS_begin = (0 -1), ACCES_VENDOR_ID = 0x1605, USB_DA12_8A_REV_A = 0xC001, USB_DA12_8A = 0xC002,
USB_DA12_8E = 0xC003, USB_DIO_32 = 0x8001, USB_DIO_32I = 0x8004, USB_DIO_48 = 0x8002,
USB_DIO_96 = 0x8003, USB_DIO24_CTR6 = 0x8006, USB_DI16A_REV_A1 = 0x8008, USB_DO16A_REV_A1 = 0x8009,
USB_DI16A_REV_A2 = 0x800a, USB_DIO_16H = 0x800c, USB_DI16A = 0x800d, USB_DO16A = 0x800e,
USB_DIO_16A = 0x800f, USB_IIRO_16 = 0x8010, USB_II_16 = 0x8011, USB_RO_16 = 0x8012,
USB_IIRO_8 = 0x8014, USB_II_8 = 0x8015, USB_IIRO_4 = 0x8016, USB_IDIO_16 = 0x8018,
USB_II_16_OLD = 0x8019, USB_IDO_16 = 0x801a, USB_IDIO_8 = 0x801c, USB_II_8_OLD = 0x801d,
USB_IDIO_4 = 0x801e, USB_CTR_15 = 0x8020, USB_IIRO4_2SM = 0x8030, USB_IIRO4_COM = 0x8031,
USB_DIO16RO8 = 0x8032, USB_DIO48DO24 = 0x803C, USB_DIO24DO12 = 0x803D, USB_DO24 = 0x803E,
PICO_DIO16RO8 = 0x8033, USBP_II8IDO4A = 0x8036, USB_AI16_16A = 0x8040, USB_AI16_16E = 0x8041,
USB_AI12_16A = 0x8042, USB_AI12_16 = 0x8043, USB_AI12_16E = 0x8044, USB_AI16_64MA = 0x8045,
USB_AI16_64ME = 0x8046, USB_AI12_64MA = 0x8047, USB_AI12_64M = 0x8048, USB_AI12_64ME = 0x8049,
USB_AI16_32A = 0x804a, USB_AI16_32E = 0x804b, USB_AI12_32A = 0x804c, USB_AI12_32 = 0x804d,
USB_AI12_32E = 0x804e, USB_AI16_64A = 0x804f, USB_AI16_64E = 0x8050, USB_AI12_64A = 0x8051,
USB_AI12_64 = 0x8052, USB_AI12_64E = 0x8053, USB_AI16_96A = 0x8054, USB_AI16_96E = 0x8055,
USB_AI12_96A = 0x8056, USB_AI12_96 = 0x8057, USB_AI12_96E = 0x8058, USB_AI16_128A = 0x8059,
USB_AI16_128E = 0x805a, USB_AI12_128A = 0x805b, USB_AI12_128 = 0x805c, USB_AI12_128E = 0x805d,
USB_AO_ARB1 = 0x8068, USB_AO16_16A = 0x8070, USB_AO16_16 = 0x8071, USB_AO16_12A = 0x8072,
USB_AO16_12 = 0x8073, USB_AO16_8A = 0x8074, USB_AO16_8 = 0x8075, USB_AO16_4A = 0x8076,
USB_AO16_4 = 0x8077, USB_AO12_16A = 0x8078, USB_AO12_16 = 0x8079, USB_AO12_12A = 0x807a,
USB_AO12_12 = 0x807b, USB_AO12_8A = 0x807c, USB_AO12_8 = 0x807d, USB_AO12_4A = 0x807e,
USB_AO12_4 = 0x807f, USB_AIO16_16A = 0x8140, USB_AIO16_16E = 0x8141, USB_AIO12_16A = 0x8142,
USB_AIO12_16 = 0x8143, USB_AIO12_16E = 0x8144, USB_AIO16_64MA = 0x8145, USB_AIO16_64ME = 0x8146,
USB_AIO12_64MA = 0x8147, USB_AIO12_64M = 0x8148, USB_AIO12_64ME = 0x8149, USB_AIO16_32A = 0x814a,
USB_AIO16_32E = 0x814b, USB_AIO12_32A = 0x814c, USB_AIO12_32 = 0x814d, USB_AIO12_32E = 0x814e,
USB_AIO16_64A = 0x814f, USB_AIO16_64E = 0x8150, USB_AIO12_64A = 0x8151, USB_AIO12_64 = 0x8152,
USB_AIO12_64E = 0x8153, USB_AIO16_96A = 0x8154, USB_AIO16_96E = 0x8155, USB_AIO12_96A = 0x8156,
USB_AIO12_96 = 0x8157, USB_AIO12_96E = 0x8158, USB_AIO16_128A = 0x8159, USB_AIO16_128E =

0x815a,
USB_AIO12_128A = 0x815b, USB_AIO12_128 = 0x815c, USB_AIO12_128E = 0x815d, ProductIDS_end }

- enum { diFirst = 0xFFFFFFFEul, diOnly = 0xFFFFFFFDul, diNone = 0xFFFFFFFFul }
- enum DACRange {
DACRange_begin = (0 -1), DAC_RANGE_0_5V, DAC_RANGE_5V, DAC_RANGE_0_10V,
DAC_RANGE_10V, DACRange_end }

  range codes passed to DACSetBoardRange()
- enum FIFO_Method {
FIFO_Method_begin = (0 -1), CLEAR_FIFO_METHOD_IMMEDIATE, CLEAR_FIFO_METHOD_AUTO, CLEA-R_FIFO_METHOD_IMMEDIATE_AND_ABORT = 5,
CLEAR_FIFO_METHOD_NOW = 0x35, CLEAR_FIFO_METHOD_WAIT = 86, FIFO_Method_end }

  FIFO clearing methods passed to AIOUSB_ClearFIFO()
- enum ResultCode {
ResultCode_begin = (0 -1), AIOUSB_SUCCESS, AIOUSB_ERROR_DEVICE_NOT_CONNECTED, AIOUSB_-ERROR_DUP_NAME,
AIOUSB_ERROR_NOT_INIT, AIOUSB_ERROR_FILE_NOT_FOUND, AIOUSB_ERROR_INVALID_DATA, AI-OUSB_ERROR_INVALID_INDEX,
AIOUSB_ERROR_INVALID_MUTEX, AIOUSB_ERROR_INVALID_PARAMETER, AIOUSB_ERROR_INVALID-_THREAD, AIOUSB_ERROR_NOT_ENOUGH_MEMORY,
AIOUSB_ERROR_INVALID_MEMORY, AIOUSB_ERROR_NOT_SUPPORTED, AIOUSB_ERROR_OPEN_FA-ILED, AIOUSB_ERROR_BAD_TOKEN_TYPE,
AIOUSB_ERROR_TIMEOUT, AIOUSB_ERROR_DIVIDE_BY_ZERO, AIOUSB_ERROR_HANDLE_EOF, AIOU-SB_ERROR_DEVICE_NOT_FOUND,
AIOUSB_ERROR_USBDEVICE_NOT_FOUND, AIOUSB_ERROR_USB_INIT, AIOUSB_ERROR_INVALID_TI-MEOUT, AIOUSB_ERROR_INVALID_AIOEITHER_ALLOCATION,
AIOUSB_ERROR_INVALID_USBDEVICE, AIOUSB_ERROR_INVALID_VOLTAGES, AIOUSB_ERROR_INVA-LID_AIOCMD, AIOUSB_ERROR_INVALID_CALLBACK,
AIOUSB_ERROR_INVALID_COUNTS, AIOUSB_ERROR_INVALID_COUNTS_CONVERTER, AIOUSB_ERR-OR_INVALID_DEVICE, AIOUSB_ERROR_INVALID_DEVICE_SETTING,
AIOUSB_ERROR_INVALID_DEVICE_FUNCTIONAL_PARAMETER, AIOUSB_ERROR_INVALID_DEVICE_S-TREAM_SETTING, AIOUSB_ERROR_INVALID_DEVICE_CHANNEL_SETTING, AIOUSB_ERROR_INVALID_-DEVICE_MUX_CHANNEL_SETTING,
AIOUSB_ERROR_INVALID_CHANNELS_PER_GROUP_SETTING, AIOUSB_ERROR_INVALID_AIOCHANN-ELMASK, AIOUSB_ERROR_INVALID_CONFIG, AIOUSB_ERROR_INVALID_DIOBUF,
AIOUSB_ERROR_INVALID_GAINCODE, AIOUSB_ERROR_INVALID_CALMODE, AIOUSB_ERROR_INVALI-D_CHANNEL_NUMBER, AIOUSB_ERROR_INVALID_AIOCONFIGURATION,
AIOUSB_ERROR_INVALID_AIOARGUMENT, AIOUSB_ERROR_INVALID_AIODEVICE_QUERY, AIOUSB_E-RROR_INVALID_AIOEITHER, AIOUSB_ERROR_INVALID_AIOFIFO,
AIOUSB_ERROR_INVALID_ADCCONFIG, AIOUSB_ERROR_INVALID_ADCCONFIG_SIZE, AIOUSB_ERRO-R_INVALID_ADCCONFIG_SETTING, AIOUSB_ERROR_INVALID_ADCCONFIG_TRIGGER_SETTING,
AIOUSB_ERROR_INVALID_ADCCONFIG_CAL_SETTING, AIOUSB_ERROR_INVALID_ADCCONFIG_CHAN-NEL_SETTING, AIOUSB_ERROR_INVALID_ADCCONFIG_OVERSAMPLE_SETTING, AIOUSB_ERROR_IN-VALID_ADCCONFIG_REGISTER_SETTING,
AIOUSB_ERROR_INVALID_ADCCONFIG_MUX_SETTING, AIOUSB_ERROR_INVALID_ADCCONFIG_DEVI-CE, AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER, AIOUSB_ERROR_INVALID_AIOCONTINUOU-S_BUFFER_NUM_CHANNELS,
AIOUSB_ERROR_INVALID_AIOBUFTYPE, AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_CHANNEL_RA-NGE, AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_NUM_CHANNELS, AIOUSB_ERROR_AIOCOMMAN-DLINE_INVALID_INDEX_NUM,
AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_START_END_CHANNEL, AIOUSB_ERROR_AIOCOMMAN-DLINE_HELP, AIOUSB_ERROR_INVALID_LIBUSB_DEVICE_HANDLE, AIOUSB_FIFO_COPY_ERROR,
AIOUSB_ERROR_LIBUSB, ResultCode_end }

  The AIOUSB function result codes are a bit confusing; the result codes used in the Windows implementation of the API are defined in a system file, winerror.h; these result codes are generic and can apply to many applications; the very first result code, ERROR_SUCCESS, sounds like an oxymoron; the result codes used in libusb, on the other hand, are a lot more appealing; the result code for success is LIBUSB_SUCCESS; the result codes for errors are LIBUSB_ERROR_-xxx; further complicating matters is that the AIOUSB result codes must be non-negative since all the functions return an unsigned result, whereas the LIBUSB result codes are negative in the case of errors; both schemes use zero to denote success; it would also be nice to return the original libusb result code in cases where a libusb error causes an AIOUSB API function to fail; so to satisfy all these requirements, we've employed the following scheme:
- enum { AD_MAX_CHANNELS = 128, AD_GAIN_CODE_MASK = 7 }
- enum ADRegister {
ADRegister_begin = (16 -1), AD_REGISTER_CAL_MODE, AD_REGISTER_TRIG_COUNT, AD_REGISTER_-START_END,
AD_REGISTER_OVERSAMPLE, AD_REGISTER_MUX_START_END, ADRegister_end }
- enum {
AD_MAX_CONFIG_REGISTERS = 21, AD_MIN_CONFIG_REGISTERS = 20, AD_MAX_TIMEOUT = 8000, A-D_MIN_TIMEOUT = 500,
AD_NUM_GAIN_CODE_REGISTERS = 16, AD_CONFIG_GAIN_CODE = 0, AD_REGISTER_GAIN_CODE = 0, AD_CONFIG_CAL_MODE = 0x10,
AD_CONFIG_TRIG_COUNT = 0x11, AD_CONFIG_START_END = 0x12, AD_CONFIG_OVERSAMPLE = 0x13,

AD_CONFIG_MUX_START_END = 0x14,
AD_CONFIG_START_STOP_CHANNEL_EX = 21, AD_NUM_GAIN_CODES = 8, AD_DIFFERENTIAL_MODE = 8, AD_TRIGGER_CTR0_EXT = 0x10,
AD_TRIGGER_FALLING_EDGE = 0x08, AD_TRIGGER_SCAN = 0x04, AD_TRIGGER_EXTERNAL = 0x02, A-D_TRIGGER_TIMER = 0x01,
AD_TRIGGER_VALID_MASK }

- enum ADGainCode {
ADGainCode_begin = (0 -1), AD_GAIN_CODE_0_10V, AD_GAIN_CODE_10V, AD_GAIN_CODE_0_5V,
AD_GAIN_CODE_5V, AD_GAIN_CODE_0_2V, AD_GAIN_CODE_2V, AD_GAIN_CODE_0_1V,
AD_GAIN_CODE_1V, ADGainCode_end }
- enum VENDOR_REQUEST {
VENDOR_REQUEST_begin = (0 -1), AUR_DIO_WRITE = 0x10, AUR_DIO_READ = 0x11, AUR_DIO_CONFIG = 0x12,
AUR_DIO_CONFIG_QUERY = 0x13, AUR_CTR_READ = 0x20, AUR_CTR_MODE = 0x21, AUR_CTR_LOAD = 0x22,
AUR_CTR_MODELOAD = 0x23, AUR_CTR_SELGATE = 0x24, AUR_CTR_READALL = 0x25, AUR_CTR_RE-ADLATCHED = 0x26,
AUR_CTR_COS_BULK_GATE2 = 0x27, AUR_CTR_PUR_FIRST = 0x28, AUR_CTR_PUR_OFRQ = 0x28, AU-R_CTR_COS_BULK_ABORT = 0x29,
AUR_CTR_PUR_MFRQ = 0x2C, AUR_CTR_PUR_EVCT = 0x2D, AUR_CTR_PUR_MPUL = 0x2E, AUR_WDG-_STATUS = 0x2E,
AUR_DIO_WDG16_DEPREC = 0x2F, AUR_READBACK_GLOBAL_STATE = 0x30, AUR_SAVE_GLOBAL_ST-ATE = 0x31, AUR_GEN_CLEAR_FIFO_NEXT = 0x34,
AUR_GEN_CLEAR_FIFO = 0x35, AUR_GEN_CLEAR_FIFO_WAIT = 0x36, AUR_GEN_ABORT_AND_CLEAR = 0x38, AUR_WDG = 0x44,
AUR_OFFLINE_READWRITE = 0x50, AUR_SELF_TEST_1 = 0x91, AUR_EEPROM_READ = 0xA2, AUR_EE-PROM_WRITE = 0XA2,
AUR_DAC_CONTROL = 0xB0, AUR_DAC_DATAPTR = 0xB1, AUR_DAC_DIVISOR = 0xB2, AUR_DAC_IMM-EDIATE = 0xB3,
AUR_GEN_STREAM_STATUS = 0xB4, AUR_FLASH_READWRITE = 0xB5, AUR_DAC_RANGE = 0xB7, AUR-_PROBE_CALFEATURE = 0xBA,
AUR_LOAD_BULK_CALIBRATION_BLOCK = 0xBB, AUR_DIO_STREAM_OPEN_OUTPUT = 0xBB, AUR_ST-ART_ACQUIRING_BLOCK = 0xBC, AUR_DIO_STREAM_OPEN_INPUT = 0xBC,
AUR_DIO_SETCLOCKS = 0xBD, AUR_ADC_SET_CONFIG = 0xBE, AUR_ADC_IMMEDIATE = 0xBF, AUR_DI-O_SPI_WRITE = 0xC0,
AUR_DIO_SPI_READ = 0xC1, AUR_ADC_GET_CONFIG = 0xD2, CYPRESS_GET_DESC = 0x06, VENDOR_-REQUEST_end }
- enum {
BITS_PER_BYTE = 8, AI_16_MAX_COUNTS = 65535, MAX_IMM_ADCS = 2, CAL_TABLE_WORDS = (64 ∗ 1024),
COUNTERS_PER_BLOCK = 3, COUNTER_NUM_MODES = 6, DAC_RESET = 0x80, CYPRESS_DESC_PAR-AMS = 0x0302,
CYPRESS_MAX_DESC_SIZE = 256, AIOUSB_MAX_NAME_SIZE = 100, EEPROM_SERIAL_NUMBER_ADD-RESS = 0x1DF8, EEPROM_CUSTOM_BASE_ADDRESS = 0x1E00,
EEPROM_CUSTOM_MIN_ADDRESS = 0, EEPROM_CUSTOM_MAX_ADDRESS = 0x1FF, AD_CONFIG_REG-ISTERS = 20, AD_MUX_CONFIG_REGISTERS = 21,
USB_WRITE_TO_DEVICE = 0x40, USB_READ_FROM_DEVICE = 0xC0, USB_BULK_WRITE_ENDPOINT = 2,
USB_BULK_READ_ENDPOINT = 6 }
- enum ADCalMode {
ADCalMode_begin = -1, AD_CAL_MODE_NORMAL = 0, AD_CAL_MODE_GROUND = 1, AD_CAL_MODE_R-EFERENCE = 3,
AD_CAL_MODE_BIP_GROUND = 5, AD_CAL_MODE_HIGH_REF = 7, ADCalMode_end = 8 }
- enum AIOCommandCode {
AIOCommandCode_begin = (0 -1), GENERIC_DOSOMETHING_PLACEHOLDER, AIO_CONTINUE_RUNNIN-G, AIO_TERMINATE_CALLBACK,
AIOCommandCode_end }

Enums that govern how commands are performed and operated.

24.107.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.107.2 Macro Definition Documentation

#define HAS_PTHREAD 1

#define EXPORTED_FUNCTION

#define CREATE_ENUM(*name,* *...* **) typedef enum { name ## _begin, __VA_ARGS__, name ## _end } name;**

#define CREATE_ENUM_W_START(*name,* *num,* *...* **) typedef enum { name ## _begin = (num-1), __VA_ARGS__, name ## _end } name;**

#define LAST_ENUM(*name* **) (name ## _end-1)**

#define FIRST_ENUM(*name* **) (name ## _begin+1)**

#define MIN_VALUE(*name* **) (name ## _begin+1)**

#define MAX_VALUE(*name* **) (name ## _end-1)**

#define VALID_ENUM(*name,* *value* **) (value >= FIRST_ENUM(name) && value <= LAST_ENUM(name))**

#define ERR_UNLESS_VALID_ENUM(*name,* *value* **) assert((value >= FIRST_ENUM(name) && value <= LAST_ENUM(name)))**

#define VALID_PRODUCT(*product* **) (VALID_ENUM(ProductIDS, product))**

#define GCC_VERSION

Value:

```
(__GNUC__ * 10000 \
                     + __GNUC_MINOR__ * 100     \
                     + __GNUC_PATCHLEVEL__)
```

#define ACCES_DEPRECATED(*FOO* **) __attribute__ ((deprecated))**

#define LAMBDA(*return_type,* *header,* *function_body* **)**

Value:

```
({                                                                  \
        return_type __fn ## __FILE__ ## __LINE__ header function_body   \
            __fn ## __FILE__ ## __LINE__ ;                              \
    })
```

#define MIN(*a,* *b* **) (((a)<(b))?(a):(b))**

#define MAX(*a,* *b* **) (((a)>(b))?(a):(b))**

#define AUR_CBUF_SETUP 0x01000007

#define AUR_CBUF_EXIT 0x00020002

#define NUMBER_CHANNELS 16

Simple macro for iterating over objects.

#define foreach_array(*i,* *ary,* *size*)

Value:

```
i = ary[0]; \
                                                    for ( int j = 0; j < size ; j ++, i = ary[i] )
```

#define AIO_MAKE_ERROR(*N*) -1∗abs(N)

#define AIOUSB_ERROR_VALUE(*N*) -1∗abs(N)

#define AIO_ASSERT(*...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { errno = -**AIOUSB_ERROR_INVALID_PARAMETER**; return -**AIOUSB_ERROR_INVALID_PARAMETER**; }

#define AIO_ASSERT_RET(*ret,* *...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { return ret; }

#define AIO_ASSERT_AIORET_TYPE(*ret,* *...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { errno = -abs(ret); return -abs(ret); }

#define AIO_ASSERT_NO_RETURN(*...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { return; }

#define AIO_ASSERT_EXIT(*...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { exit(-**AIOUSB_ERROR_INVALID_PARAMET-ER**); }

#define AIO_ASSERT_ERR_NO_RETURN(*err,* *...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { exit(-err); }

#define AIO_ASSERT_VALID_DATA(*err,* *...*) assert(__VA_ARGS__) ; if (!(__VA_ARGS__)) { return err; }

#define AIO_ASSERT_USB(*...*) **AIO_ASSERT_VALID_DATA**(-**AIOUSB_ERROR_INVALID_USBDEVICE**, __VA_ARGS__)

#define AIO_ASSERT_DIOBUF(*...*) **AIO_ASSERT_VALID_DATA**(-**AIOUSB_ERROR_INVALID_DIOBUF**, __VA_ARGS__)

#define AIO_ASSERT_CHANNELMASK(*...*) **AIO_ASSERT_VALID_DATA**(-**AIOUSB_ERROR_INVALID_AIOCHANNEL-MASK**, __VA_ARGS__)

#define AIO_ASSERT_CONFIG(*...*) **AIO_ASSERT_VALID_DATA**(-**AIOUSB_ERROR_INVALID_ADCCONFIG** , __VA_ARGS__)

#define AIO_ASSERT_AIOCONTBUF(*...*) **AIO_ASSERT_VALID_DATA**(-**AIOUSB_ERROR_INVALID_AIOCONTINUOU-S_BUFFER**, __VA_ARGS__);

#define AIO_ASSERT_AIOEITHER(*err,* *msg,* *...*) assert(__VA_ARGS__); if (!(__VA_ARGS__)) { **AIOEither** tmp; tmp.left = err; tmp.errmsg=strdup(msg); return tmp; }

#define AIO_ERROR(*X*) (-abs(X))

AIO_ERROR ∗ are just like the regular ASSERTIONS meaning that they argument should evaluate to true otherwise it will fail.

#define AIO_ERROR_VALID_DATA(*err,* *...*) if (!(__VA_ARGS__)) { return err; }

#define AIO_ERROR_VALID_DATA_RETVAL(*err,* *...*) if (!(__VA_ARGS__)) { return -abs(err); }

#define AIO_ERROR_VALID_AIORET_TYPE(*err,* *...*) if (!(__VA_ARGS__)) { return -abs(err); }

#define AIO_ERROR_AIOEITHER_VALID_DATA(*err,* *...*)

Value:

```
if ( !(__VA_ARGS__) ) { \
        AIOEither tmp; tmp.left = err; tmp.errmsg=NULL; return tmp; }
```

#define AIO_ERROR_VALID_DATA_W_CODE(*err,* *code,* *...*) if (!(__VA_ARGS__)) { { code; }; return err; }

#define AIO_ERROR_VALID_DATA_WITH_CODE(*retval,* *err,* *code*) if (! (code)) { errno = -abs(err); return retval; }

#define G_STMT_START do

#define G_STMT_END while (0)

#define G_BREAKPOINT( ) G_STMT_START{ raise (SIGTRAP); }G_STMT_END

#define EXIT_FN_IF_NO_VALID_USB(*d, r, f, u, g*)

Value:

```
do {            \
        if ( !d ) {                                            \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;                \
            goto g;                                            \
        } else if ( ( r = f ) != AIOUSB_SUCCESS  ) {           \
            goto g;                                            \
        } else if ( !(u = AIOUSBDeviceGetUSBHandle( d )))  {   \
            r = -AIOUSB_ERROR_INVALID_USBDEVICE;               \
            goto g;                                            \
        }                                                      \
    } while (0 )
```

#define AIOUSB_ERROR_OFFSET 100

#define LIBUSB_RESULT_TO_AIOUSB_RESULT(*code*) (unsigned long)(AIOUSB_ERROR_OFFSET + -(int)(code))

#define AIOUSB_RESULT_TO_LIBUSB_RESULT(*code*) (-((int)(code) - AIOUSB_ERROR_OFFSET))

#define ROOTCLOCK 10000000

24.107.3 Typedef Documentation

typedef int64_t AIORET_TYPE

typedef unsigned long AIORESULT

typedef unsigned short∗ COUNTS

typedef struct ushort_array Ushort_Array

typedef uint16_t AIOBufferType

typedef enum AIOUSB_BOOL_VAL AIOUSB_BOOL

typedef long double AIO_NUMBER

typedef struct lookup EnumStringLookup

24.107.4 Enumeration Type Documentation

enum AIO_SCAN_TYPE

Enumerator

AIO_PER_OVERSAMPLE
AIO_PER_CHANNEL
AIO_PER_SCANS

enum THREAD_STATUS

Enumerator

THREAD_STATUS_begin
INVALID_OBJECT
NOT_STARTED
RUNNING
WITH_DATA
TERMINATED
RUNNING_OR_WITH_DATA
JOINED
TERMINATED_OVERRUN
TERMINATING
THREAD_STATUS_end

enum **AIOContinuousBufMode**

Enumerator

AIOContinuousBufMode_begin

AIOCONTINUOUS_BUF_ALLORNONE

AIOCONTINUOUS_BUF_NORMAL

AIOCONTINUOUS_BUF_OVERRIDE

AIOContinuousBufMode_end

anonymous enum

Enumerator

MAX_USB_DEVICES

enum **AIOUSB_BOOL_VAL**

other libraries often declare BOOL, TRUE and FALSE, and worse, they declare these using #define; so we sidestep that potential conflict by declaring the same types prefixed with AIOUSB_; it's ugly, but if people want to use the shorter names and they are certain they won't conflict with anything else, they can define the ENABLE_BOOL_TYPE macro

Enumerator

AIOUSB_FALSE

AIOUSB_TRUE

enum **ProductIDS**

Enumerator

ProductIDS_begin

ACCES_VENDOR_ID

USB_DA12_8A_REV_A • these product IDs are constant ∗

USB_DA12_8A

USB_DA12_8E

USB_DIO_32 • these are the product IDs after firmware is uploaded to the device; prior ∗ to uploading the firmware, the product ID is that shown below minus 0x8000

USB_DIO_32I

USB_DIO_48

USB_DIO_96

USB_DIO24_CTR6

USB_DI16A_REV_A1

USB_DO16A_REV_A1

USB_DI16A_REV_A2

USB_DIO_16H

USB_DI16A

USB_DO16A

USB_DIO_16A

USB_IIRO_16

USB_II_16

USB_RO_16

USB_IIRO_8

USB_II_8

USB_IIRO_4

USB_IDIO_16

USB_II_16_OLD

USB_IDO_16

USB_IDIO_8

USB_II_8_OLD

USB_IDIO_4

USB_CTR_15

USB_IIRO4_2SM

USB_IIRO4_COM

USB_DIO16RO8

USB_DIO48DO24

USB_DIO24DO12

USB_DO24

PICO_DIO16RO8

USBP_II8IDO4A

USB_AI16_16A

USB_AI16_16E

USB_AI12_16A

USB_AI12_16

USB_AI12_16E

USB_AI16_64MA

USB_AI16_64ME

USB_AI12_64MA

USB_AI12_64M

USB_AI12_64ME

USB_AI16_32A

USB_AI16_32E

USB_AI12_32A

USB_AI12_32

USB_AI12_32E

USB_AI16_64A

USB_AI16_64E

USB_AI12_64A

USB_AI12_64

USB_AI12_64E

USB_AI16_96A

USB_AI16_96E

USB_AI12_96A

USB_AI12_96

USB_AI12_96E

USB_AI16_128A

USB_AI16_128E

USB_AI12_128A

USB_AI12_128

USB_AI12_128E

USB_AO_ARB1

USB_AO16_16A

USB_AO16_16

USB_AO16_12A

USB_AO16_12

USB_AO16_8A

USB_AO16_8

USB_AO16_4A

USB_AO16_4

USB_AO12_16A

USB_AO12_16

USB_AO12_12A

USB_AO12_12

USB_AO12_8A

USB_AO12_8

USB_AO12_4A

USB_AO12_4

USB_AIO16_16A

USB_AIO16_16E

USB_AIO12_16A

USB_AIO12_16

USB_AIO12_16E

USB_AIO16_64MA

USB_AIO16_64ME

USB_AIO12_64MA

USB_AIO12_64M

USB_AIO12_64ME

USB_AIO16_32A

USB_AIO16_32E

USB_AIO12_32A

USB_AIO12_32

USB_AIO12_32E

USB_AIO16_64A

USB_AIO16_64E

USB_AIO12_64A

USB_AIO12_64

USB_AIO12_64E

USB_AIO16_96A

USB_AIO16_96E

USB_AIO12_96A

USB_AIO12_96

USB_AIO12_96E

USB_AIO16_128A

USB_AIO16_128E

USB_AIO12_128A

USB_AIO12_128

USB_AIO12_128E

ProductIDS_end

anonymous enum

Enumerator

diFirst

diOnly

diNone

enum DACRange

range codes passed to DACSetBoardRange()

Enumerator

DACRange_begin

DAC_RANGE_0_5V

DAC_RANGE_5V

DAC_RANGE_0_10V

DAC_RANGE_10V

DACRange_end

enum FIFO_Method

FIFO clearing methods passed to AIOUSB_ClearFIFO()

Enumerator

FIFO_Method_begin

CLEAR_FIFO_METHOD_IMMEDIATE

CLEAR_FIFO_METHOD_AUTO

CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT

CLEAR_FIFO_METHOD_NOW

CLEAR_FIFO_METHOD_WAIT

FIFO_Method_end

enum ResultCode

The AIOUSB function result codes are a bit confusing; the result codes used in the Windows implementation of the API are defined in a system file, winerror.h; these result codes are generic and can apply to many applications; the very first result code, ERROR_SUCCESS, sounds like an oxymoron; the result codes used in libusb, on the other hand, are a lot more appealing; the result code for success is LIBUSB_SUCCESS; the result codes for errors are LIBUSB_ERROR_-xxx; further complicating matters is that the AIOUSB result codes must be non-negative since all the functions return an unsigned result, whereas the LIBUSB result codes are negative in the case of errors; both schemes use zero to denote success; it would also be nice to return the original libusb result code in cases where a libusb error causes an AIOUSB API function to fail; so to satisfy all these requirements, we've employed the following scheme:

- AIOUSB result codes in Linux start with "AIOUSB_"; the result code for success is AIOUSB_SUCCESS, which has a value of zero; the result codes for errors are AIOUSB_ERROR_xxx, which have positive values, starting with one (1)
- in order to offer users the option of using result codes whose names are similar to those cited in the AIOUSB API specification, we define a second set of result codes with names similar to those in API specification but which map to the same values as the AIOUSB_xxx result codes; these alternate result code names can be enabled by defining the macro ENABLE_WINDOWS_RESULT_CODES, which is not enabled by default
- in order to preserve the original libusb result codes and pass them back from an AIOUSB API function, we translate the libusb result codes to a format that conforms to the one AIOUSB employs and provide macros for converting the AIOUSB result code back to a libusb result code; LIBUSB_RESULT_TO_AIOUSB_RESULT() converts a libusb result code to an AIOUSB result code; AIOUSB_RESULT_TO_LIBUSB_RESULT() does the reverse; these macros cannot be used with LIBUSB_SUCCESS
- we provide an extended AIOUSB API function named AIOUSB_GetResultCodeAsString() that returns a string representation of the result code, including those derived from a libusb result code

Enumerator

ResultCode_begin

AIOUSB_SUCCESS

AIOUSB_ERROR_DEVICE_NOT_CONNECTED

AIOUSB_ERROR_DUP_NAME

AIOUSB_ERROR_NOT_INIT

AIOUSB_ERROR_FILE_NOT_FOUND

AIOUSB_ERROR_INVALID_DATA

AIOUSB_ERROR_INVALID_INDEX

AIOUSB_ERROR_INVALID_MUTEX

AIOUSB_ERROR_INVALID_PARAMETER

AIOUSB_ERROR_INVALID_THREAD

AIOUSB_ERROR_NOT_ENOUGH_MEMORY

AIOUSB_ERROR_INVALID_MEMORY

AIOUSB_ERROR_NOT_SUPPORTED

AIOUSB_ERROR_OPEN_FAILED

AIOUSB_ERROR_BAD_TOKEN_TYPE

AIOUSB_ERROR_TIMEOUT

AIOUSB_ERROR_DIVIDE_BY_ZERO

AIOUSB_ERROR_HANDLE_EOF

AIOUSB_ERROR_DEVICE_NOT_FOUND

AIOUSB_ERROR_USBDEVICE_NOT_FOUND

AIOUSB_ERROR_USB_INIT

AIOUSB_ERROR_INVALID_TIMEOUT

AIOUSB_ERROR_INVALID_AIOEITHER_ALLOCATION

AIOUSB_ERROR_INVALID_USBDEVICE

AIOUSB_ERROR_INVALID_VOLTAGES

AIOUSB_ERROR_INVALID_AIOCMD

AIOUSB_ERROR_INVALID_CALLBACK

AIOUSB_ERROR_INVALID_COUNTS

AIOUSB_ERROR_INVALID_COUNTS_CONVERTER

AIOUSB_ERROR_INVALID_DEVICE

AIOUSB_ERROR_INVALID_DEVICE_SETTING

AIOUSB_ERROR_INVALID_DEVICE_FUNCTIONAL_PARAMETER

AIOUSB_ERROR_INVALID_DEVICE_STREAM_SETTING

AIOUSB_ERROR_INVALID_DEVICE_CHANNEL_SETTING

AIOUSB_ERROR_INVALID_DEVICE_MUX_CHANNEL_SETTING

AIOUSB_ERROR_INVALID_CHANNELS_PER_GROUP_SETTING

AIOUSB_ERROR_INVALID_AIOCHANNELMASK

AIOUSB_ERROR_INVALID_CONFIG

AIOUSB_ERROR_INVALID_DIOBUF

AIOUSB_ERROR_INVALID_GAINCODE

AIOUSB_ERROR_INVALID_CALMODE

AIOUSB_ERROR_INVALID_CHANNEL_NUMBER

AIOUSB_ERROR_INVALID_AIOCONFIGURATION

AIOUSB_ERROR_INVALID_AIOARGUMENT

AIOUSB_ERROR_INVALID_AIODEVICE_QUERY

AIOUSB_ERROR_INVALID_AIOEITHER

AIOUSB_ERROR_INVALID_AIOFIFO

AIOUSB_ERROR_INVALID_ADCCONFIG

AIOUSB_ERROR_INVALID_ADCCONFIG_SIZE

AIOUSB_ERROR_INVALID_ADCCONFIG_SETTING

AIOUSB_ERROR_INVALID_ADCCONFIG_TRIGGER_SETTING

AIOUSB_ERROR_INVALID_ADCCONFIG_CAL_SETTING

AIOUSB_ERROR_INVALID_ADCCONFIG_CHANNEL_SETTING

AIOUSB_ERROR_INVALID_ADCCONFIG_OVERSAMPLE_SETTING

AIOUSB_ERROR_INVALID_ADCCONFIG_REGISTER_SETTING

AIOUSB_ERROR_INVALID_ADCCONFIG_MUX_SETTING

AIOUSB_ERROR_INVALID_ADCCONFIG_DEVICE

AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER

AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER_NUM_CHANNELS

AIOUSB_ERROR_INVALID_AIOBUFTYPE

AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_CHANNEL_RANGE

AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_NUM_CHANNELS

AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_INDEX_NUM

AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_START_END_CHANNEL

AIOUSB_ERROR_AIOCOMMANDLINE_HELP

AIOUSB_ERROR_INVALID_LIBUSB_DEVICE_HANDLE

AIOUSB_FIFO_COPY_ERROR

AIOUSB_ERROR_LIBUSB

ResultCode_end

anonymous enum

Enumerator

AD_MAX_CHANNELS

AD_GAIN_CODE_MASK

enum ADRegister

Enumerator

ADRegister_begin

AD_REGISTER_CAL_MODE

AD_REGISTER_TRIG_COUNT

AD_REGISTER_START_END

AD_REGISTER_OVERSAMPLE

AD_REGISTER_MUX_START_END

ADRegister_end

anonymous enum

Enumerator

AD_MAX_CONFIG_REGISTERS

AD_MIN_CONFIG_REGISTERS

AD_MAX_TIMEOUT

AD_MIN_TIMEOUT

AD_NUM_GAIN_CODE_REGISTERS

AD_CONFIG_GAIN_CODE

AD_REGISTER_GAIN_CODE

AD_CONFIG_CAL_MODE

AD_CONFIG_TRIG_COUNT

AD_CONFIG_START_END

AD_CONFIG_OVERSAMPLE

AD_CONFIG_MUX_START_END

AD_CONFIG_START_STOP_CHANNEL_EX

AD_NUM_GAIN_CODES

AD_DIFFERENTIAL_MODE

AD_TRIGGER_CTR0_EXT

AD_TRIGGER_FALLING_EDGE

AD_TRIGGER_SCAN

AD_TRIGGER_EXTERNAL

AD_TRIGGER_TIMER

AD_TRIGGER_VALID_MASK

enum ADGainCode

Enumerator

ADGainCode_begin

AD_GAIN_CODE_0_10V

AD_GAIN_CODE_10V

AD_GAIN_CODE_0_5V

AD_GAIN_CODE_5V

AD_GAIN_CODE_0_2V

AD_GAIN_CODE_2V

AD_GAIN_CODE_0_1V

AD_GAIN_CODE_1V

ADGainCode_end

enum VENDOR_REQUEST

Enumerator

VENDOR_REQUEST_begin

AUR_DIO_WRITE

AUR_DIO_READ

AUR_DIO_CONFIG

AUR_DIO_CONFIG_QUERY

AUR_CTR_READ

AUR_CTR_MODE

AUR_CTR_LOAD

AUR_CTR_MODELOAD

AUR_CTR_SELGATE

AUR_CTR_READALL

AUR_CTR_READLATCHED

AUR_CTR_COS_BULK_GATE2

AUR_CTR_PUR_FIRST

AUR_CTR_PUR_OFRQ

AUR_CTR_COS_BULK_ABORT

AUR_CTR_PUR_MFRQ

AUR_CTR_PUR_EVCT

AUR_CTR_PUR_MPUL

AUR_WDG_STATUS

AUR_DIO_WDG16_DEPREC

AUR_READBACK_GLOBAL_STATE

AUR_SAVE_GLOBAL_STATE

AUR_GEN_CLEAR_FIFO_NEXT

AUR_GEN_CLEAR_FIFO

AUR_GEN_CLEAR_FIFO_WAIT

AUR_GEN_ABORT_AND_CLEAR

AUR_WDG

AUR_OFFLINE_READWRITE

AUR_SELF_TEST_1

AUR_EEPROM_READ

AUR_EEPROM_WRITE

AUR_DAC_CONTROL

AUR_DAC_DATAPTR

AUR_DAC_DIVISOR

AUR_DAC_IMMEDIATE

AUR_GEN_STREAM_STATUS

AUR_FLASH_READWRITE

AUR_DAC_RANGE

AUR_PROBE_CALFEATURE

AUR_LOAD_BULK_CALIBRATION_BLOCK

AUR_DIO_STREAM_OPEN_OUTPUT

AUR_START_ACQUIRING_BLOCK

AUR_DIO_STREAM_OPEN_INPUT

AUR_DIO_SETCLOCKS

AUR_ADC_SET_CONFIG

AUR_ADC_IMMEDIATE

AUR_DIO_SPI_WRITE

AUR_DIO_SPI_READ

AUR_ADC_GET_CONFIG

CYPRESS_GET_DESC

VENDOR_REQUEST_end

anonymous enum

Enumerator

- ***BITS_PER_BYTE***
- ***AI_16_MAX_COUNTS***
- ***MAX_IMM_ADCS***
- ***CAL_TABLE_WORDS***
- ***COUNTERS_PER_BLOCK***
- ***COUNTER_NUM_MODES***
- ***DAC_RESET***
- ***CYPRESS_DESC_PARAMS***
- ***CYPRESS_MAX_DESC_SIZE***
- ***AIOUSB_MAX_NAME_SIZE***
- ***EEPROM_SERIAL_NUMBER_ADDRESS***
- ***EEPROM_CUSTOM_BASE_ADDRESS***
- ***EEPROM_CUSTOM_MIN_ADDRESS***
- ***EEPROM_CUSTOM_MAX_ADDRESS***
- ***AD_CONFIG_REGISTERS***
- ***AD_MUX_CONFIG_REGISTERS***
- ***USB_WRITE_TO_DEVICE***
- ***USB_READ_FROM_DEVICE***
- ***USB_BULK_WRITE_ENDPOINT***
- ***USB_BULK_READ_ENDPOINT***

enum ADCalMode

Enumerator

- ***ADCalMode_begin***
- ***AD_CAL_MODE_NORMAL***
- ***AD_CAL_MODE_GROUND***
- ***AD_CAL_MODE_REFERENCE***
- ***AD_CAL_MODE_BIP_GROUND***
- ***AD_CAL_MODE_HIGH_REF***
- ***ADCalMode_end***

enum AIOCommandCode

Enums that govern how commands are performed and operated.

Enumerator

- ***AIOCommandCode_begin***
- ***GENERIC_DOSOMETHING_PLACEHOLDER***
- ***AIO_CONTINUE_RUNNING***
- ***AIO_TERMINATE_CALLBACK***
- ***AIOCommandCode_end***

24.108 lib/aiousb.h File Reference

General header files for the AIOUSB library.

```
#include <stdlib.h>
#include <assert.h>
#include "AIOTypes.h"
#include "AIODeviceInfo.h"
#include "AIODeviceQuery.h"
#include "AIODeviceTable.h"
#include "AIOUSBDevice.h"
#include "ADCConfigBlock.h"
#include "AIOUSB_Properties.h"
#include "AIOUSB_DIO.h"
#include "AIOUSB_ADC.h"
#include "AIOUSB_CTR.h"
#include "AIOUSB_DAC.h"
#include "AIOUSB_CustomEEPROM.h"
#include "USBDevice.h"
#include "AIOUSB_Log.h"
#include "AIOCommandLine.h"
```

24.108.1 Detailed Description

General header files for the AIOUSB library.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.109 lib/AIOUSB_ADC.c File Reference

Configuration functions for ADC elements.

```
#include "AIOUSB_ADC.h"
#include "AIOUSB_CTR.h"
#include "AIOTypes.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Core.h"
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>
```

Macros

- #define DEVICE_SAMPLE_BUFFER_SIZE 1024
- #define STREAMING_PNA_DEFINITIONS

Functions

- AIORET_TYPE adc_get_bulk_data (ADCConfigBlock ∗config, USBDevice ∗usb, unsigned char endpoint, unsigned char ∗data, int datasize, int ∗bytes, unsigned timeout)
- unsigned long ADC_ResetDevice (unsigned long DeviceIndex)
- unsigned long ∗ ADC_GetConfigSize (ADConfigBlock ∗config)
- unsigned char ∗ ADC_GetConfigRegisters (ADConfigBlock ∗config)
- AIORET_TYPE ADC_ReadADConfigBlock (unsigned long DeviceIndex, ADConfigBlock ∗config)
- unsigned long ReadConfigBlock (unsigned long DeviceIndex, AIOUSB_BOOL forceRead)
- unsigned long WriteConfigBlock (unsigned long DeviceIndex)
- AIORESULT ADC_Acquire_Reference_Counts (unsigned long DeviceIndex, double ∗groundCounts, double ∗referenceCounts)

  Performs a number of ADC_GetImmediate calls and then averages out the values to determine adequate values for the Ground and Reference values.
- PRIVATE AIORET_TYPE AIOUSB_GetScan (unsigned long DeviceIndex, unsigned short counts[ ])

  Performs a scan and averages the voltage values.
- PRIVATE unsigned long AIOUSB_ArrayCountsToVolts (unsigned long DeviceIndex, int startChannel, int numChannels, const unsigned short counts[ ], double volts[ ])
- PRIVATE AIORET_TYPE AIOUSB_ArrayVoltsToCounts (unsigned long DeviceIndex, int startChannel, int numChannels, const double volts[ ], unsigned short counts[ ])
- unsigned short AIOUSB_VoltsToCounts (unsigned long DeviceIndex, unsigned channel, double volts)
- AIORET_TYPE ADC_GetChannelV (unsigned long DeviceIndex, unsigned long ChannelIndex, double ∗singlevoltage)

  Read one voltage input's current value.
- AIORET_TYPE ADC_GetScanV (unsigned long DeviceIndex, double ∗pBuf)

  Preferred way to get immediate scan readings.
- AIORET_TYPE ADC_GetScan (unsigned long DeviceIndex, unsigned short ∗pBuf)

  This simple function takes one scan of A/D data, in counts.
- unsigned long ADC_GetConfig (unsigned long DeviceIndex, unsigned char ∗ConfigBuf, unsigned long ∗ConfigBufSize)

  Determ ine inform ation about the device found at a specific DeviceIndex.
- int adcblock_valid_trigger_settings (ADConfigBlock ∗config)
- int adcblock_valid_channel_settings (ADConfigBlock ∗config, int ADCMUXChannels)
- unsigned long valid_config_block (ADConfigBlock ∗config)
- int adcblock_valid_size (ADConfigBlock ∗config)
- unsigned long ADC_SetConfig (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize)
- unsigned long ADC_CopyConfig (unsigned long DeviceIndex, ADConfigBlock ∗config)

  Copies the given ADConfig object into the cachedConfigBlock that is used to communicate with the USB device.
- unsigned long ADC_RangeAll (unsigned long DeviceIndex, unsigned char ∗pGainCodes, unsigned long bSingleEnded)
- unsigned long ADC_Range1 (unsigned long DeviceIndex, unsigned long ADChannel, unsigned char GainCode, unsigned long bSingleEnded)
- unsigned long ADC_ADMode (unsigned long DeviceIndex, unsigned char TriggerMode, unsigned char CalMode)
- AIORESULT ADC_SetOversample (unsigned long DeviceIndex, unsigned char Oversample)
- unsigned ADC_GetOversample (unsigned long DeviceIndex)
- AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long DeviceIndex, unsigned gain, AIOUSB_BOOL differentialMode)
- AIORESULT ADC_GetMaxClockRate (unsigned long ProductID, unsigned int num_channels, unsigned int num_oversamples)

  Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.
- AIORESULT ADC_ClockRateForADCProduct (unsigned long ProductID)
- unsigned long ADC_SetScanLimits (unsigned long DeviceIndex, unsigned long StartChannel, unsigned long EndChannel)
- unsigned long ADC_SetCal (unsigned long DeviceIndex, const char ∗CalFileName)
- AIOUSB_BOOL ADC_CanCalibrate (unsigned long productID)
- unsigned long ADC_QueryCal (unsigned long DeviceIndex)
- unsigned long ADC_Initialize (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize, const char ∗CalFileName)

  Determ ine information about the device found at a specific DeviceIndex.
- unsigned long ADC_BulkAcquire (unsigned long DeviceIndex, unsigned long BufSize, void ∗pBuf)

  Determine inform ation about the device found at a specific DeviceIndex.
- AIOBuf ∗ CreateSmartBuffer (unsigned long DeviceIndex)

  After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.
- unsigned long ADC_BulkPoll (unsigned long DeviceIndex, unsigned long ∗BytesLeft)
- unsigned long ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)

Creates FastIT Config Blocks.
- unsigned char ∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗config)
- AIORESULT ADC_ClearFastITConfig (unsigned long DeviceIndex)

  Frees memory associated with the FastConfig Config blocks.
- unsigned long ADC_CreateADBuf (AIOUSBDevice ∗const deviceDesc, int size)
- void ADC_ClearADBuf (AIOUSBDevice ∗deviceDesc)
- unsigned long ADC_InitFastITScanV (unsigned long DeviceIndex)
- unsigned long ADC_ResetFastITScanV (unsigned long DeviceIndex)
- unsigned long ADC_SetFastITScanVChannels (unsigned long DeviceIndex, unsigned long NewChannels)
- void ADC_Debug_Register_Settings (ADConfigBlock ∗config)

  Just a debugging function for listing all attributes of a config object.
- char ∗ ADConfigBlockToYAML (ADConfigBlock ∗config)
- unsigned long ADC_GetFastITScanV (unsigned long DeviceIndex, double ∗pData)
- unsigned long ADC_GetITScanV (unsigned long DeviceIndex, double ∗pBuf)
- AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetDiscardFirstSample (unsigned long DeviceIndex, AIOUSB_BOOL discard)
- void AIOUSB_Copy_Config_Block (ADConfigBlock ∗to, ADConfigBlock ∗from)
- unsigned long AIOUSB_Validate_ADC_Device (unsigned long DeviceIndex)
- double GetHiRef (unsigned long deviceIndex)
- void DoLoadCalTable (unsigned short ∗const calTable, unsigned long DeviceIndex, double groundCounts, double referenceCounts)

  Loads the Cal table for Automatic internal calibration.
- AIORESULT AIOUSB_SetRangeSingle (ADConfigBlock ∗config, unsigned long channel, unsigned char gainCode)
- AIORET_TYPE ConfigureAndBulkAcquire (unsigned long DeviceIndex, ADConfigBlock ∗config)
- unsigned long AIOUSB_ADC_InternalCal (unsigned long DeviceIndex, AIOUSB_BOOL autoCal, unsigned short returnCalTable[ ], const char ∗saveFileName)

  Performs automatic calibration of the ADC.
- void AIOUSB_SetRegister (ADConfigBlock ∗cb, unsigned int Register, unsigned char value)
- unsigned char AIOUSB_GetRegister (ADConfigBlock ∗cb, unsigned int Register)
- AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock ∗config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock ∗config, unsigned channel)
- AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock ∗config, unsigned channel, unsigned gainCode)
- AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock ∗config, unsigned channel)
- AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock ∗config, unsigned channel, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock ∗config)
- AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock ∗config, unsigned calMode)
- unsigned AIOUSB_GetTriggerMode (const ADConfigBlock ∗config)
- AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock ∗config, unsigned triggerMode)
- unsigned AIOUSB_GetStartChannel (const ADConfigBlock ∗config)
- unsigned AIOUSB_GetEndChannel (const ADConfigBlock ∗config)
- AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock ∗config, unsigned startChannel, unsigned endChannel)
- AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock ∗config)
- AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock ∗config, unsigned overSample)
- unsigned long AIOUSB_ADC_ExternalCal (unsigned long DeviceIndex, const double points[ ], int numPoints, unsigned short returnCalTable[ ], const char ∗saveFileName)

Variables

- struct ADRange adRanges [AD_NUM_GAIN_CODES]
- int dRef = 3

24.109.1 Detailed Description

Configuration functions for ADC elements.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.109.2 Macro Definition Documentation

#define DEVICE_SAMPLE_BUFFER_SIZE 1024

#define STREAMING_PNA_DEFINITIONS

Value:

```
struct timespec foo , bar;                                      \
    unsigned deltas[16*8192];                                        \
    unsigned transactions[16*8192];                                  \
    int tindex = 0;                                                  \
    int num_reads = 0;
```

24.109.3 Function Documentation

AIORET_TYPE adc_get_bulk_data (ADCConfigBlock ∗ *config,* USBDevice ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *datasize,* int ∗ *bytes,* unsigned *timeout*)

unsigned long ADC_ResetDevice (unsigned long *DeviceIndex*)

unsigned long∗ ADC_GetConfigSize (ADConfigBlock ∗ *config*)

unsigned char∗ ADC_GetConfigRegisters (ADConfigBlock ∗ *config*)

AIORET_TYPE ADC_ReadADConfigBlock (unsigned long *DeviceIndex,* ADConfigBlock ∗ *config*)

unsigned long ReadConfigBlock (unsigned long *DeviceIndex,* AIOUSB_BOOL *forceRead*)

Parameters

DeviceIndex	
forceRead	

unsigned long WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

AIORESULT ADC_Acquire_Reference_Counts (unsigned long *DeviceIndex,* double ∗ *groundCounts,* double ∗ *referenceCounts*)

Performs a number of ADC_GetImmediate calls and then averages out the values to determine adequate values for the Ground and Reference values.

Parameters

DeviceIndex	
groundCounts	
referenceCounts	

PRIVATE AIORET_TYPE AIOUSB_GetScan (unsigned long *DeviceIndex,* unsigned short *counts[]*)

Performs a scan and averages the voltage values.

Parameters

DeviceIndex	
counts	

Returns

Note

In theory, all the A/D functions, including AIOUSB_GetScan(), should work in all measurement modes, including calibration mode; in practice, however, the device will return only a single sample in calibration mode; therefore, users must be careful to select a single channel and set oversample to zero during calibration mode; attempting to read more than one channel or use an oversample setting of more than zero in calibration mode will result in a timeout error; as a convenience to the user we automatically impose this restriction here in AIOUSB_GetScan(); if the device is changed to permit normal use of the A/D functions in calibration mode, we will have to modify this function to somehow recognize which devices support that capability, or simply delete this restriction altogether and rely on the users' good judgment

The oversample setting dictates how many samples to take *in addition* to the primary sample; if oversample is zero, we take just one sample for each channel; if oversample is greater than zero then we average the primary sample and all of its over-samples; if the discardFirstSample setting is enabled, then we discard the primary sample, leaving just the over-samples; thus, if discardFirstSample is enabled, we must take at least one over-sample in order to have any data left; there's another complication: the device buffer is limited to a small number of samples, so we have to limit the number of over-samples to what the device buffer can accommodate, so the actual oversample setting depends on the number of channels being scanned; we also preserve and restore the original oversample setting specified by the user; since the user is expecting to average (1 + oversample) samples, then if discardFirstSample is enabled we simply always add one

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

PRIVATE unsigned long AIOUSB_ArrayCountsToVolts (unsigned long *DeviceIndex,* int *startChannel,* int *numChannels,* const unsigned short *counts[ ],* double *volts[ ]*)

Parameters

DeviceIndex	
startChannel	
numChannels	
counts	
volts	

Returns

PRIVATE AIORET_TYPE AIOUSB_ArrayVoltsToCounts (unsigned long *DeviceIndex,* int *startChannel,* int *numChannels,* const double *volts[ ],* unsigned short *counts[ ]*)

Parameters

DeviceIndex	
startChannel	
numChannels	
volts	
counts	

Returns

unsigned short AIOUSB_VoltsToCounts (unsigned long *DeviceIndex,* unsigned *channel,* double *volts*)

Parameters

DeviceIndex	
channel	
volts	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex,* unsigned long *ChannelIndex,* double ∗ *singlevoltage*)

Read one voltage input's current value.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ChannelIndex	number indicating which channel's data you wish to get
singlevoltage	a pointer to a double precision IEEE floating point num ber which will receive the value read

Note

This is a slow function

Returns

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

AIORET_TYPE ADC_GetScanV (unsigned long *DeviceIndex,* double ∗ *pBuf*)

Preferred way to get immediate scan readings.

Will Scan all channels (ie vectored) perform averaging and culling of data.

Parameters

DeviceIndex	
pBuf	

Returns

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

AIORET_TYPE ADC_GetScan (unsigned long *DeviceIndex,* unsigned short ∗ *pBuf*)

This simple function takes one scan of A/D data, in counts.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
pBuf	Pointer to an array of W ORDs. Each elem ent in the array will receive the value read from the corresponding A/D input channel. The array m ust be at least as large as the num ber of A/D input channels your product contains (16, 32, 64, 96, or 128) - but it is safe to always pass a pointer to an array of 128 W ORDs. Only elem ents in the array corresponding to A/D channels actually acquired during the scan will be updated: start-channel through end-channel, inclusive. Other values will rem ain unchanged.

Returns

AIORET_TYPE either AIOUSB_SUCCESS or a failure

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

unsigned long ADC_GetConfig (unsigned long *DeviceIndex,* unsigned char ∗ *ConfigBuf,* unsigned long ∗ *ConfigBufSize*)

Determ ine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ConfigBuf	a pointer to the first of an array of bytes for configuration data
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to read. W ill be set to the num ber of configuration bytes read

Returns

int adcblock_valid_trigger_settings (ADConfigBlock ∗ *config*)

int adcblock_valid_channel_settings (ADConfigBlock ∗ *config,* int *ADCMUXChannels*)

unsigned long valid_config_block (ADConfigBlock ∗ *config*)

int adcblock_valid_size (ADConfigBlock ∗ *config*)

unsigned long ADC_SetConfig (unsigned long *DeviceIndex,* unsigned char ∗ *pConfigBuf,* unsigned long ∗ *ConfigBufSize*)

Parameters

DeviceIndex	
pConfigBuf	
ConfigBufSize	

Returns

validate settings

unsigned long ADC_CopyConfig (unsigned long *DeviceIndex,* ADConfigBlock ∗ *config*)

Copies the given ADConfig object into the cachedConfigBlock that is used to communicate with the USB device.

Parameters

DeviceIndex	
config	

Returns

validate settings

unsigned long ADC_RangeAll (unsigned long *DeviceIndex,* unsigned char ∗ *pGainCodes,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
pGainCodes	
bSingleEnded	

Returns

unsigned long ADC_Range1 (unsigned long *DeviceIndex*, unsigned long *ADChannel*, unsigned char *GainCode*, unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
ADChannel	
GainCode	
bSingleEnded	

Returns

unsigned long ADC_ADMode (unsigned long *DeviceIndex*, unsigned char *TriggerMode*, unsigned char *CalMode*)

Parameters

DeviceIndex	
TriggerMode	
CalMode	

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex*, unsigned char *Oversample*)

Parameters

DeviceIndex	
Oversample	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long *DeviceIndex*, unsigned *gain*, AIOUSB_BOOL *differentialMode*)

AIORESULT ADC_GetMaxClockRate (unsigned long *ProductID*, unsigned int *num_channels*, unsigned int *num_oversamples*)

Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.

Parameters

ProductID	produc we are looking up
num_channels	Number of channels we will be sampling on
num_-oversamples	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

unsigned long ADC_SetScanLimits (unsigned long *DeviceIndex,* unsigned long *StartChannel,* unsigned long *EndChannel*)

Parameters

DeviceIndex	
StartChannel	
EndChannel	

Returns

unsigned long ADC_SetCal (unsigned long *DeviceIndex,* const char ∗ *CalFileName*)

Parameters

DeviceIndex	
CalFileName	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *productID*)

unsigned long ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long ADC_Initialize (unsigned long *DeviceIndex,* unsigned char ∗ *pConfigBuf,* unsigned long ∗ *ConfigBufSize,* const char ∗ *CalFileName*)

Determ ine information about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
pConfigBuf	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to write.
CalFileName	the file nam e of a calibration file, or a com m and string. See ADC_SetCal() for details.

Returns

AIOUSB_SUCCESS if successful, error otherwise.

unsigned long ADC_BulkAcquire (unsigned long *DeviceIndex,* unsigned long *BufSize,* void ∗ *pBuf*)

Determine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
BufSize	the size, in bytes, of the buffer to receive the data
pBuf	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return im m ediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or m oved. Use ADC_BulkPoll() to query this background operation.

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

AIOBuf∗ CreateSmartBuffer (unsigned long *DeviceIndex*)

After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.

Parameters

DeviceIndex	

Returns

AIOBuf ∗ new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

unsigned long ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long ∗ *BytesLeft*)

Parameters

DeviceIndex	
BytesLeft	

Returns

unsigned long ADC_CreateFastITConfig (unsigned long *DeviceIndex,* int *size*)

Creates FastIT Config Blocks.

Parameters

DeviceIndex	
size	

Returns

unsigned char∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗ *config*)

AIORESULT ADC_ClearFastITConfig (unsigned long *DeviceIndex*)

Frees memory associated with the FastConfig Config blocks.

Use this call after you are done using the ADC_FastIT∗ Functions

Parameters

DeviceIndex	

unsigned long ADC_CreateADBuf (AIOUSBDevice ∗const *deviceDesc,* int *size*)

void ADC_ClearADBuf (AIOUSBDevice ∗ *deviceDesc*)

unsigned long ADC_InitFastITScanV (unsigned long *DeviceIndex*)

unsigned long ADC_ResetFastITScanV (unsigned long *DeviceIndex*)

unsigned long ADC_SetFastITScanVChannels (unsigned long *DeviceIndex,* unsigned long *NewChannels*)

void ADC_Debug_Register_Settings (ADConfigBlock ∗ *config*)

Just a debugging function for listing all attributes of a config object.

char∗ ADConfigBlockToYAML (ADConfigBlock ∗ *config*)

```
* ---
* config:
*   channels:
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   - gain: 0-10V
*   calibration: Normal
*   trigger:
*     edge: falling edge
*     scan: all channels
*     type: external
*
```

unsigned long ADC_GetFastITScanV (unsigned long *DeviceIndex,* double ∗ *pData*)

Parameters

DeviceIndex	
pData	buffer we will write data into

Returns

unsigned long ADC_GetITScanV (unsigned long *DeviceIndex,* double ∗ *pBuf*)

Parameters

DeviceIndex	
pBuf	

Returns

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex,* AIOUSB_BOOL *discard*)

Parameters

DeviceIndex	
discard	

Returns

void AIOUSB_Copy_Config_Block (ADConfigBlock ∗ *to,* ADConfigBlock ∗ *from*)

unsigned long AIOUSB_Validate_ADC_Device (unsigned long *DeviceIndex*)

double GetHiRef (unsigned long *deviceIndex*)

Parameters

deviceIndex	

Returns

void DoLoadCalTable (unsigned short ∗const *calTable,* unsigned long *DeviceIndex,* double *groundCounts,* double *referenceCounts*)

Loads the Cal table for Automatic internal calibration.

Parameters

calTable	
DeviceIndex	
groundCounts	
referenceCounts	

AIORESULT AIOUSB_SetRangeSingle (ADConfigBlock ∗ *config,* unsigned long *channel,* unsigned char *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE ConfigureAndBulkAcquire (unsigned long *DeviceIndex,* ADConfigBlock ∗ *config*)

unsigned long AIOUSB_ADC_InternalCal (unsigned long *DeviceIndex,* AIOUSB_BOOL *autoCal,* unsigned short *returnCalTable[ ],* const char ∗ *saveFileName*)

Performs automatic calibration of the ADC.

Parameters

DeviceIndex	
autoCal	
returnCalTable	
saveFileName	

Returns

void AIOUSB_SetRegister (**ADConfigBlock** ∗ *cb,* unsigned int *Register,* unsigned char *value*)

unsigned char AIOUSB_GetRegister (**ADConfigBlock** ∗ *cb,* unsigned int *Register*)

AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (**ADConfigBlock** ∗ *config,* unsigned *gainCode,* **AIOUSB_BOOL** *differentialMode*)

AIORET_TYPE AIOUSB_GetGainCode (const **ADConfigBlock** ∗ *config,* unsigned *channel*)

AIORET_TYPE AIOUSB_SetGainCode (**ADConfigBlock** ∗ *config,* unsigned *channel,* unsigned *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE AIOUSB_IsDifferentialMode (const **ADConfigBlock** ∗ *config,* unsigned *channel*)

Parameters

config	
channel	

Returns

AIORET_TYPE AIOUSB_SetDifferentialMode (**ADConfigBlock** ∗ *config,* unsigned *channel,* **AIOUSB_BOOL** *differentialMode*)

Parameters

config	
channel	
differentialMode	

AIORET_TYPE AIOUSB_GetCalMode (const **ADConfigBlock** ∗ *config*)

AIORET_TYPE AIOUSB_SetCalMode (**ADConfigBlock** ∗ *config,* unsigned *calMode*)

Parameters

config	
calMode	

unsigned AIOUSB_GetTriggerMode (const **ADConfigBlock** ∗ *config*)

Parameters

config	

Returns

AIORET_TYPE AIOUSB_SetTriggerMode (**ADConfigBlock** ∗ *config,* unsigned *triggerMode*)

unsigned AIOUSB_GetStartChannel (const **ADConfigBlock** ∗ *config*)

unsigned AIOUSB_GetEndChannel (const **ADConfigBlock** ∗ *config*)

AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock ∗ *config*, unsigned *startChannel*, unsigned *endChannel*)

Parameters

config	
startChannel	
endChannel	

AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock ∗ *config*)

AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock ∗ *config,* unsigned *overSample*)

unsigned long AIOUSB_ADC_ExternalCal (unsigned long *DeviceIndex,* const double *points[ ],* int *numPoints,* unsigned short *returnCalTable[ ],* const char ∗ *saveFileName*)

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                            ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

24.109.4 Variable Documentation

struct ADRange adRanges[AD_NUM_GAIN_CODES]

Initial value:

```
= {
  { 0   , 10 },
  { -10 , 20 },
  { 0   , 5  },
  { -5  , 10 },
  { 0   , 2  },
  { -2  , 4  },
  { 0   , 1  },
  { -1  , 2  }
}
```

int dRef = 3

24.110 lib/AIOUSB_ADC.h File Reference

```
#include "AIOTypes.h"
#include "AIOBuf.h"
#include "ADCConfigBlock.h"
#include "USBDevice.h"
```

Functions

- AIORET_TYPE ADC_GetScanV (unsigned long DeviceIndex, double ∗voltages)

  Preferred way to get immediate scan readings.
- AIORESULT ADC_RangeAll (unsigned long DeviceIndex, unsigned char ∗pGainCodes, unsigned long bSingleEnded)
- AIORET_TYPE ADC_GetChannelV (unsigned long DeviceIndex, unsigned long ChannelIndex, double ∗singlevoltage)

  Read one voltage input's current value.
- AIORET_TYPE ADC_GetScan (unsigned long DeviceIndex, unsigned short ∗pBuf)

  This simple function takes one scan of A/D data, in counts.
- AIORET_TYPE AIOUSB_GetScan (unsigned long DeviceIndex, unsigned short counts[ ])

  Performs a scan and averages the voltage values.
- AIORESULT ADC_GetConfig (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize)

  Determ ine inform ation about the device found at a specific DeviceIndex.
- AIORESULT ADC_SetConfig (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize)
- AIORESULT ADC_Range1 (unsigned long DeviceIndex, unsigned long ADChannel, unsigned char GainCode, unsigned long bSingleEnded)
- AIORESULT ADC_ADMode (unsigned long DeviceIndex, unsigned char TriggerMode, unsigned char CalMode)
- AIORESULT ADC_SetScanLimits (unsigned long DeviceIndex, unsigned long StartChannel, unsigned long EndChannel)
- AIORESULT ADC_GetMaxClockRate (unsigned long ProductID, unsigned int num_channels, unsigned int num_oversamples)

  Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.
- AIORESULT ADC_ClockRateForADCProduct (unsigned long ProductID)
- AIORESULT ADC_SetCal (unsigned long DeviceIndex, const char ∗CalFileName)
- AIORESULT ADC_QueryCal (unsigned long DeviceIndex)
- AIOUSB_BOOL ADC_CanCalibrate (unsigned long ProductID)
- AIORESULT ADC_Initialize (unsigned long DeviceIndex, unsigned char ∗pConfigBuf, unsigned long ∗ConfigBufSize, const char ∗CalFileName)

  Determ ine information about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkAcquire (unsigned long DeviceIndex, unsigned long BufSize, void ∗pBuf)

  Determine inform ation about the device found at a specific DeviceIndex.
- AIORESULT ADC_BulkPoll (unsigned long DeviceIndex, unsigned long ∗BytesLeft)
- AIORESULT ADC_InitFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)

  Creates FastIT Config Blocks.
- AIORESULT ADC_ResetFastITScanV (unsigned long DeviceIndex)
- AIORESULT ADC_SetFastITScanVChannels (unsigned long DeviceIndex, unsigned long NewChannels)
- AIORESULT ADC_GetFastITScanV (unsigned long DeviceIndex, double ∗pData)
- AIORESULT ADC_GetITScanV (unsigned long DeviceIndex, double ∗pBuf)
- unsigned ADC_GetOversample (unsigned long DeviceIndex)
- AIORESULT ADC_SetOversample (unsigned long DeviceIndex, unsigned char Oversample)
- AIORESULT WriteConfigBlock (unsigned long DeviceIndex)

- AIORESULT ReadConfigBlock (unsigned long DeviceIndex, AIOUSB_BOOL forceRead)
- AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock ∗config, unsigned gainCode, AIOUSB_BOOL differentialMode)
- AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock ∗config, unsigned channel)
- AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock ∗config, unsigned channel, unsigned gainCode)
- AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock ∗config, unsigned channel)
- AIORESULT AIOUSB_ADC_ExternalCal (unsigned long DeviceIndex, const double points[ ], int numPoints, unsigned short returnCalTable[ ], const char ∗saveFileName)
- AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long DeviceIndex, unsigned gain, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetAllGainCodeAndDiffMode instead")
- AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock ∗config, unsigned channel, AIOUSB_BOOL differentialMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetDifferentialMode")
- AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetCalMode")
- AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock ∗config, unsigned calMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetCalMode")
- AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock ∗config, unsigned overSample) ACCES_DEPRECATED("Please use ADCConfigBlockSetOversample")
- AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetOversample")
- unsigned AIOUSB_GetTriggerMode (const ADConfigBlock ∗config) ACCES_DEPRECATED("Please use ADCConfigBlockGetTriggerMode")
- AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock ∗config, unsigned triggerMode) ACCES_DEPRECATED("Please use ADCConfigBlockSetTriggerMode")
- unsigned AIOUSB_GetStartChannel (const ADConfigBlock ∗config)
- unsigned AIOUSB_GetEndChannel (const ADConfigBlock ∗config)
- AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock ∗config, unsigned startChannel, unsigned endChannel)
- unsigned long AIOUSB_SetStreamingBlockSize (unsigned long DeviceIndex, unsigned long BlockSize)
- AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetDiscardFirstSample (unsigned long DeviceIndex, AIOUSB_BOOL discard)
- AIOBuf ∗ CreateSmartBuffer (unsigned long DeviceIndex)

  After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.
- unsigned long AIOUSB_ADC_InternalCal (unsigned long DeviceIndex, AIOUSB_BOOL autoCal, unsigned short returnCalTable[ ], const char ∗saveFileName)

  Performs automatic calibration of the ADC.
- unsigned char ∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗config)
- void AIOUSB_SetRegister (ADConfigBlock ∗cb, unsigned int Register, unsigned char value)
- unsigned char AIOUSB_GetRegister (ADConfigBlock ∗cb, unsigned int Register)

24.110.1 Function Documentation

AIORET_TYPE ADC_GetScanV (unsigned long *DeviceIndex,* double ∗ *pBuf*)

Preferred way to get immediate scan readings.

Will Scan all channels (ie vectored) perform averaging and culling of data.

Parameters

DeviceIndex	
pBuf	

Returns

get raw A/D counts

Convert from A/D counts to volts; only the channels from startChannel to endChannel contain valid data, so we only convert those; pBuf[] is expected to contain entries for all the A/D channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

convert remaining channels to volts

AIORESULT ADC_RangeAll (unsigned long *DeviceIndex,* unsigned char ∗ *pGainCodes,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
pGainCodes	
bSingleEnded	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex,* unsigned long *ChannelIndex,* double ∗ *singlevoltage*)

Read one voltage input's current value.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ChannelIndex	number indicating which channel's data you wish to get
singlevoltage	a pointer to a double precision IEEE floating point num ber which will receive the value read

Note

This is a slow function

Returns

there is no guarantee that ChannelIndex, passed by the user, is within the current channel scan range; if it is not, then valid data cannot be returned; in addition, since we're only returning the data for a single channel, there's no need to scan all the channels; the Windows implementation attempts to improve performance by caching all the values read; but the technique is riddled with problems; first of all, it can easily return extremely stale data, without any indication to the user; secondly, it can return data for channels that weren't even scanned, without any indication to the user; thirdly, caching is unnecessary; if the user wants to read a single channel they can call ADC_GetChannelV(); if the user wants to improve performance by reading multiple channels they can call ADC_GetScanV(); so to address all these issues, we temporarily compress the scan range to just ChannelIndex and then restore it when we're done; so in this implementation all calls to ADC_GetChannelV() return "real-time" data for the specified channel

AIORET_TYPE ADC_GetScan (unsigned long *DeviceIndex,* unsigned short ∗ *pBuf*)

This simple function takes one scan of A/D data, in counts.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
pBuf	Pointer to an array of W ORDs. Each elem ent in the array will receive the value read from the corresponding A/D input channel. The array m ust be at least as large as the num ber of A/D input channels your product contains (16, 32, 64, 96, or 128) - but it is safe to always pass a pointer to an array of 128 W ORDs. Only elem ents in the array corresponding to A/D channels actually acquired during the scan will be updated: start-channel through end-channel, inclusive. Other values will rem ain unchanged.

Returns

AIORET_TYPE either AIOUSB_SUCCESS or a failure

pBuf[] is expected to contain entries for all the A/D channels, even though we may be reading only a few channels; so for cleanliness, we zero out the channels in pBuf[] that aren't going to be filled in with real readings

AIORET_TYPE AIOUSB_GetScan (unsigned long *DeviceIndex,* unsigned short *counts[ ]*)

Performs a scan and averages the voltage values.

Parameters

DeviceIndex	
counts	

Returns

Note

In theory, all the A/D functions, including AIOUSB_GetScan(), should work in all measurement modes, including calibration mode; in practice, however, the device will return only a single sample in calibration mode; therefore, users must be careful to select a single channel and set oversample to zero during calibration mode; attempting to read more than one channel or use an oversample setting of more than zero in calibration mode will result in a timeout error; as a convenience to the user we automatically impose this restriction here in AIOUSB_GetScan(); if the device is changed to permit normal use of the A/D functions in calibration mode, we will have to modify this function to somehow recognize which devices support that capability, or simply delete this restriction altogether and rely on the users' good judgment

The oversample setting dictates how many samples to take *in addition* to the primary sample; if oversample is zero, we take just one sample for each channel; if oversample is greater than zero then we average the primary sample and all of its over-samples; if the discardFirstSample setting is enabled, then we discard the primary sample, leaving just the over-samples; thus, if discardFirstSample is enabled, we must take at least one over-sample in order to have any data left; there's another complication: the device buffer is limited to a small number of samples, so we have to limit the number of over-samples to what the device buffer can accommodate, so the actual oversample setting depends on the number of channels being scanned; we also preserve and restore the original oversample setting specified by the user; since the user is expecting to average (1 + oversample) samples, then if discardFirstSample is enabled we simply always add one

Turn scan on and turn timer and external trigger off

make sure device buffer can accommodate this number of samples

Needs to be the correct values written out ... Should resemble (04|05) F0 0E

Compute the average of all the samples taken for each channel, discarding the first sample if that option is enabled; each byte in sampleBuffer[] is 1 of 2 bytes for each sample, the first byte being the LSB and the second byte the MSB, in other words, little-endian format; so for convenience we simply declare sampleBuffer[] to be of type 'unsigned short' and the data is already in the correct format; the device returns data only for the channels requested, from startChannel to endChannel; AIOUSB_GetScan() returns the averaged data readings in counts[], putting the reading for startChannel in counts[0], and the reading for endChannel in counts[numChannels-1]

AIORESULT ADC_GetConfig (unsigned long *DeviceIndex,* unsigned char ∗ *ConfigBuf,* unsigned long ∗ *ConfigBufSize*)

Determ ine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
ConfigBuf	a pointer to the first of an array of bytes for configuration data
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to read. W ill be set to the num ber of configuration bytes read

Returns

AIORESULT ADC_SetConfig (unsigned long *DeviceIndex,* unsigned char ∗ *pConfigBuf,* unsigned long ∗ *ConfigBufSize*)

Parameters

DeviceIndex	
pConfigBuf	
ConfigBufSize	

Returns

validate settings

AIORESULT ADC_Range1 (unsigned long *DeviceIndex,* unsigned long *ADChannel,* unsigned char *GainCode,* unsigned long *bSingleEnded*)

Parameters

DeviceIndex	
ADChannel	
GainCode	
bSingleEnded	

Returns

AIORESULT ADC_ADMode (unsigned long *DeviceIndex,* unsigned char *TriggerMode,* unsigned char *CalMode*)

Parameters

DeviceIndex	
TriggerMode	
CalMode	

Returns

AIORESULT ADC_SetScanLimits (unsigned long *DeviceIndex,* unsigned long *StartChannel,* unsigned long *EndChannel*)

Parameters

DeviceIndex	
StartChannel	
EndChannel	

Returns

AIORESULT ADC_GetMaxClockRate (unsigned long *ProductID,* unsigned int *num_channels,* unsigned int *num_oversamples*)

Returns the maximum clock rate for the product in question and the number of oversamples + number of channels for the device.

Parameters

ProductID	produc we are looking up
num_channels	Number of channels we will be sampling on
num_-oversamples	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

AIORESULT ADC_SetCal (unsigned long *DeviceIndex,* const char ∗ *CalFileName*)

Parameters

DeviceIndex	
CalFileName	

Returns

AIORESULT ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *ProductID*)

AIORESULT ADC_Initialize (unsigned long *DeviceIndex,* unsigned char * *pConfigBuf,* unsigned long * *ConfigBufSize,* const char * *CalFileName*)

Determ ine information about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
pConfigBuf	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
ConfigBufSize	a pointer to a variable holding the num ber of configuration bytes to write.
CalFileName	the file nam e of a calibration file, or a com m and string. See ADC_SetCal() for details.

Returns

AIOUSB_SUCCESS if successful, error otherwise.

AIORESULT ADC_BulkAcquire (unsigned long *DeviceIndex,* unsigned long *BufSize,* void * *pBuf*)

Determine inform ation about the device found at a specific DeviceIndex.

Parameters

DeviceIndex	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
BufSize	the size, in bytes, of the buffer to receive the data
pBuf	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return im m ediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or m oved. Use ADC_BulkPoll() to query this background operation.

we initialize the worker thread status here in case the thread doesn't start for some reason, such as an improperly locked mutex; this pre-initialization is necessary so that the thread status doesn't make it appear as though the worker thread has completed successfully

AIORESULT ADC_BulkPoll (unsigned long *DeviceIndex,* unsigned long * *BytesLeft*)

Parameters

DeviceIndex	
BytesLeft	

Returns

AIORESULT ADC_InitFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_CreateFastITConfig (unsigned long *DeviceIndex,* int *size*)

Creates FastIT Config Blocks.

Parameters

DeviceIndex	
size	

Returns

AIORESULT ADC_ResetFastITScanV (unsigned long *DeviceIndex*)

AIORESULT ADC_SetFastITScanVChannels (unsigned long *DeviceIndex,* unsigned long *NewChannels*)

AIORESULT ADC_GetFastITScanV (unsigned long *DeviceIndex,* double ∗ *pData*)

Parameters

DeviceIndex	
pData	buffer we will write data into

Returns

AIORESULT ADC_GetITScanV (unsigned long *DeviceIndex,* double ∗ *pBuf*)

Parameters

DeviceIndex	
pBuf	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex,* unsigned char *Oversample*)

Parameters

DeviceIndex	
Oversample	

Returns

AIORESULT WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

AIORESULT ReadConfigBlock (unsigned long *DeviceIndex,* AIOUSB_BOOL *forceRead*)

Parameters

DeviceIndex	
forceRead	

AIORET_TYPE AIOUSB_SetAllGainCodeAndDiffMode (ADConfigBlock * *config,* unsigned *gainCode,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_GetGainCode (const ADConfigBlock * *config,* unsigned *channel*)

AIORET_TYPE AIOUSB_SetGainCode (ADConfigBlock * *config,* unsigned *channel,* unsigned *gainCode*)

Parameters

config	
channel	
gainCode	

AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock * *config,* unsigned *channel*)

Parameters

config	
channel	

Returns

AIORESULT AIOUSB_ADC_ExternalCal (unsigned long *DeviceIndex,* const double *points[ ],* int *numPoints,* unsigned short *returnCalTable[ ],* const char * *saveFileName*)

Note

```
* sort table into ascending order by input voltage; then verify that both the
* input voltages and the measured counts are unique and uniformly increasing;
* since the user's points[] array is declared to be 'const' we need to allocate
* a working table that we can sort; in addition, we want to allocate space for
* a slope and offset between each pair of points; so while points[] is like a
* table with numPoints rows and two columns (input voltage, measured counts),
* the working table effectively has the same number of rows, but four columns
* (input voltage, measured counts, slope, offset)
*
*       points[] format:
*       +-----------------+       +-----------------+
*   [0] |  input voltage  |   [1] | measured counts |
*       |=================|       |=================|
*   [2] |  input voltage  |   [3] | measured counts |
*       |=================|       |=================|
*                         ...
*       |=================|       |=================|
* [n-2] |  input voltage  | [n-1] | measured counts |
*       +-----------------+       +-----------------+
* 'n' is not numPoints, but numPoints*2
*
```

if table of calibration points looks good, then proceed to calculate slopes and offsets of line segments between points; we verified that no two points in the table are equal, so we should not get any division by zero errors

```
the calibration table really only applies to one range if precision is our
* objective; therefore, we assume that all the channels are configured for the
* same range during calibration mode, and that the user is still using the same
* range now as when they collected the calibration data points; if all these
* assumptions are correct, then we can use the range setting for channel 0
*
* the calculations are based on the following model:
*   mcounts = icounts x slope + offset
* where,
*   mcounts is the measured counts (reported by an uncalibrated A/D)
*   icounts is the input counts from an external voltage source
*   slope is the gain error inherent in the A/D and associated circuitry
*   offset is the offset error inherent in the A/D and associated circuitry
* to reverse the effect of these slope and offset errors, we use this equation:
*   ccounts = ( mcounts - offset ) / slope
* where,
*   ccounts is the corrected counts
* we calculate the slope and offset using these equations:
*   slope = ( mcounts[s] - mcounts[z] ) / ( icounts[m] - icounts[z] )
*   offset = mcounts[z] - icounts[z] x slope
* where,
*   [s] is the reading at "span" (the upper reference point)
*   [z] is the reading at "zero" (the lower reference point)
* in the simplest case, we would use merely two points to correct the entire voltage
* range of the A/D; in such a simple case, the "zero" point would be a point near 0V,
* and the "span" point would be a point near the top of the voltage range, such as 9.9V;
* however, since this function is actually calculating a whole bunch of slope/offset
* correction factors, one between each pair of points, "zero" refers to the lower of
* two points, and "span" refers to the higher of the two points
*
```

generate calibration table using the equation ccounts = (mcounts – offset) / slope described above; each slope/offset pair in workingPoints[] describes the line segment running between the *previous* point and the current one; in addition, the first row in workingPoints[] doesn't contain a valid slope/offset pair because there is no previous point before the first one (!), so we stretch the first line segment (between points 0 and 1) backward to the beginning of the A/D count range; similarly, since the highest calibration point is probably not right at the top of the A/D count range, we stretch the highest line segment (between points n-2 and n-1) up to the top of the A/D count range

AIORESULT ADC_SetAllGainCodeAndDiffMode (unsigned long *DeviceIndex,* unsigned *gain,* AIOUSB_BOOL *differentialMode*)

AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock ∗ *config,* unsigned *channel,* AIOUSB_BOOL *differentialMode*)

Parameters

config	
channel	
differentialMode	

AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock ∗ *config*)

AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock ∗ *config,* unsigned *calMode*)

Parameters

config	
calMode	

AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock ∗ *config,* unsigned *overSample*)

AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock ∗ *config*)

unsigned AIOUSB_GetTriggerMode (const ADConfigBlock ∗ *config*)

Parameters

config	

Returns

AIORET_TYPE AIOUSB_SetTriggerMode (ADConfigBlock ∗ *config,* unsigned *triggerMode*)

unsigned AIOUSB_GetStartChannel (const ADConfigBlock ∗ *config*)

unsigned AIOUSB_GetEndChannel (const ADConfigBlock ∗ *config*)

AIORET_TYPE AIOUSB_SetScanRange (ADConfigBlock ∗ *config,* unsigned *startChannel,* unsigned *endChannel*)

Parameters

config	
startChannel	
endChannel	

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex,* unsigned long *BlockSize*)

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

DeviceIndex	

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex,* AIOUSB_BOOL *discard*)

Parameters

DeviceIndex	
discard	

Returns

AIOBuf∗ CreateSmartBuffer (unsigned long *DeviceIndex*)

After setting up your oversamples and such, creates a new AIOBuf object that can be used for BulkAcquiring.

Parameters

DeviceIndex	

Returns

AIOBuf ∗ new Buffer object for BulkAcquire methods

Todo Replace 16 with correct channels returned by probing the device

unsigned long AIOUSB_ADC_InternalCal (unsigned long *DeviceIndex,* AIOUSB_BOOL *autoCal,* unsigned short *returnCalTable[ ],* const char ∗ *saveFileName*)

Performs automatic calibration of the ADC.

Parameters

DeviceIndex	
autoCal	
returnCalTable	
saveFileName	

Returns

unsigned char∗ ADC_GetADConfigBlock_Registers (ADConfigBlock ∗ *config*)

void AIOUSB_SetRegister (ADConfigBlock ∗ *cb,* unsigned int *Register,* unsigned char *value*)

unsigned char AIOUSB_GetRegister (ADConfigBlock ∗ *cb,* unsigned int *Register*)

24.111 lib/AIOUSB_Core.c File Reference

General header files for the AIOUSB library.

```
#include "ADCConfigBlock.h"
#include "AIOUSB_Core.h"
#include "AIODeviceTable.h"
#include "AIOUSB_ADC.h"
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <libusb.h>
```

Macros

- #define BACKTRACE_DEBUG(n) {}
- #define AIOUSB_ENABLE_MUTEX

Functions

- AIOUSB_BOOL AIOUSB_Lock ()

    Notes on mutual exclusion / threading:
- AIOUSB_BOOL AIOUSB_UnLock ()
- AIORET_TYPE AIOUSB_ResetChip (unsigned long DeviceIndex)
- unsigned long AIOUSB_Validate_Lock (unsigned long ∗DeviceIndex)
- unsigned long AIOUSB_Validate (unsigned long ∗DeviceIndex)
- unsigned long ResolveDeviceIndex (unsigned long DeviceIndex)
- DeviceDescriptor ∗ DeviceTableAtIndex (unsigned long DeviceIndex)
- DeviceDescriptor ∗ DeviceTableAtIndex_Lock (unsigned long DeviceIndex)
- DeviceDescriptor ∗ AIOUSB_GetDevice (unsigned long DeviceIndex)
- ADConfigBlock ∗ AIOUSB_GetConfigBlock (DeviceDescriptor ∗dev)
- long AIOUSB_GetStreamingBlockSize (unsigned long DeviceIndex)

    This function is deprecated.
- unsigned long AIOUSB_SetStreamingBlockSize (unsigned long DeviceIndex, unsigned long BlockSize)
- unsigned long AIOUSB_ClearFIFO (unsigned long DeviceIndex, FIFO_Method Method)
- const char ∗ AIOUSB_GetVersion ()
- const char ∗ AIOUSB_GetVersionDate ()
- AIORESULT AIOUSB_GetMiscClock (unsigned long DeviceIndex)
- AIORESULT AIOUSB_SetMiscClock (unsigned long DeviceIndex, double clockHz)
- unsigned AIOUSB_GetCommTimeout (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetCommTimeout (unsigned long DeviceIndex, unsigned timeout)
- unsigned long AIOUSB_Validate_Device (unsigned long DeviceIndex)
- AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock ∗config, unsigned long DeviceIndex, AIOUSB_BOOL defaults)
- unsigned long AIOUSB_ADC_LoadCalTable (unsigned long DeviceIndex, const char ∗fileName)
- unsigned long AIOUSB_ADC_SetCalTable (unsigned long DeviceIndex, const unsigned short calTable[ ])
- unsigned long GenericVendorWrite (unsigned long deviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void ∗bufData, unsigned long ∗bytes_written)

    Performs a generic vendor USB write.
- unsigned long GenericVendorRead (unsigned long deviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void ∗bufData, unsigned long ∗bytes_read)

    Performs basic low level USB vendor request.

Variables

- int aio_errno
- ProductIDName productIDNameTable [ ]

24.111.1 Detailed Description

General header files for the AIOUSB library.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.111.2 Macro Definition Documentation

#define BACKTRACE_DEBUG(*n* **) {}**

#define AIOUSB_ENABLE_MUTEX

24.111.3 Function Documentation

AIOUSB_BOOL AIOUSB_Lock (void)

Notes on mutual exclusion / threading:

- Our mutual exclusion scheme is *not* intended to be bulletproof. It's primarily intended to ensure mutually exclusive access to deviceTable[] and other global variables. It does NOT ensure mutually exclusive access to the USB bus. In fact, we want to permit threads to communicate with multiple devices simultaneously, to the extent possible with USB.

- Nor does this scheme prevent multiple threads from altering the configuration of the same device or communicating with the same device. In other words, it's entirely possible for one thread to configure and communicate with a device, only to have another thread come along and to the same. It's up to the users of this library to ensure that such a scenario doesn't occur.

- This library does seek to permit one thread to control one device, and another thread to control another device. Each thread may then safely communicate with its own device and alter the portion of deviceTable[] that pertains to its device.

- Our mutual exclusion scheme also permits two threads to cooperate in the operation of a single device, such as in cases where a background thread does the actual work and the foreground thread monitors the progress. In such a case, the background thread might update a status variable which the foreground thread monitors. This form of resource sharing is supported by our mutual exclusion scheme.

AIOUSB_BOOL AIOUSB_UnLock (void)

AIORET_TYPE AIOUSB_ResetChip (unsigned long ***DeviceIndex*** **)**

unsigned long AIOUSB_Validate_Lock (unsigned long ∗ ***DeviceIndex*** **)**

unsigned long AIOUSB_Validate (unsigned long ∗ ***DeviceIndex*** **)**

unsigned long ResolveDeviceIndex (unsigned long ***DeviceIndex*** **)**

DeviceDescriptor∗ DeviceTableAtIndex (unsigned long ***DeviceIndex*** **)**

DeviceDescriptor∗ DeviceTableAtIndex_Lock (unsigned long ***DeviceIndex*** **)**

Todo Replace AIOUSB_Lock() with thread safe lock on a per device index basis

Insert correct error messages into global error string in case of failure

DeviceDescriptor∗ AIOUSB_GetDevice (unsigned long ***DeviceIndex*** **)**

ADConfigBlock∗ AIOUSB_GetConfigBlock (DeviceDescriptor ∗ ***dev*** **)**

long AIOUSB_GetStreamingBlockSize (unsigned long ***DeviceIndex*** **)**

This function is deprecated.

Parameters

DeviceIndex	

Returns

0 or greater if the blocksize is correct, negative number on error

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex*, unsigned long *BlockSize*)

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex*, FIFO_Method *Method*)

const char∗ AIOUSB_GetVersion (void)

const char∗ AIOUSB_GetVersionDate (void)

AIORESULT AIOUSB_GetMiscClock (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_SetMiscClock (unsigned long *DeviceIndex*, double *clockHz*)

unsigned AIOUSB_GetCommTimeout (unsigned long *DeviceIndex*)

unsigned long AIOUSB_SetCommTimeout (unsigned long *DeviceIndex*, unsigned *timeout*)

unsigned long AIOUSB_Validate_Device (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock ∗ *config*, unsigned long *DeviceIndex*, AIOUSB_BOOL *defaults*)

Parameters

config	
DeviceIndex	
defaults	

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex*, const char ∗ *fileName*)

Parameters

DeviceIndex	
fileName	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex*, const unsigned short *calTable[]*)

Parameters

DeviceIndex	
calTable	

Returns

unsigned long GenericVendorWrite (unsigned long *deviceIndex*, unsigned char *Request*, unsigned short *Value*, unsigned short *Index*, void ∗ *bufData*, unsigned long ∗ *bytes_written*)

Performs a generic vendor USB write.

unsigned long GenericVendorRead (unsigned long *deviceIndex*, unsigned char *Request*, unsigned short *Value*, unsigned short *Index*, void ∗ *bufData*, unsigned long ∗ *bytes_read*)

Performs basic low level USB vendor request.

Returns

24.111.4 Variable Documentation

int aio_errno

ProductIDName productIDNameTable[ ]

24.112 lib/AIOUSB_Core.h File Reference

```
#include "AIOUSBDevice.h"
#include "libusb.h"
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
```

Data Structures

- struct BulkAcquireWorkerParams
- struct aiousboption
- struct aioerror
- struct ProductIDName
- struct ADRange

Macros

- #define PUBLIC_EXTERN extern
- #define PRIVATE
- #define PROD_NAME_SIZE 40

Typedefs

- typedef struct aiousboption AIOOption
- typedef struct aioerror AIOError

Functions

- unsigned long ADC_ResetDevice (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long DeviceIndex)
- AIORET_TYPE AIOUSB_ResetChip (unsigned long DeviceIndex)
- AIOUSB_BOOL AIOUSB_Lock (void)

  Notes on mutual exclusion / threading:
- AIOUSB_BOOL AIOUSB_UnLock (void)
- AIORESULT AIOUSB_InitTest (void)
- AIORESULT AIOUSB_Validate (unsigned long ∗DeviceIndex)
- AIORESULT AIOUSB_Validate_Lock (unsigned long ∗DeviceIndex)
- DeviceDescriptor ∗ DeviceTableAtIndex (unsigned long DeviceIndex)
- DeviceDescriptor ∗ DeviceTableAtIndex_Lock (unsigned long DeviceIndex)
- DeviceDescriptor ∗ AIOUSB_GetDevice (unsigned long DeviceIndex)
- ADConfigBlock ∗ AIOUSB_GetConfigBlock (DeviceDescriptor ∗dev)
- AIORESULT AIOUSB_SetMiscClock (unsigned long DeviceIndex, double clockHz)
- AIORESULT AIOUSB_GetMiscClock (unsigned long DeviceIndex)
- unsigned long AIOUSB_SetCommTimeout (unsigned long DeviceIndex, unsigned timeout)
- unsigned AIOUSB_GetCommTimeout (unsigned long DeviceIndex)
- const char ∗ AIOUSB_GetVersion (void)
- const char ∗ AIOUSB_GetVersionDate (void)
- const char ∗ AIOUSB_GetResultCodeAsString (unsigned long value)
- unsigned short AIOUSB_VoltsToCounts (unsigned long DeviceIndex, unsigned channel, double volts)
- unsigned long AIOUSB_ADC_LoadCalTable (unsigned long DeviceIndex, const char ∗fileName)
- unsigned long AIOUSB_ADC_SetCalTable (unsigned long DeviceIndex, const unsigned short calTable[ ])
- unsigned long AIOUSB_ClearFIFO (unsigned long DeviceIndex, FIFO_Method Method)
- long AIOUSB_GetStreamingBlockSize (unsigned long DeviceIndex)

  This function is deprecated.
- AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock ∗config, unsigned long DeviceIndex, AIOUSB_BOOL defaults)

- AIORESULT GenericVendorRead (unsigned long deviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void ∗bufData, unsigned long ∗bytes_read)

  Performs basic low level USB vendor request.
- AIORESULT GenericVendorWrite (unsigned long DeviceIndex, unsigned char Request, unsigned short Value, unsigned short Index, void ∗bufData, unsigned long ∗bytes_write)

  Performs a generic vendor USB write.
- AIORESULT AIOUSB_Validate_Device (unsigned long DeviceIndex)

Variables

- int aio_errno
- struct ADRange adRanges [AD_NUM_GAIN_CODES]
- unsigned long AIOUSB_INIT_PATTERN
- unsigned long aiousbInit

24.112.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

Note

All the API functions that DO NOT begin "AIOUSB_" are standard API functions, largely documented in the `http://accesio.com/MANUALS/USB%20Software%20Reference.pdf`. The functions that DO begin with "AIOUSB_" are "extended" API functions added to the Linux implementation. Source code lines in this sample program that are prefixed with the comment highlight calls to the AIOUSB API.

See Also

Compilation
CmakeCompilation

24.112.2 Macro Definition Documentation

#define PUBLIC_EXTERN extern

#define PRIVATE

#define PROD_NAME_SIZE 40

24.112.3 Typedef Documentation

typedef struct aiousboption AIOOption

typedef struct aioerror AIOError

24.112.4 Function Documentation

unsigned long ADC_ResetDevice (unsigned long *DeviceIndex*)

AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long *DeviceIndex*)

AIORET_TYPE AIOUSB_ResetChip (unsigned long *DeviceIndex*)

AIOUSB_BOOL AIOUSB_Lock (void)

Notes on mutual exclusion / threading:

- Our mutual exclusion scheme is *not* intended to be bulletproof. It's primarily intended to ensure mutually exclusive access to deviceTable[] and other global variables. It does NOT ensure mutually exclusive access to the USB bus. In fact, we want to permit threads to communicate with multiple devices simultaneously, to the extent possible with USB.
- Nor does this scheme prevent multiple threads from altering the configuration of the same device or communicating with the same device. In other words, it's entirely possible for one thread to configure and communicate with a device, only to have another thread come along and to the same. It's up to the users of this library to ensure that such a scenario doesn't occur.
- This library does seek to permit one thread to control one device, and another thread to control another device. Each thread may then safely communicate with its own device and alter the portion of deviceTable[] that pertains to its device.
- Our mutual exclusion scheme also permits two threads to cooperate in the operation of a single device, such as in cases where a background thread does the actual work and the foreground thread monitors the progress. In such a case, the background thread might update a status variable which the foreground thread monitors. This form of resource sharing is supported by our mutual exclusion scheme.

AIOUSB_BOOL AIOUSB_UnLock (void)

AIORESULT AIOUSB_InitTest (void)

AIORESULT AIOUSB_Validate (unsigned long ∗ *DeviceIndex*)

AIORESULT AIOUSB_Validate_Lock (unsigned long ∗ *DeviceIndex*)

DeviceDescriptor∗ DeviceTableAtIndex (unsigned long *DeviceIndex*)

DeviceDescriptor∗ DeviceTableAtIndex_Lock (unsigned long *DeviceIndex*)

Todo Replace AIOUSB_Lock() with thread safe lock on a per device index basis

Insert correct error messages into global error string in case of failure

DeviceDescriptor∗ AIOUSB_GetDevice (unsigned long *DeviceIndex*)

ADConfigBlock∗ AIOUSB_GetConfigBlock (DeviceDescriptor ∗ *dev*)

AIORESULT AIOUSB_SetMiscClock (unsigned long *DeviceIndex*, double *clockHz*)

AIORESULT AIOUSB_GetMiscClock (unsigned long *DeviceIndex*)

unsigned long AIOUSB_SetCommTimeout (unsigned long *DeviceIndex,* unsigned *timeout*)

unsigned AIOUSB_GetCommTimeout (unsigned long *DeviceIndex*)

const char∗ AIOUSB_GetVersion (void)

const char∗ AIOUSB_GetVersionDate (void)

const char∗ AIOUSB_GetResultCodeAsString (unsigned long *value*)

build index of result codes

unsigned short AIOUSB_VoltsToCounts (unsigned long *DeviceIndex,* unsigned *channel,* double *volts*)

Parameters

DeviceIndex	
channel	
volts	

Returns

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex,* const char ∗ *fileName*)

Parameters

DeviceIndex	
fileName	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex,* const unsigned short *calTable[ ]*)

Parameters

DeviceIndex	
calTable	

Returns

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex,* FIFO_Method *Method*)

long AIOUSB_GetStreamingBlockSize (unsigned long *DeviceIndex*)

This function is deprecated.

Parameters

DeviceIndex	

Returns

0 or greater if the blocksize is correct, negative number on error

AIORESULT AIOUSB_InitConfigBlock (ADConfigBlock ∗ *config,* unsigned long *DeviceIndex,* AIOUSB_BOOL *defaults*)

Parameters

config	
DeviceIndex	
defaults	

AIORESULT GenericVendorRead (unsigned long *deviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void ∗ *bufData,* unsigned long ∗ *bytes_read*)

Performs basic low level USB vendor request.

Returns

AIORESULT GenericVendorWrite (unsigned long *DeviceIndex,* unsigned char *Request,* unsigned short *Value,* unsigned short *Index,* void ∗ *bufData,* unsigned long ∗ *bytes_write*)

Performs a generic vendor USB write.

AIORESULT AIOUSB_Validate_Device (unsigned long *DeviceIndex*)

24.112.5 Variable Documentation

int aio_errno

struct ADRange adRanges[AD_NUM_GAIN_CODES]

unsigned long AIOUSB_INIT_PATTERN

unsigned long aiousbInit

24.113 lib/AIOUSB_CTR.c File Reference

Counter functionality.

```
#include "AIOUSB_CTR.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Log.h"
#include <math.h>
```

Macros

- #define RETURN_IF_INVALID_INPUT(d, r, f)
- #define JUMP_IF_INVALID_INPUT(d, r, f, g)
- #define JUMP_IF_NO_VALID_USB(d, r, f, u, g)

Functions

- AIORET_TYPE CTR_8254Mode (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long Counter-Index, unsigned long Mode)
- AIORET_TYPE CTR_8254Load (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long Counter-Index, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ReadModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue, unsigned short ∗pReadValue)
- AIORET_TYPE CTR_8254Read (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long Counter-Index, unsigned short ∗pReadValue)
- AIORET_TYPE CTR_8254ReadAll (unsigned long DeviceIndex, unsigned short ∗pData)
- AIORET_TYPE CTR_8254ReadStatus (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short ∗pReadValue, unsigned char ∗pStatus)
- AIORET_TYPE CTR_CalculateCountersForClock (int hz, int ∗diva, int ∗divb)

  Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.

- AIORET_TYPE CTR_StartOutputFreq (unsigned long DeviceIndex, unsigned long BlockIndex, double ∗pHz)
- AIORET_TYPE CTR_8254SelectGate (unsigned long DeviceIndex, unsigned long GateIndex)
- AIORET_TYPE CTR_8254ReadLatched (unsigned long DeviceIndex, unsigned short ∗pData)

24.113.1 Detailed Description

Counter functionality.

Author

Format:

an <ae>

Date

Format:

ad

Copyright:

©

24.113.2 Macro Definition Documentation

#define RETURN_IF_INVALID_INPUT(*d,* *r,* *f*)

Value:

```
do { \
        if( !d )                                                  \
            return (AIORET_TYPE)-AIOUSB_ERROR_INVALID_INDEX;
        \
        if( ( r = f ) != AIOUSB_SUCCESS ) {
      \
            AIOUSB_UnLock();                                      \
            return r;                                             \
        }                                                         \
    } while (0)
```

#define JUMP_IF_INVALID_INPUT(*d,* *r,* *f,* *g*)

Value:

```
do { \
        if ( !d ) { \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;\
            goto g;\
        } else if ( (r = f) != AIOUSB_SUCCESS ) { \
            goto g;\
        } \
    } while (0)
```

#define JUMP_IF_NO_VALID_USB(*d,* *r,* *f,* *u,* *g*)

Value:

```
do {              \
        if ( !d ) {                                          \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;              \
            goto g;                                          \
        } else if ( ( r = f ) != AIOUSB_SUCCESS  ) {         \
            goto g;                                          \
        } else if ( !(u = AIOUSBDeviceGetUSBHandle( d )))  { \
            r = -AIOUSB_ERROR_INVALID_USBDEVICE;             \
            goto g;                                          \
        }                                                    \
    } while (0 )
```

24.113.3 Function Documentation

AIORET_TYPE CTR_8254Mode (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode*)

AIORET_TYPE CTR_8254Load (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ReadModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254Read (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254ReadAll (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

AIORET_TYPE CTR_8254ReadStatus (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue,* unsigned char ∗ *pStatus*)

AIORET_TYPE CTR_CalculateCountersForClock (int *hz,* int ∗ *diva,* int ∗ *divb*)

Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.

Parameters

	hz	
out	*diva*	Divisor A to be calculated
out	*divb*	Divisor B to be calculated

Returns

>= 0 if succesful, - if failure

AIORET_TYPE CTR_StartOutputFreq (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* double ∗ *pHz*)

AIORET_TYPE CTR_8254SelectGate (unsigned long *DeviceIndex,* unsigned long *GateIndex*)

AIORET_TYPE CTR_8254ReadLatched (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

24.114 lib/AIOUSB_CTR.h File Reference

```
#include "AIOTypes.h"
#include "AIOUSB_Core.h"
```

Functions

- AIORET_TYPE CTR_CalculateCountersForClock (int hz, int ∗diva, int ∗divb)

  Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.
- AIORET_TYPE CTR_8254Mode (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode)
- AIORET_TYPE CTR_8254Load (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue)
- AIORET_TYPE CTR_8254ReadModeLoad (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned long Mode, unsigned short LoadValue, unsigned short ∗pReadValue)
- AIORET_TYPE CTR_8254Read (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short ∗pReadValue)
- AIORET_TYPE CTR_8254ReadAll (unsigned long DeviceIndex, unsigned short ∗pData)
- AIORET_TYPE CTR_8254ReadStatus (unsigned long DeviceIndex, unsigned long BlockIndex, unsigned long CounterIndex, unsigned short ∗pReadValue, unsigned char ∗pStatus)
- AIORET_TYPE CTR_StartOutputFreq (unsigned long DeviceIndex, unsigned long BlockIndex, double ∗pHz)
- AIORET_TYPE CTR_8254SelectGate (unsigned long DeviceIndex, unsigned long GateIndex)
- AIORET_TYPE CTR_8254ReadLatched (unsigned long DeviceIndex, unsigned short ∗pData)

24.114.1 Function Documentation

AIORET_TYPE CTR_CalculateCountersForClock (int *hz,* int ∗ *diva,* int ∗ *divb*)

Calculates the register values for buf->divisora, and buf->divisorb to create an output clock that matches the value stored in buf->hz ∗.

Parameters

	hz	
`out`	*diva*	Divisor A to be calculated
`out`	*divb*	Divisor B to be calculated

Returns

>= 0 if succesful, - if failure

AIORET_TYPE CTR_8254Mode (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode*)

AIORET_TYPE CTR_8254Load (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue*)

AIORET_TYPE CTR_8254ReadModeLoad (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned long *Mode,* unsigned short *LoadValue,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254Read (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue*)

AIORET_TYPE CTR_8254ReadAll (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

AIORET_TYPE CTR_8254ReadStatus (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* unsigned long *CounterIndex,* unsigned short ∗ *pReadValue,* unsigned char ∗ *pStatus*)

AIORET_TYPE CTR_StartOutputFreq (unsigned long *DeviceIndex,* unsigned long *BlockIndex,* double ∗ *pHz*)

AIORET_TYPE CTR_8254SelectGate (unsigned long *DeviceIndex,* unsigned long *GateIndex*)

AIORET_TYPE CTR_8254ReadLatched (unsigned long *DeviceIndex,* unsigned short ∗ *pData*)

24.115 lib/AIOUSB_CustomEEPROM.c File Reference

General header files for EEProm functionality.

```
#include "AIOUSB_CustomEEPROM.h"
#include "AIOUSB_Core.h"
#include "AIODeviceTable.h"
```

Macros

- #define EXIT_FN_IF_NO_VALID_USB(d, r, f, u, g)

Functions

- unsigned long CustomEEPROMWrite (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long DataSize, void ∗Data)

  EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)
- unsigned long CustomEEPROMRead (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long ∗DataSize, void ∗Data)

  EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

24.115.1 Detailed Description

General header files for EEProm functionality.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.115.2 Macro Definition Documentation

#define EXIT_FN_IF_NO_VALID_USB(*d, r, f, u, g*)

Value:

```
do {            \
        if ( !d ) {                                             \
            r = -AIOUSB_ERROR_DEVICE_NOT_FOUND;                 \
            goto g;                                             \
        } else if ( ( r = f ) != AIOUSB_SUCCESS  ) {            \
            goto g;                                             \
        } else if ( !(u = AIOUSBDeviceGetUSBHandle( d )))  {    \
            r = -AIOUSB_ERROR_INVALID_USBDEVICE;                \
            goto g;                                             \
        }                                                       \
    } while (0 )
```

24.115.3 Function Documentation

unsigned long CustomEEPROMWrite (unsigned long *DeviceIndex,* unsigned long *StartAddress,* unsigned long *DataSize,* void ∗ *Data*)

EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

unsigned long CustomEEPROMRead (unsigned long *DeviceIndex,* unsigned long *StartAddress,* unsigned long ∗ *DataSize,* void ∗ *Data*)

EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

24.116 lib/AIOUSB_CustomEEPROM.h File Reference

```
#include "AIOTypes.h"
```

Functions

- unsigned long CustomEEPROMWrite (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long DataSize, void ∗Data)

EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

- unsigned long CustomEEPROMRead (unsigned long DeviceIndex, unsigned long StartAddress, unsigned long ∗DataSize, void ∗Data)

  EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

24.116.1 Function Documentation

unsigned long CustomEEPROMWrite (unsigned long *DeviceIndex*, unsigned long *StartAddress*, unsigned long *DataSize*, void ∗ *Data*)

EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

unsigned long CustomEEPROMRead (unsigned long *DeviceIndex*, unsigned long *StartAddress*, unsigned long ∗ *DataSize*, void ∗ *Data*)

EEPROM layout: program code: 0x0000 -> EEPROM_CUSTOM_BASE_ADDRESS - 1 user space : EEPROM_CUSTOM_BASE_ADDRESS -> EEPROM_CUSTOM_BASE_ADDRESS + EEPROM_CUSTOM_MAX_ADDRESS - 1 (user space is addressed as 0 -> EEPROM_CUSTOM_MAX_ADDRESS - 1)

24.117 lib/AIOUSB_DAC.c File Reference

Core code to handle DACs on AIOUSB devices.

```
#include "AIOUSB_Core.h"
#include "AIODeviceTable.h"
#include <math.h>
#include <string.h>
```

Functions

- unsigned long DACDirect (unsigned long DeviceIndex, unsigned short Channel, unsigned short Value)
- unsigned long DACMultiDirect (unsigned long DeviceIndex, unsigned short ∗pDACData, unsigned long DACDataCount)

  pDACData is an array of DACDataCount channel/count 16-bit word pairs:
- unsigned long DACSetBoardRange (unsigned long DeviceIndex, unsigned long RangeCode)
- unsigned long DACOutputOpen (unsigned long DeviceIndex, double ∗pClockHz)
- unsigned long DACOutputClose (unsigned long DeviceIndex, unsigned long bWait)
- unsigned long DACOutputCloseNoEnd (unsigned long DeviceIndex, unsigned long bWait)
- unsigned long DACOutputSetCount (unsigned long DeviceIndex, unsigned long NewCount)
- unsigned long DACOutputFrame (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short ∗FrameData)
- unsigned long DACOutputFrameRaw (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short ∗FrameData)
- unsigned long DACOutputStart (unsigned long DeviceIndex)
- unsigned long DACOutputSetInterlock (unsigned long DeviceIndex, unsigned long bInterlock)

24.117.1 Detailed Description

Core code to handle DACs on AIOUSB devices.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.117.2 Function Documentation

unsigned long DACDirect (unsigned long *DeviceIndex,* unsigned short *Channel,* unsigned short *Value*)

unsigned long DACMultiDirect (unsigned long *DeviceIndex,* unsigned short ∗ *pDACData,* unsigned long *DACDataCount*)

pDACData is an array of DACDataCount channel/count 16-bit word pairs:

```
*   +----------------+
*   |    channel     | word 0
*   |----------------|
*   |     count      | word 1
*   |----------------|
*         ...
*   |----------------|
*   |    channel     |
*   |----------------|
*   |     count      | word ( DACDataCount * 2 ) - 1
*   +----------------+
*
* this array has to be converted to a different format when passed to the board:
*        Block 0
*   +----------------+
*   |   chan mask    | byte 0
*   |----------------|
*   |  chan 0 count  | bytes 1-2
*   |----------------|
*         ...
*   |----------------|
*   |  chan 6 count  | bytes 13-14
*   |----------------|
*   |  chan 7 count  | bytes 15-16
*   +----------------+
*        Block 1
*   +----------------+
*   |   chan mask    | byte 17
*   |----------------|
*   |  chan 0 count  | bytes 18-19
*   |----------------|
*         ...
*   |----------------|
*   |  chan 6 count  | bytes 30-31
*   |----------------|
*   |  chan 7 count  | bytes 32-33
*   +----------------+
*         ...
*        Block n
*   +----------------+
*   |   chan mask    |
*   |----------------|
*   |  chan 0 count  |
*   |----------------|
*         ...
*   |----------------|
*   |  chan 6 count  |
*   |----------------|
*   |  chan 7 count  | bytes ( ( 17 * n ) - 2 ) - ( ( 17 * n ) - 1 )
*   +----------------+
*
```

the channel mask (the first byte of each block) has a bit set to one for each channel whose output is to be set; the count values are zero for channels that aren't to be set; for example, a mask of 0x01 would write to only channel 0 on a given block; a mask of 0x80 would write to only channel 7

since the DAC configuration blocks are contiguous, the byte offset to a channel's count within the buffer containing all the configuration blocks can be calculated as: offset = (channel ∗ sizeof(unsigned short)) + (channel / 8) + 1; although this calculation is correct, it's difficult to follow, so the code below uses a slightly less efficient calculation that's easier to understand when sending the DAC configuration blocks to the device we have to send all the blocks from block 0 up to the block containing the highest channel number being set determine highest channel number addressed in pDACData; no checking is performed to ensure that the same channel is not set more than once

unsigned long DACSetBoardRange (unsigned long *DeviceIndex,* unsigned long *RangeCode*)

unsigned long DACOutputOpen (unsigned long *DeviceIndex,* double ∗ *pClockHz*)

unsigned long DACOutputClose (unsigned long *DeviceIndex,* unsigned long *bWait*)

unsigned long DACOutputCloseNoEnd (unsigned long *DeviceIndex,* unsigned long *bWait*)

unsigned long DACOutputSetCount (unsigned long *DeviceIndex,* unsigned long *NewCount*)

unsigned long DACOutputFrame (unsigned long *DeviceIndex,* unsigned long *FramePoints,* unsigned short ∗ *FrameData*)

unsigned long DACOutputFrameRaw (unsigned long *DeviceIndex,* unsigned long *FramePoints,* unsigned short ∗ *FrameData*)

unsigned long DACOutputStart (unsigned long *DeviceIndex*)

unsigned long DACOutputSetInterlock (unsigned long *DeviceIndex,* unsigned long *bInterlock*)

24.118 lib/AIOUSB_DAC.h File Reference

```
#include "AIOTypes.h"
```

Functions

- unsigned long DACDirect (unsigned long DeviceIndex, unsigned short Channel, unsigned short Value)
- unsigned long DACMultiDirect (unsigned long DeviceIndex, unsigned short ∗pDACData, unsigned long DACDataCount)

  pDACData is an array of DACDataCount channel/count 16-bit word pairs:
- unsigned long DACSetBoardRange (unsigned long DeviceIndex, unsigned long RangeCode)
- unsigned long DACOutputOpen (unsigned long DeviceIndex, double ∗pClockHz)
- unsigned long DACOutputClose (unsigned long DeviceIndex, unsigned long bWait)
- unsigned long DACOutputCloseNoEnd (unsigned long DeviceIndex, unsigned long bWait)
- unsigned long DACOutputSetCount (unsigned long DeviceIndex, unsigned long NewCount)
- unsigned long DACOutputFrame (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short ∗FrameData)
- unsigned long DACOutputFrameRaw (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short ∗FrameData)
- unsigned long DACOutputStart (unsigned long DeviceIndex)
- unsigned long DACOutputSetInterlock (unsigned long DeviceIndex, unsigned long bInterlock)

24.118.1 Function Documentation

unsigned long DACDirect (unsigned long *DeviceIndex,* unsigned short *Channel,* unsigned short *Value*)

unsigned long DACMultiDirect (unsigned long *DeviceIndex,* unsigned short ∗ *pDACData,* unsigned long *DACDataCount*)

pDACData is an array of DACDataCount channel/count 16-bit word pairs:

```
*    +----------------+
*    |    channel     | word 0
*    |----------------|
*    |     count      | word 1
*    |----------------|
*           ...
*    |----------------|
*    |    channel     |
*    |----------------|
*    |     count      | word ( DACDataCount * 2 ) - 1
*    +----------------+
*
* this array has to be converted to a different format when passed to the board:
*          Block 0
*    +----------------+
*    |   chan mask    | byte 0
*    |----------------|
*    |  chan 0 count  | bytes 1-2
*   |----------------|
*          ...
*   |----------------|
*   |  chan 6 count  | bytes 13-14
*   |----------------|
*   |  chan 7 count  | bytes 15-16
*   +----------------+
*        Block 1
*   +----------------+
*   |   chan mask    | byte 17
*   |----------------|
*   |  chan 0 count  | bytes 18-19
*   |----------------|
*          ...
*   |----------------|
*   |  chan 6 count  | bytes 30-31
*   |----------------|
*   |  chan 7 count  | bytes 32-33
*   +----------------+
*          ...
*        Block n
*   +----------------+
*   |   chan mask    |
*   |----------------|
*   |  chan 0 count  |
*   |----------------|
*          ...
*   |----------------|
*   |  chan 6 count  |
*   |----------------|
*   |  chan 7 count  | bytes ( ( 17 * n ) - 2 ) - ( ( 17 * n ) - 1 )
*   +----------------+
*
```

the channel mask (the first byte of each block) has a bit set to one for each channel whose output is to be set; the count values are zero for channels that aren't to be set; for example, a mask of 0x01 would write to only channel 0 on a given block; a mask of 0x80 would write to only channel 7

since the DAC configuration blocks are contiguous, the byte offset to a channel's count within the buffer containing all the configuration blocks can be calculated as: offset = (channel ∗ sizeof(unsigned short)) + (channel / 8) + 1; although this calculation is correct, it's difficult to follow, so the code below uses a slightly less efficient calculation that's easier to understand

when sending the DAC configuration blocks to the device we have to send all the blocks from block 0 up to the block containing the highest channel number being set determine highest channel number addressed in pDACData; no checking is performed to ensure that the same channel is not set more than once

unsigned long DACSetBoardRange (unsigned long *DeviceIndex*, unsigned long *RangeCode*)

unsigned long DACOutputOpen (unsigned long *DeviceIndex*, double ∗ *pClockHz*)

unsigned long DACOutputClose (unsigned long *DeviceIndex*, unsigned long *bWait*)

unsigned long DACOutputCloseNoEnd (unsigned long *DeviceIndex*, unsigned long *bWait*)

unsigned long DACOutputSetCount (unsigned long *DeviceIndex*, unsigned long *NewCount*)

unsigned long DACOutputFrame (unsigned long *DeviceIndex*, unsigned long *FramePoints*, unsigned short ∗ *FrameData*)

unsigned long DACOutputFrameRaw (unsigned long *DeviceIndex*, unsigned long *FramePoints*, unsigned short ∗ *FrameData*)

unsigned long DACOutputStart (unsigned long *DeviceIndex*)

unsigned long DACOutputSetInterlock (unsigned long *DeviceIndex*, unsigned long *bInterlock*)

24.119 lib/AIOUSB_DIO.c File Reference

Core code to interface with Digital cards.

```
#include "AIOUSB_DIO.h"
#include "AIODeviceTable.h"
#include "AIOUSB_Core.h"
#include "USBDevice.h"
#include <arpa/inet.h>
```

Macros

- #define GET_ENDPOINT(isread) (isread ? (LIBUSB_ENDPOINT_IN | USB_BULK_READ_ENDPOINT) : (LIBUSB_ENDPOINT_OUT | USB_BULK_WRITE_ENDPOINT))

Functions

- int MASK_BYTES_SIZE (AIOUSBDevice ∗device)
- int TRISTATE_BYTES_SIZE (AIOUSBDevice ∗device)
- unsigned short aiousb_htons (unsigned short octaveOffset)

  Returns the number in Big-Endian format.
- AIORESULT DIO_ConfigureWithDIOBuf (unsigned long DeviceIndex, unsigned char bTristate, AIOChannelMask ∗mask, DIOBuf ∗buf)
- AIORESULT DIO_Configure (unsigned long DeviceIndex, unsigned char bTristate, void ∗pOutMask, void ∗pData)
- AIORESULT DIO_ConfigureEx (unsigned long DeviceIndex, void ∗pOutMask, void ∗pData, void ∗pTristateMask)
- AIORESULT DIO_ConfigurationQuery (unsigned long DeviceIndex, void ∗pOutMask, void ∗pTristateMask)
- AIORESULT DIO_WriteAll (unsigned long DeviceIndex, void ∗pData)
- AIORESULT DIO_Write8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char Data)
- AIORESULT DIO_Write1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char bData)
- AIORESULT DIO_ReadAll (unsigned long DeviceIndex, void ∗buf)
- AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf)
- AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf)
- AIORESULT DIO_ReadAllToCharStr (unsigned long DeviceIndex, char ∗buf, unsigned size)
- AIORESULT DIO_Read8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char ∗pdat)
- AIORESULT DIO_Read1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char ∗bit)
- AIORESULT DIO_StreamOpen (unsigned long DeviceIndex, unsigned long bIsRead)
- AIORESULT DIO_StreamClose (unsigned long DeviceIndex)
- AIORESULT DIO_StreamSetClocks (unsigned long DeviceIndex, double ∗ReadClockHz, double ∗WriteClockHz)
- int pow_of_minsize (int val)
- AIORESULT DIO_StreamFrame (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short ∗pFrameData, unsigned long ∗BytesTransferred)

24.119.1 Detailed Description

Core code to interface with Digital cards.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.119.2 Macro Definition Documentation

#define GET_ENDPOINT(*isread*) (isread ? (LIBUSB_ENDPOINT_IN | USB_BULK_READ_ENDPOINT) : (LIBUSB_ENDPOINT_OUT | USB_BULK_WRITE_ENDPOINT))

24.119.3 Function Documentation

int MASK_BYTES_SIZE (AIOUSBDevice ∗ *device*)

int TRISTATE_BYTES_SIZE (AIOUSBDevice ∗ *device*)

unsigned short aiousb_htons (unsigned short *octaveOffset*)

Returns the number in Big-Endian format.

AIORESULT DIO_ConfigureWithDIOBuf (unsigned long *DeviceIndex,* unsigned char *bTristate,* AIOChannelMask ∗ *mask,* DIOBuf ∗ *buf*)

AIORESULT DIO_Configure (unsigned long *DeviceIndex,* unsigned char *bTristate,* void ∗ *pOutMask,* void ∗ *pData*)

AIORESULT DIO_ConfigureEx (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pData,* void ∗ *pTristateMask*)

AIORESULT DIO_ConfigurationQuery (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pTristateMask*)

AIORESULT DIO_WriteAll (unsigned long *DeviceIndex,* void ∗ *pData*)

AIORESULT DIO_Write8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char *Data*)

AIORESULT DIO_Write1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char *bData*)

AIORESULT DIO_ReadAll (unsigned long *DeviceIndex,* void ∗ *buf*)

AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long *DeviceIndex,* DIOBuf ∗ *buf*)

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long *DeviceIndex,* DIOBuf ∗ *buf*)

AIORESULT DIO_ReadAllToCharStr (unsigned long *DeviceIndex,* char ∗ *buf,* unsigned *size*)

AIORESULT DIO_Read8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char ∗ *pdat*)

AIORESULT DIO_Read1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char ∗ *bit*)

AIORESULT DIO_StreamOpen (unsigned long *DeviceIndex,* unsigned long *bIsRead*)

AIORESULT DIO_StreamClose (unsigned long *DeviceIndex*)

AIORESULT DIO_StreamSetClocks (unsigned long *DeviceIndex,* double ∗ *ReadClockHz,* double ∗ *WriteClockHz*)

Note

```
* fill in data for the vendor request
* byte 0 used enable/disable read and write clocks
*   bit 0 is write clock
*   bit 1 is read  clock
*     1 = off/disable
*     0 = enable (1000 Khz is default value whenever enabled)
* bytes 1-2 = write clock value
* bytes 3-4 = read clock value
*
```

int pow_of_minsize (int *val*)

AIORESULT DIO_StreamFrame (unsigned long *DeviceIndex,* unsigned long *FramePoints,* unsigned short ∗ *pFrameData,* unsigned long ∗ *BytesTransferred*)

Note

convert parameter types to those that libusb likes

24.120 lib/AIOUSB_DIO.h File Reference

```
#include "AIOUSB_Core.h"
#include "DIOBuf.h"
#include "AIOChannelMask.h"
#include <assert.h>
#include <math.h>
#include <string.h>
```

Functions

- AIORESULT DIO_ConfigureWithDIOBuf (unsigned long DeviceIndex, unsigned char bTristate, AIOChannelMask ∗mask, DIOBuf ∗buf)
- unsigned long DIO_Configure (unsigned long DeviceIndex, unsigned char bTristate, void ∗pOutMask, void ∗p-Data)
- unsigned long DIO_ConfigureEx (unsigned long DeviceIndex, void ∗pOutMask, void ∗pData, void ∗pTristateMask)
- unsigned long DIO_ConfigurationQuery (unsigned long DeviceIndex, void ∗pOutMask, void ∗pTristateMask)
- unsigned long DIO_WriteAll (unsigned long DeviceIndex, void ∗pData)
- unsigned long DIO_Write8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char Data)
- unsigned long DIO_Write1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char bData)
- AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf)
- AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long DeviceIndex, DIOBuf ∗buf) ACCES_DEPRECATE-D("Please use DIO_ReadAllToDIOBuf")
- AIORESULT DIO_ReadAll (unsigned long DeviceIndex, void ∗buf)
- unsigned long DIO_ReadAllToCharStr (unsigned long DeviceIndex, char ∗buf, unsigned size)
- unsigned long DIO_Read8 (unsigned long DeviceIndex, unsigned long ByteIndex, unsigned char ∗pdat)
- unsigned long DIO_Read1 (unsigned long DeviceIndex, unsigned long BitIndex, unsigned char ∗bit)
- unsigned long DIO_StreamOpen (unsigned long DeviceIndex, unsigned long bIsRead)
- unsigned long DIO_StreamClose (unsigned long DeviceIndex)
- unsigned long DIO_StreamSetClocks (unsigned long DeviceIndex, double ∗ReadClockHz, double ∗WriteClock-Hz)
- unsigned long DIO_StreamFrame (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short ∗p-FrameData, unsigned long ∗BytesTransferred)

24.120.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.120.2 Function Documentation

AIORESULT DIO_ConfigureWithDIOBuf (unsigned long *DeviceIndex,* unsigned char *bTristate,* AIOChannelMask ∗ *mask,* DIOBuf ∗ *buf*)

unsigned long DIO_Configure (unsigned long *DeviceIndex,* unsigned char *bTristate,* void ∗ *pOutMask,* void ∗ *pData*)

unsigned long DIO_ConfigureEx (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pData,* void ∗ *pTristateMask*)

unsigned long DIO_ConfigurationQuery (unsigned long *DeviceIndex,* void ∗ *pOutMask,* void ∗ *pTristateMask*)

unsigned long DIO_WriteAll (unsigned long *DeviceIndex,* void ∗ *pData*)

unsigned long DIO_Write8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char *Data*)

unsigned long DIO_Write1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char *bData*)

AIORET_TYPE DIO_ReadAllToDIOBuf (unsigned long *DeviceIndex,* DIOBuf ∗ *buf*)

AIORET_TYPE DIO_ReadIntoDIOBuf (unsigned long *DeviceIndex,* DIOBuf ∗ *buf*)

Deprecated You should use the function DIO_ReadAllToDIOBuf instead

Parameters

DeviceIndex	
buf	

Returns

AIORESULT DIO_ReadAll (unsigned long *DeviceIndex,* void ∗ *buf*)

unsigned long DIO_ReadAllToCharStr (unsigned long *DeviceIndex,* char ∗ *buf,* unsigned *size*)

unsigned long DIO_Read8 (unsigned long *DeviceIndex,* unsigned long *ByteIndex,* unsigned char ∗ *pdat*)

unsigned long DIO_Read1 (unsigned long *DeviceIndex,* unsigned long *BitIndex,* unsigned char ∗ *bit*)

unsigned long DIO_StreamOpen (unsigned long *DeviceIndex,* unsigned long *bIsRead*)

unsigned long DIO_StreamClose (unsigned long *DeviceIndex*)

unsigned long DIO_StreamSetClocks (unsigned long *DeviceIndex,* double ∗ *ReadClockHz,* double ∗ *WriteClockHz*)

Note

```
* fill in data for the vendor request
* byte 0 used enable/disable read and write clocks
*   bit 0 is write clock
*   bit 1 is read  clock
*     1 = off/disable
*     0 = enable (1000 Khz is default value whenever enabled)
* bytes 1-2 = write clock value
* bytes 3-4 = read clock value
*
```

unsigned long DIO_StreamFrame (unsigned long *DeviceIndex,* unsigned long *FramePoints,* unsigned short ∗ *pFrameData,* unsigned long ∗ *BytesTransferred*)

Note

convert parameter types to those that libusb likes

24.121 lib/AIOUSB_Log.c File Reference

```
#include "AIOUSB_Log.h"
#include "AIOTypes.h"
```

Variables

- pthread_t cont_thread
- pthread_mutex_t message_lock = PTHREAD_MUTEX_INITIALIZER
- FILE ∗ outfile = NULL
- AIO_DEBUG_LEVEL AIOUSB_DEBUG_LEVEL = (AIO_DEBUG_LEVEL)7

24.121.1 Variable Documentation

pthread_t cont_thread

pthread_mutex_t message_lock = PTHREAD_MUTEX_INITIALIZER

FILE∗ outfile = NULL

AIO_DEBUG_LEVEL AIOUSB_DEBUG_LEVEL = (AIO_DEBUG_LEVEL)7

24.122 lib/AIOUSB_Log.h File Reference

```
#include <pthread.h>
#include <stdio.h>
#include "AIOTypes.h"
```

Macros

- #define GREEN "\033[0;32m"
- #define RED "\033[0;31m"
- #define MAGENTA "\033[0;35m"
- #define CYAN "\033[0;36m"
- #define AIO_DEVEL_STR GREEN"<Devel>"
- #define AIO_DEBUG_STR GREEN"<Debug>"
- #define AIO_WARN_STR CYAN"<Warn>"
- #define AIO_INFO_STR "<Info>"
- #define AIO_ERROR_STR RED"<Error>"
- #define AIO_FATAL_STR MAGENTA"<Fatal>"
- #define AIO_RESET_STR "\033[0m"
- #define AIOUSB_LOG(fmt,...)
- #define AIOUSB_DEVEL(...) if (0) { }
- #define AIOUSB_DEBUG(...) if (0) { }
- #define AIOUSB_WARN(...) if (0) { AIOUSB_LOG("<Warn>\t" __VA_ARGS__) }
- #define AIOUSB_INFO(...) if (0) { AIOUSB_LOG("<Info>\t" __VA_ARGS__); }
- #define AIOUSB_ERROR(...) AIOUSB_LOG("<Error>\t" __VA_ARGS__)
- #define AIOUSB_FATAL(...) AIOUSB_LOG("<Fatal>\t" __VA_ARGS__)

Enumerations

- enum AIO_DEBUG_LEVEL {
  AIOFATAL_LEVEL = 1, AIOERROR_LEVEL = 1, AIOINFO_LEVEL = 2, AIOWARN_LEVEL = 4,
  AIODEFAULT_LOG_LEVEL = 7, AIODEBUG_LEVEL = 8, AIODEVEL_LEVEL = 16 }

Variables

- AIO_DEBUG_LEVEL AIOUSB_DEBUG_LEVEL
- int LOG_LEVEL

  Compile with -DAIOUSB_DISABLE_LOG_MESSAGES if you don't wish to see these warning messages.
- pthread_t cont_thread
- pthread_mutex_t message_lock
- FILE ∗ outfile

24.122.1 Macro Definition Documentation

#define GREEN "\033[0;32m"

#define RED "\033[0;31m"

#define MAGENTA "\033[0;35m"

#define CYAN "\033[0;36m"

#define AIO_DEVEL_STR GREEN"<Devel>"

#define AIO_DEBUG_STR GREEN"<Debug>"

#define AIO_WARN_STR CYAN"<Warn>"

#define AIO_INFO_STR "<Info>"

#define AIO_ERROR_STR RED"<Error>"

#define AIO_FATAL_STR MAGENTA"<Fatal>"

#define AIO_RESET_STR "\033[0m"

#define AIOUSB_LOG(*fmt,* ...)

Value:

```
do {                                  \
        pthread_mutex_lock( &message_lock );                           \
    fprintf( (!outfile ? stdout : outfile ), fmt AIO_RESET_STR ,  ##__VA_ARGS__
      ); \
    pthread_mutex_unlock(&message_lock);                               \
  } while ( 0 )
```

#define AIOUSB_DEVEL(...) if (0) { }

#define AIOUSB_DEBUG(...) if (0) { }

#define AIOUSB_WARN(...) if (0) { AIOUSB_LOG("<Warn>\t" __VA_ARGS__) }

#define AIOUSB_INFO(...) if (0) { AIOUSB_LOG("<Info>\t" __VA_ARGS__); }

#define AIOUSB_ERROR(...) AIOUSB_LOG("<Error>\t" __VA_ARGS__)

#define AIOUSB_FATAL(...) AIOUSB_LOG("<Fatal>\t" __VA_ARGS__)

24.122.2 Enumeration Type Documentation

enum AIO_DEBUG_LEVEL

Enumerator

- ***AIOFATAL_LEVEL***
- ***AIOERROR_LEVEL***
- ***AIOINFO_LEVEL***
- ***AIOWARN_LEVEL***
- ***AIODEFAULT_LOG_LEVEL***
- ***AIODEBUG_LEVEL***
- ***AIODEVEL_LEVEL***

24.122.3 Variable Documentation

AIO_DEBUG_LEVEL AIOUSB_DEBUG_LEVEL

int LOG_LEVEL

Compile with -DAIOUSB_DISABLE_LOG_MESSAGES if you don't wish to see these warning messages.

pthread_t cont_thread

pthread_mutex_t message_lock

FILE∗ outfile

24.123 lib/AIOUSB_Properties.c File Reference

ACCES I/O USB Property utilities for Linux. These functions assist with identifying cards and verifying the devices attached are the correct type of card.

```
#include "AIOUSB_Properties.h"
#include "AIODeviceTable.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "AIOList.h"
```

Data Structures

- struct **ResultCodeName**

  AIOUSB result codes.

Macros

- #define RESULT_TEXT_SIZE 40
- #define NUM_RESULT_CODES (sizeof(resultCodeTable) / sizeof(resultCodeTable[0]))

Functions

- int non_usb_supported_device (int minProductID, int maxProductID, int maxDevices, int ∗deviceList)
- unsigned long AIOUSB_GetDeviceByProductID (int minProductID, int maxProductID, int maxDevices, int ∗device-List)
- AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long DeviceIndex)
- AIORESULT GetDeviceSerialNumber (unsigned long DeviceIndex, uint64_t ∗pSerialNumber)
- unsigned long GetDeviceBySerialNumber (uint64_t serialNumber)
- AIORET_TYPE AIOUSB_FindDevices (int ∗∗where, int ∗length, AIOUSB_BOOL(∗is_ok_device)(AIOUSBDevice ∗dev))

  Friendly function that can be called first.
- AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗indices, AIOProductGroup ∗pg)
- AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗where, int ∗length, AIOProductGroup ∗pg)
- unsigned long AIOUSB_GetDeviceProperties (unsigned long DeviceIndex, DeviceProperties ∗properties)

  AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()
- const char ∗ AIOUSB_GetResultCodeAsString (unsigned long result_value)
- AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType display_type)
- AIORET_TYPE AIOUSB_ListDevices ()

24.123.1 Detailed Description

ACCES I/O USB Property utilities for Linux. These functions assist with identifying cards and verifying the devices attached are the correct type of card.

Author

$Author$

Date

$Date$

Copyright:

©

Todo Implement a friendly FindDevices() function as well as FindDeviceByCriteria() function to replace all of the standard looping while (deviceMask != 0)...

24.123.2 Macro Definition Documentation

#define RESULT_TEXT_SIZE 40

#define NUM_RESULT_CODES (sizeof(resultCodeTable) / sizeof(resultCodeTable[0]))

24.123.3 Function Documentation

int non_usb_supported_device (int *minProductID,* int *maxProductID,* int *maxDevices,* int ∗ *deviceList*)

unsigned long AIOUSB_GetDeviceByProductID (int *minProductID,* int *maxProductID,* int *maxDevices,* int ∗ *deviceList*)

Parameters

minProductID	
maxProductID	
maxDevices	
deviceList	[1 + maxDevices ∗ 2]

Returns

< deviceList[] contains device index-product ID pairs, one pair per device found

AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long *DeviceIndex*)

AIORESULT GetDeviceSerialNumber (unsigned long *DeviceIndex,* uint64_t ∗ *pSerialNumber*)

Parameters

DeviceIndex	
pSerialNumber	

Returns

0 if successful, otherwise

unsigned long GetDeviceBySerialNumber (uint64_t *serialNumber*)

else, even if we get an error requesting the serial number from this device, keep searching

AIORET_TYPE AIOUSB_FindDevices (int ∗∗ *where,* int ∗ *length,* AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *is_ok_device*)

Friendly function that can be called first.

It

Parameters

where	
length	
is_ok_device	

Returns

AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗ *indices,* AIOProductGroup ∗ *pg*)

AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗ *where,* int ∗ *length,* AIOProductGroup ∗ *pg*)

unsigned long AIOUSB_GetDeviceProperties (unsigned long *DeviceIndex,* DeviceProperties ∗ *properties*)

AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()

const char∗ AIOUSB_GetResultCodeAsString (unsigned long *result_value*)

build index of result codes

AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType *display_type*)

AIORET_TYPE AIOUSB_ListDevices ( )

24.124 lib/AIOUSB_Properties.h File Reference

```
#include "AIOUSB_Core.h"
#include "AIOTypes.h"
#include "AIOProductTypes.h"
#include "AIOList.h"
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
```

Enumerations

- enum AIODisplayType { BASIC = 0, TERSE = 1, JSON = 2, YAML = 3 }

Functions

- AIORESULT AIOUSB_GetDeviceByProductID (int minProductID, int maxProductID, int maxDevices, int ∗deviceList)
- AIORESULT GetDeviceBySerialNumber (uint64_t pSerialNumber)
- AIORESULT GetDeviceSerialNumber (unsigned long DeviceIndex, uint64_t ∗pSerialNumber)
- AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long DeviceIndex)
- AIORESULT AIOUSB_GetDeviceProperties (unsigned long DeviceIndex, DeviceProperties ∗properties)

  AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()
- const char ∗ AIOUSB_GetResultCodeAsString (unsigned long result_value)
- AIORET_TYPE AIOUSB_ListDevices ()
- AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType display_type)
- AIORET_TYPE AIOUSB_FindDevices (int ∗∗where, int ∗length, AIOUSB_BOOL(∗is_ok_device)(AIOUSBDevice ∗dev))

  Friendly function that can be called first.
- AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗where, int ∗length, AIOProductGroup ∗pg)
- AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗indices, AIOProductGroup ∗pg)

24.124.1 Detailed Description

Author

$Author$

Date

$Date$

Copyright:

©

24.124.2 Enumeration Type Documentation

enum AIODisplayType

Enumerator

BASIC

TERSE

JSON

YAML

24.124.3 Function Documentation

AIORESULT AIOUSB_GetDeviceByProductID (int *minProductID,* int *maxProductID,* int *maxDevices,* int ∗ *deviceList*)

Parameters

minProductID	
maxProductID	
maxDevices	
deviceList	[1 + maxDevices ∗ 2]

Returns

< deviceList[] contains device index-product ID pairs, one pair per device found

AIORESULT GetDeviceBySerialNumber (uint64_t *pSerialNumber*)

else, even if we get an error requesting the serial number from this device, keep searching

AIORESULT GetDeviceSerialNumber (unsigned long *DeviceIndex,* uint64_t ∗ *pSerialNumber*)

Parameters

DeviceIndex	
pSerialNumber	

Returns

0 if successful, otherwise

AIORET_TYPE AIOUSB_GetDeviceSerialNumber (unsigned long *DeviceIndex*)

AIORESULT AIOUSB_GetDeviceProperties (unsigned long *DeviceIndex,* DeviceProperties ∗ *properties*)

AIOUSB_GetDeviceProperties() returns a richer amount of information than QueryDeviceInfo()

const char∗ AIOUSB_GetResultCodeAsString (unsigned long *result_value*)

build index of result codes

AIORET_TYPE AIOUSB_ListDevices ( )

AIORET_TYPE AIOUSB_ShowDevices (AIODisplayType *display_type*)

AIORET_TYPE AIOUSB_FindDevices (int ∗∗ *where,* int ∗ *length,* AIOUSB_BOOL(∗)(AIOUSBDevice ∗dev) *is_ok_device*)

Friendly function that can be called first.

It

Parameters

where	
length	
is_ok_device	

Returns

AIORET_TYPE AIOUSB_FindDevicesByGroup (int ∗∗ *where,* int ∗ *length,* AIOProductGroup ∗ *pg*)

AIORET_TYPE AIOUSB_FindDeviceIndicesByGroup (intlist ∗ *indices,* AIOProductGroup ∗ *pg*)

24.125 lib/AIOUSB_USB.c File Reference

```
#include "AIOUSB_USB.h"
```

24.126 lib/AIOUSB_USB.h File Reference

Macros

- #define usb_control_xfer libusb_control_transfer
- #define usb_bulk_xfer libusb_bulk_transfer
- #define usb_open libusb_open
- #define usb_close libusb_close
- #define usb_free_devices libusb_free_device_list
- #define usb_get_devices libusb_get_device_list

24.126.1 Macro Definition Documentation

#define usb_control_xfer libusb_control_transfer

#define usb_bulk_xfer libusb_bulk_transfer

#define usb_open libusb_open

#define usb_close libusb_close

#define usb_free_devices libusb_free_device_list

#define usb_get_devices libusb_get_device_list

24.127 lib/AIOUSB_WDG.c File Reference

```
#include "AIOUSB_WDG.h"
#include "AIOUSB_Core.h"
#include <stdio.h>
```

Functions

- AIOWDGConfig ∗ NewWDGConfig (void)

  Creates a new Watchdog configuration object that can be used to trigger watchdog petting / resets.
- void doSomething ()
- void DeleteWDGConfig (AIOWDGConfig ∗obj)

  Deletes the AIOWDGConfig object.
- AIORET_TYPE WDG_SetConfig (unsigned long DeviceIndex, AIOWDGConfig ∗obj)

  Assigns the watchdog object to the device index in question.
- AIORET_TYPE WDG_GetStatus (unsigned long DeviceIndex, AIOWDGConfig ∗obj)
- AIORET_TYPE WDG_Pet (unsigned long DeviceIndex, AIOWDGConfig ∗obj)

  Pets the watchdog and keeps it from resetting the device.

24.127.1 Function Documentation

AIOWDGConfig∗ NewWDGConfig (void)

Creates a new Watchdog configuration object that can be used to trigger watchdog petting / resets.

Returns

AIOWDGConfig ∗obj New Watchdog configuration object

void doSomething ( )

void DeleteWDGConfig (AIOWDGConfig ∗ *obj*)

Deletes the AIOWDGConfig object.

Parameters

obj	

AIORET_TYPE WDG_SetConfig (unsigned long *DeviceIndex,* AIOWDGConfig ∗ *obj*)

Assigns the watchdog object to the device index in question.

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_GetStatus (unsigned long *DeviceIndex,* AIOWDGConfig ∗ *obj*)

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_Pet (unsigned long *DeviceIndex,* AIOWDGConfig ∗ *obj*)

Pets the watchdog and keeps it from resetting the device.

Parameters

DeviceIndex	
obj	

Returns

>= 0 if successful, < 0 otherwise

24.128 lib/AIOUSB_WDG.h File Reference

```
#include "aiousb.h"
```

Data Structures

- struct AIOWDGConfig

Enumerations

- enum WDGVals { WDGVals_begin = (0x00 -1), AIOUSB_WDG_READ_VALUE, AIOUSB_WDG_READ_INDEX = 0x0041, WDGVals_end }

Functions

- AIOWDGConfig ∗ NewWDGConfig (void)

  Creates a new Watchdog configuration object that can be used to trigger watchdog petting / resets.
- void DeleteWDGConfig (AIOWDGConfig ∗obj)

  Deletes the AIOWDGConfig object.
- AIORET_TYPE WDG_SetConfig (unsigned long DeviceIndex, AIOWDGConfig ∗obj)

  Assigns the watchdog object to the device index in question.
- AIORET_TYPE WDG_GetStatus (unsigned long DeviceIndex, AIOWDGConfig ∗obj)
- AIORET_TYPE WDG_Pet (unsigned long DeviceIndex, AIOWDGConfig ∗obj)

  Pets the watchdog and keeps it from resetting the device.

24.128.1 Enumeration Type Documentation

enum WDGVals

Enumerator

WDGVals_begin

AIOUSB_WDG_READ_VALUE

AIOUSB_WDG_READ_INDEX

WDGVals_end

24.128.2 Function Documentation

AIOWDGConfig∗ NewWDGConfig (void)

Creates a new Watchdog configuration object that can be used to trigger watchdog petting / resets.

Returns

AIOWDGConfig ∗obj New Watchdog configuration object

void DeleteWDGConfig (AIOWDGConfig ∗ *obj*)

Deletes the AIOWDGConfig object.

Parameters

obj	

AIORET_TYPE WDG_SetConfig (unsigned long *DeviceIndex*, AIOWDGConfig ∗ *obj*)

Assigns the watchdog object to the device index in question.

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_GetStatus (unsigned long *DeviceIndex*, AIOWDGConfig ∗ *obj*)

Parameters

DeviceIndex	
obj	

Returns

AIORET_TYPE WDG_Pet (unsigned long *DeviceIndex*, AIOWDGConfig ∗ *obj*)

Pets the watchdog and keeps it from resetting the device.

Parameters

DeviceIndex	
obj	

Returns

$>=$ 0 if successful, $<$ 0 otherwise

24.129 lib/AIOUSBDevice.c File Reference

```
#include "AIOUSBDevice.h"
#include "AIODeviceTable.h"
#include "AIOUSB_ADC.h"
#include "AIOUSB_Core.h"
```

Functions

- AIOUSBDevice ∗ NewAIOUSBDevice (unsigned long DeviceIndex)
- AIOUSBDevice ∗ NewAIOUSBDeviceFromJSON (char ∗str)
- char ∗ AIOUSBDeviceToJSON (AIOUSBDevice ∗device)
- void DeleteAIOUSBDevice (AIOUSBDevice ∗dev)
- AIORET_TYPE AIOUSBDeviceInitializeWithProductID (AIOUSBDevice ∗device, ProductIDS productID)
- AIORET_TYPE AIOUSBDeviceSetTimeout (AIOUSBDevice ∗device, unsigned timeout)
- AIORET_TYPE AIOUSBDeviceGetTimeout (AIOUSBDevice ∗device)
- AIORET_TYPE AIOUSBDeviceCopyADCConfigBlock (AIOUSBDevice ∗dev, ADCConfigBlock ∗newone)
- AIORET_TYPE AIOUSBDeviceSetADCConfigBlock (AIOUSBDevice ∗dev, ADCConfigBlock ∗conf)
- AIORET_TYPE AIOUSBDeviceSize ()
- ADCConfigBlock ∗ AIOUSBDeviceGetADCConfigBlock (AIOUSBDevice ∗dev)
- AIORET_TYPE AIOUSBDeviceWriteADCConfig (AIOUSBDevice ∗device, ADCConfigBlock ∗config)
- AIORET_TYPE AIOUSBDeviceSetTesting (AIOUSBDevice ∗dev, AIOUSB_BOOL testing)
- AIORET_TYPE AIOUSBDeviceGetStreamingBlockSize (AIOUSBDevice ∗dev)
- AIORET_TYPE AIOUSBDeviceGetTesting (AIOUSBDevice ∗dev)
- USBDevice ∗ AIOUSBDeviceGetUSBHandle (AIOUSBDevice ∗dev)
- AIORET_TYPE AIOUSBDeviceSetUSBHandle (AIOUSBDevice ∗dev, USBDevice ∗usb)
- USBDevice ∗ AIOUSBDeviceGetUSBHandleFromDeviceIndex (unsigned long DeviceIndex, AIOUSBDevice ∗∗dev, AIORESULT ∗result)
- AIORET_TYPE AIOUSBDeviceGetDiscardFirstSample (AIOUSBDevice ∗device)
- AIORET_TYPE AIOUSBDeviceSetDiscardFirstSample (AIOUSBDevice ∗device, AIOUSB_BOOL discard)

24.129.1 Function Documentation

AIOUSBDevice∗ NewAIOUSBDevice (unsigned long *DeviceIndex*)

AIOUSBDevice ∗ NewAIOUSBDeviceFromJSON (char ∗ *str*)

char∗ AIOUSBDeviceToJSON (AIOUSBDevice ∗ *device*)

void DeleteAIOUSBDevice (AIOUSBDevice ∗ *dev*)

AIORET_TYPE AIOUSBDeviceInitializeWithProductID (AIOUSBDevice ∗ *device*, ProductIDS *productID*)

AIORET_TYPE AIOUSBDeviceSetTimeout (AIOUSBDevice ∗ *device*, unsigned *timeout*)

AIORET_TYPE AIOUSBDeviceGetTimeout (AIOUSBDevice ∗ *device*)

AIORET_TYPE AIOUSBDeviceCopyADCConfigBlock (AIOUSBDevice ∗ *dev*, ADCConfigBlock ∗ *newone*)

AIORET_TYPE AIOUSBDeviceSetADCConfigBlock (AIOUSBDevice ∗ *dev*, ADCConfigBlock ∗ *conf*)

AIORET_TYPE AIOUSBDeviceSize ( )

ADCConfigBlock∗ AIOUSBDeviceGetADCConfigBlock (AIOUSBDevice ∗ *dev*)

AIORET_TYPE AIOUSBDeviceWriteADCConfig (AIOUSBDevice ∗ *device*, ADCConfigBlock ∗ *config*)

AIORET_TYPE AIOUSBDeviceSetTesting (AIOUSBDevice ∗ *dev*, AIOUSB_BOOL *testing*)

AIORET_TYPE AIOUSBDeviceGetStreamingBlockSize (AIOUSBDevice ∗ *dev*)

AIORET_TYPE AIOUSBDeviceGetTesting (AIOUSBDevice ∗ *dev*)

USBDevice∗ AIOUSBDeviceGetUSBHandle (AIOUSBDevice ∗ *dev*)

AIORET_TYPE AIOUSBDeviceSetUSBHandle (AIOUSBDevice ∗ *dev*, USBDevice ∗ *usb*)

USBDevice∗ AIOUSBDeviceGetUSBHandleFromDeviceIndex (unsigned long *DeviceIndex,* AIOUSBDevice ∗∗ *dev,* AIORESULT ∗ *result*)

AIORET_TYPE AIOUSBDeviceGetDiscardFirstSample (AIOUSBDevice ∗ *device*)

AIORET_TYPE AIOUSBDeviceSetDiscardFirstSample (AIOUSBDevice ∗ *device,* AIOUSB_BOOL *discard*)

24.130 lib/AIOUSBDevice.h File Reference

```
#include "AIOTypes.h"
#include "ADCConfigBlock.h"
#include "USBDevice.h"
#include "cJSON.h"
#include <string.h>
#include <semaphore.h>
#include <libusb.h>
#include <pthread.h>
```

Data Structures

- struct AIOUSBDevice

Typedefs

- typedef AIOUSBDevice DeviceDescriptor

Functions

- char ∗ AIOUSBDeviceToJSON (AIOUSBDevice ∗device)
- AIOUSBDevice ∗ NewAIOUSBDeviceFromJSON (char ∗str)
- AIORET_TYPE AIOUSBDeviceInitializeWithProductID (AIOUSBDevice ∗device, ProductIDS productID)
- USBDevice ∗ AIOUSBDeviceGetUSBHandle (AIOUSBDevice ∗dev)
- USBDevice ∗ AIOUSBDeviceGetUSBHandleFromDeviceIndex (unsigned long DeviceIndex, AIOUSBDevice ∗∗dev, AIORESULT ∗res)
- AIORET_TYPE AIOUSBDeviceSetUSBHandle (AIOUSBDevice ∗dev, USBDevice ∗usb)
- AIORET_TYPE AIOUSBDeviceSetADCConfigBlock (AIOUSBDevice ∗dev, ADCConfigBlock ∗conf)
- ADCConfigBlock ∗ AIOUSBDeviceGetADCConfigBlock (AIOUSBDevice ∗dev)
- AIORET_TYPE AIOUSBDeviceCopyADCConfigBlock (AIOUSBDevice ∗dev, ADCConfigBlock ∗newone)
- AIORET_TYPE AIOUSBDeviceSetTesting (AIOUSBDevice ∗dev, AIOUSB_BOOL testing)
- AIORET_TYPE AIOUSBDeviceSize ()
- AIORET_TYPE AIOUSBDeviceGetTesting (AIOUSBDevice ∗dev)
- AIORET_TYPE AIOUSBDeviceGetStreamingBlockSize (AIOUSBDevice ∗deviceDesc)
- AIORET_TYPE AIOUSBDeviceGetDiscardFirstSample (AIOUSBDevice ∗device)
- AIORET_TYPE AIOUSBDeviceSetDiscardFirstSample (AIOUSBDevice ∗device, AIOUSB_BOOL discard)
- AIORET_TYPE AIOUSBDeviceSetTimeout (AIOUSBDevice ∗device, unsigned timeout)
- AIORET_TYPE AIOUSBDeviceGetTimeout (AIOUSBDevice ∗device)
- AIORET_TYPE AIOUSBDeviceWriteADCConfig (AIOUSBDevice ∗device, ADCConfigBlock ∗config)

24.130.1 Typedef Documentation

typedef AIOUSBDevice DeviceDescriptor

24.130.2 Function Documentation

char∗ AIOUSBDeviceToJSON (AIOUSBDevice ∗ *device*)

AIOUSBDevice∗ NewAIOUSBDeviceFromJSON (char ∗ *str*)

AIORET_TYPE AIOUSBDeviceInitializeWithProductID (AIOUSBDevice ∗ *device,* ProductIDS *productID*)

USBDevice∗ AIOUSBDeviceGetUSBHandle (AIOUSBDevice ∗ *dev*)

USBDevice∗ AIOUSBDeviceGetUSBHandleFromDeviceIndex (unsigned long *DeviceIndex,* AIOUSBDevice ∗∗ *dev,* AIORESULT ∗ *res*)

AIORET_TYPE AIOUSBDeviceSetUSBHandle (AIOUSBDevice ∗ *dev,* USBDevice ∗ *usb*)

AIORET_TYPE AIOUSBDeviceSetADCConfigBlock (AIOUSBDevice ∗ *dev,* ADCConfigBlock ∗ *conf*)

ADCConfigBlock∗ AIOUSBDeviceGetADCConfigBlock (AIOUSBDevice ∗ *dev*)

AIORET_TYPE AIOUSBDeviceCopyADCConfigBlock (AIOUSBDevice ∗ *dev,* ADCConfigBlock ∗ *newone*)

AIORET_TYPE AIOUSBDeviceSetTesting (AIOUSBDevice ∗ *dev,* AIOUSB_BOOL *testing*)

AIORET_TYPE AIOUSBDeviceSize ( )

AIORET_TYPE AIOUSBDeviceGetTesting (AIOUSBDevice ∗ *dev*)

AIORET_TYPE AIOUSBDeviceGetStreamingBlockSize (AIOUSBDevice ∗ *deviceDesc*)

AIORET_TYPE AIOUSBDeviceGetDiscardFirstSample (AIOUSBDevice ∗ *device*)

AIORET_TYPE AIOUSBDeviceSetDiscardFirstSample (AIOUSBDevice ∗ *device,* AIOUSB_BOOL *discard*)

AIORET_TYPE AIOUSBDeviceSetTimeout (AIOUSBDevice ∗ *device,* unsigned *timeout*)

AIORET_TYPE AIOUSBDeviceGetTimeout (AIOUSBDevice ∗ *device*)

AIORET_TYPE AIOUSBDeviceWriteADCConfig (AIOUSBDevice ∗ *device,* ADCConfigBlock ∗ *config*)

24.131 lib/cJSON.c File Reference

```
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <float.h>
#include <limits.h>
#include <ctype.h>
#include "cJSON.h"
```

Functions

- const char ∗ cJSON_GetErrorPtr (void)
- void cJSON_InitHooks (cJSON_Hooks ∗hooks)
- void cJSON_Delete (cJSON ∗c)
- cJSON ∗ cJSON_ParseWithOpts (const char ∗value, const char ∗∗return_parse_end, int require_null_-terminated)
- cJSON ∗ cJSON_Parse (const char ∗value)
- char ∗ cJSON_Print (cJSON ∗item)
- char ∗ cJSON_PrintUnformatted (cJSON ∗item)
- int cJSON_AsInteger (cJSON ∗item)
- int cJSON_GetArraySize (cJSON ∗array)
- cJSON ∗ cJSON_GetArrayItem (cJSON ∗array, int item)
- cJSON ∗ cJSON_GetObjectItem (cJSON ∗object, const char ∗string)
- void cJSON_AddItemToArray (cJSON ∗array, cJSON ∗item)
- void cJSON_AddItemToObject (cJSON ∗object, const char ∗string, cJSON ∗item)
- void cJSON_AddItemReferenceToArray (cJSON ∗array, cJSON ∗item)
- void cJSON_AddItemReferenceToObject (cJSON ∗object, const char ∗string, cJSON ∗item)
- cJSON ∗ cJSON_DetachItemFromArray (cJSON ∗array, int which)
- void cJSON_DeleteItemFromArray (cJSON ∗array, int which)
- cJSON ∗ cJSON_DetachItemFromObject (cJSON ∗object, const char ∗string)
- void cJSON_DeleteItemFromObject (cJSON ∗object, const char ∗string)
- void cJSON_ReplaceItemInArray (cJSON ∗array, int which, cJSON ∗newitem)
- void cJSON_ReplaceItemInObject (cJSON ∗object, const char ∗string, cJSON ∗newitem)
- cJSON ∗ cJSON_CreateNull (void)
- cJSON ∗ cJSON_CreateTrue (void)
- cJSON ∗ cJSON_CreateFalse (void)
- cJSON ∗ cJSON_CreateBool (int b)
- cJSON ∗ cJSON_CreateNumber (double num)
- cJSON ∗ cJSON_CreateString (const char ∗string)

- cJSON ∗ cJSON_CreateArray (void)
- cJSON ∗ cJSON_CreateObject (void)
- cJSON ∗ cJSON_CreateIntArray (const int ∗numbers, int count)
- cJSON ∗ cJSON_CreateFloatArray (const float ∗numbers, int count)
- cJSON ∗ cJSON_CreateDoubleArray (const double ∗numbers, int count)
- cJSON ∗ cJSON_CreateStringArray (const char ∗∗strings, int count)
- cJSON ∗ cJSON_Duplicate (cJSON ∗item, int recurse)
- void cJSON_Minify (char ∗json)

24.131.1 Function Documentation

const char∗ cJSON_GetErrorPtr (void)

void cJSON_InitHooks (cJSON_Hooks ∗ *hooks*)

void cJSON_Delete (cJSON ∗ *c*)

cJSON∗ cJSON_ParseWithOpts (const char ∗ *value,* const char ∗∗ *return_parse_end,* int *require_null_terminated*)

cJSON∗ cJSON_Parse (const char ∗ *value*)

char∗ cJSON_Print (cJSON ∗ *item*)

char∗ cJSON_PrintUnformatted (cJSON ∗ *item*)

int cJSON_AsInteger (cJSON ∗ *item*)

int cJSON_GetArraySize (cJSON ∗ *array*)

cJSON∗ cJSON_GetArrayItem (cJSON ∗ *array,* int *item*)

cJSON∗ cJSON_GetObjectItem (cJSON ∗ *object,* const char ∗ *string*)

void cJSON_AddItemToArray (cJSON ∗ *array,* cJSON ∗ *item*)

void cJSON_AddItemToObject (cJSON ∗ *object,* const char ∗ *string,* cJSON ∗ *item*)

void cJSON_AddItemReferenceToArray (cJSON ∗ *array,* cJSON ∗ *item*)

void cJSON_AddItemReferenceToObject (cJSON ∗ *object,* const char ∗ *string,* cJSON ∗ *item*)

cJSON∗ cJSON_DetachItemFromArray (cJSON ∗ *array,* int *which*)

void cJSON_DeleteItemFromArray (cJSON ∗ *array,* int *which*)

cJSON∗ cJSON_DetachItemFromObject (cJSON ∗ *object,* const char ∗ *string*)

void cJSON_DeleteItemFromObject (cJSON ∗ *object,* const char ∗ *string*)

void cJSON_ReplaceItemInArray (cJSON ∗ *array,* int *which,* cJSON ∗ *newitem*)

void cJSON_ReplaceItemInObject (cJSON ∗ *object,* const char ∗ *string,* cJSON ∗ *newitem*)

cJSON∗ cJSON_CreateNull (void)

cJSON∗ cJSON_CreateTrue (void)

cJSON∗ cJSON_CreateFalse (void)

cJSON∗ cJSON_CreateBool (int *b*)

cJSON∗ cJSON_CreateNumber (double *num*)

cJSON∗ cJSON_CreateString (const char ∗ *string*)

cJSON∗ cJSON_CreateArray (void)

cJSON∗ cJSON_CreateObject (void)

cJSON∗ cJSON_CreateIntArray (const int ∗ *numbers,* int *count*)

cJSON∗ cJSON_CreateFloatArray (const float ∗ *numbers,* int *count*)

cJSON∗ cJSON_CreateDoubleArray (const double ∗ *numbers,* int *count*)

cJSON∗ cJSON_CreateStringArray (const char ∗∗ *strings,* int *count*)

cJSON∗ cJSON_Duplicate (cJSON ∗ *item,* int *recurse*)

void cJSON_Minify (char ∗ *json*)

24.132 lib/cJSON.h File Reference

Data Structures

- struct cJSON
- struct cJSON_Hooks

Macros

- #define cJSON_False 0
- #define cJSON_True 1
- #define cJSON_NULL 2
- #define cJSON_Number 3
- #define cJSON_String 4
- #define cJSON_Array 5
- #define cJSON_Object 6
- #define cJSON_IsReference 256
- #define cJSON_AddNullToObject(object, name) cJSON_AddItemToObject(object, name, cJSON_CreateNull())
- #define cJSON_AddTrueToObject(object, name) cJSON_AddItemToObject(object, name, cJSON_CreateTrue())
- #define cJSON_AddFalseToObject(object, name) cJSON_AddItemToObject(object, name, cJSON_CreateFalse())
- #define cJSON_AddBoolToObject(object, name, b) cJSON_AddItemToObject(object, name, cJSON_CreateBool(b))
- #define cJSON_AddNumberToObject(object, name, n) cJSON_AddItemToObject(object, name, cJSON_CreateNumber(n))
- #define cJSON_AddStringToObject(object, name, s) cJSON_AddItemToObject(object, name, cJSON_CreateString(s))
- #define cJSON_SetIntValue(object, val) ((object)?(object)->valueint=(object)->valuedouble=(val):(val))

Typedefs

- typedef struct cJSON cJSON
- typedef struct cJSON_Hooks cJSON_Hooks

Functions

- void cJSON_InitHooks (cJSON_Hooks ∗hooks)
- cJSON ∗ cJSON_Parse (const char ∗value)
- char ∗ cJSON_Print (cJSON ∗item)
- char ∗ cJSON_PrintUnformatted (cJSON ∗item)
- void cJSON_Delete (cJSON ∗c)
- int cJSON_AsInteger (cJSON ∗item)
- int cJSON_GetArraySize (cJSON ∗array)
- cJSON ∗ cJSON_GetArrayItem (cJSON ∗array, int item)
- cJSON ∗ cJSON_GetObjectItem (cJSON ∗object, const char ∗string)
- const char ∗ cJSON_GetErrorPtr (void)
- cJSON ∗ cJSON_CreateNull (void)
- cJSON ∗ cJSON_CreateTrue (void)
- cJSON ∗ cJSON_CreateFalse (void)
- cJSON ∗ cJSON_CreateBool (int b)
- cJSON ∗ cJSON_CreateNumber (double num)
- cJSON ∗ cJSON_CreateString (const char ∗string)
- cJSON ∗ cJSON_CreateArray (void)
- cJSON ∗ cJSON_CreateObject (void)

- cJSON ∗ cJSON_CreateIntArray (const int ∗numbers, int count)
- cJSON ∗ cJSON_CreateFloatArray (const float ∗numbers, int count)
- cJSON ∗ cJSON_CreateDoubleArray (const double ∗numbers, int count)
- cJSON ∗ cJSON_CreateStringArray (const char ∗∗strings, int count)
- void cJSON_AddItemToArray (cJSON ∗array, cJSON ∗item)
- void cJSON_AddItemToObject (cJSON ∗object, const char ∗string, cJSON ∗item)
- void cJSON_AddItemReferenceToArray (cJSON ∗array, cJSON ∗item)
- void cJSON_AddItemReferenceToObject (cJSON ∗object, const char ∗string, cJSON ∗item)
- cJSON ∗ cJSON_DetachItemFromArray (cJSON ∗array, int which)
- void cJSON_DeleteItemFromArray (cJSON ∗array, int which)
- cJSON ∗ cJSON_DetachItemFromObject (cJSON ∗object, const char ∗string)
- void cJSON_DeleteItemFromObject (cJSON ∗object, const char ∗string)
- void cJSON_ReplaceItemInArray (cJSON ∗array, int which, cJSON ∗newitem)
- void cJSON_ReplaceItemInObject (cJSON ∗object, const char ∗string, cJSON ∗newitem)
- cJSON ∗ cJSON_Duplicate (cJSON ∗item, int recurse)
- cJSON ∗ cJSON_ParseWithOpts (const char ∗value, const char ∗∗return_parse_end, int require_null_-terminated)
- void cJSON_Minify (char ∗json)

24.132.1 Macro Definition Documentation

#define cJSON_False 0

#define cJSON_True 1

#define cJSON_NULL 2

#define cJSON_Number 3

#define cJSON_String 4

#define cJSON_Array 5

#define cJSON_Object 6

#define cJSON_IsReference 256

#define cJSON_AddNullToObject(*object, name* **) cJSON_AddItemToObject(object, name, cJSON_CreateNull())**

#define cJSON_AddTrueToObject(*object, name* **) cJSON_AddItemToObject(object, name, cJSON_CreateTrue())**

#define cJSON_AddFalseToObject(*object, name* **) cJSON_AddItemToObject(object, name, cJSON_CreateFalse())**

#define cJSON_AddBoolToObject(*object, name, b* **) cJSON_AddItemToObject(object, name, cJSON_CreateBool(b))**

#define cJSON_AddNumberToObject(*object, name, n* **) cJSON_AddItemToObject(object, name, cJSON_CreateNumber(n))**

#define cJSON_AddStringToObject(*object, name, s* **) cJSON_AddItemToObject(object, name, cJSON_CreateString(s))**

#define cJSON_SetIntValue(*object, val* **) ((object)?(object)->valueint=(object)->valuedouble=(val):(val))**

24.132.2 Typedef Documentation

typedef struct cJSON cJSON

typedef struct cJSON_Hooks cJSON_Hooks

24.132.3 Function Documentation

void cJSON_InitHooks (**cJSON_Hooks** ∗ *hooks* **)**

cJSON∗ cJSON_Parse (const char ∗ *value* **)**

char∗ cJSON_Print (**cJSON** ∗ *item* **)**

char∗ cJSON_PrintUnformatted (**cJSON** ∗ *item* **)**

void cJSON_Delete (cJSON ∗ *c*)

int cJSON_AsInteger (cJSON ∗ *item*)

int cJSON_GetArraySize (cJSON ∗ *array*)

cJSON∗ cJSON_GetArrayItem (cJSON ∗ *array,* int *item*)

cJSON∗ cJSON_GetObjectItem (cJSON ∗ *object,* const char ∗ *string*)

const char∗ cJSON_GetErrorPtr (void)

cJSON∗ cJSON_CreateNull (void)

cJSON∗ cJSON_CreateTrue (void)

cJSON∗ cJSON_CreateFalse (void)

cJSON∗ cJSON_CreateBool (int *b*)

cJSON∗ cJSON_CreateNumber (double *num*)

cJSON∗ cJSON_CreateString (const char ∗ *string*)

cJSON∗ cJSON_CreateArray (void)

cJSON∗ cJSON_CreateObject (void)

cJSON∗ cJSON_CreateIntArray (const int ∗ *numbers,* int *count*)

cJSON∗ cJSON_CreateFloatArray (const float ∗ *numbers,* int *count*)

cJSON∗ cJSON_CreateDoubleArray (const double ∗ *numbers,* int *count*)

cJSON∗ cJSON_CreateStringArray (const char ∗∗ *strings,* int *count*)

void cJSON_AddItemToArray (cJSON ∗ *array,* cJSON ∗ *item*)

void cJSON_AddItemToObject (cJSON ∗ *object,* const char ∗ *string,* cJSON ∗ *item*)

void cJSON_AddItemReferenceToArray (cJSON ∗ *array,* cJSON ∗ *item*)

void cJSON_AddItemReferenceToObject (cJSON ∗ *object,* const char ∗ *string,* cJSON ∗ *item*)

cJSON∗ cJSON_DetachItemFromArray (cJSON ∗ *array,* int *which*)

void cJSON_DeleteItemFromArray (cJSON ∗ *array,* int *which*)

cJSON∗ cJSON_DetachItemFromObject (cJSON ∗ *object,* const char ∗ *string*)

void cJSON_DeleteItemFromObject (cJSON ∗ *object,* const char ∗ *string*)

void cJSON_ReplaceItemInArray (cJSON ∗ *array,* int *which,* cJSON ∗ *newitem*)

void cJSON_ReplaceItemInObject (cJSON ∗ *object,* const char ∗ *string,* cJSON ∗ *newitem*)

cJSON∗ cJSON_Duplicate (cJSON ∗ *item,* int *recurse*)

cJSON∗ cJSON_ParseWithOpts (const char ∗ *value,* const char ∗∗ *return_parse_end,* int *require_null_terminated*)

void cJSON_Minify (char ∗ *json*)

24.133 lib/CStringArray.c File Reference

```
#include "CStringArray.h"
#include "AIOTypes.h"
#include <stdarg.h>
#include <string.h>
#include <stddef.h>
#include <stdio.h>
```

Functions

- CStringArray ∗ NewCStringArrayWithStrings (size_t numstrings,...)
- CStringArray ∗ NewCStringArray (size_t numstrings)
- CStringArray ∗ NewCStringArrayFromCArgs (int argc, char ∗argv[])
- AIORET_TYPE DeleteCStringArray (CStringArray ∗str)
- CStringArray ∗ CopyCStringArray (CStringArray ∗str)
- char ∗ CStringArrayToString (CStringArray ∗str)
- char ∗ CStringArrayToStringWithDelimeter (CStringArray ∗str, const char ∗delim)

24.133.1 Function Documentation

CStringArray∗ NewCStringArrayWithStrings (size_t *numstrings,* *...*)

CStringArray∗ NewCStringArray (size_t *numstrings*)

CStringArray∗ NewCStringArrayFromCArgs (int *argc,* char ∗ *argv[ ]*)

AIORET_TYPE DeleteCStringArray (**CStringArray** ∗ *str*)

CStringArray∗ CopyCStringArray (**CStringArray** ∗ *str*)

char∗ CStringArrayToString (**CStringArray** ∗ *str*)

char∗ CStringArrayToStringWithDelimeter (**CStringArray** ∗ *str,* const char ∗ *delim*)

24.134 lib/CStringArray.h File Reference

```
#include "AIOTypes.h"
```

Data Structures

- struct CStringArray

Macros

- #define STRING_ARRAY(N,...)

Typedefs

- typedef struct CStringArray CStringArray

Functions

- CStringArray ∗ NewCStringArray (size_t numstrings)
- CStringArray ∗ NewCStringArrayWithStrings (size_t numstrings,...)
- CStringArray ∗ NewCStringArrayFromCArgs (int argc, char ∗argv[])
- AIORET_TYPE DeleteCStringArray (CStringArray ∗str)
- CStringArray ∗ CopyCStringArray (CStringArray ∗str)
- char ∗ CStringArrayToString (CStringArray ∗str)
- char ∗ CStringArrayToStringWithDelimeter (CStringArray ∗str, const char ∗delim)

24.134.1 Macro Definition Documentation

#define STRING_ARRAY(*N,* *...*)

24.134.2 Typedef Documentation

typedef struct **CStringArray** **CStringArray**

24.134.3 Function Documentation

CStringArray∗ NewCStringArray (size_t *numstrings*)

CStringArray∗ NewCStringArrayWithStrings (size_t *numstrings,* *...*)

CStringArray∗ NewCStringArrayFromCArgs (int *argc,* char ∗ *argv[ ]*)

AIORET_TYPE DeleteCStringArray (**CStringArray** ∗ *str*)

CStringArray∗ CopyCStringArray (**CStringArray** ∗ *str*)

char∗ CStringArrayToString (**CStringArray** ∗ *str*)

char∗ CStringArrayToStringWithDelimeter (**CStringArray** ∗ *str,* const char ∗ *delim*)

24.135 lib/DIOBuf.c File Reference

A smart buffer for handling Bit values and performing Bit arithmatic. This alleviates the need to perform bitwise operations on unsigned chars or other primitive data types in programming languages.

```
#include "DIOBuf.h"
```

Functions

- DIOBuf ∗ NewDIOBuf (unsigned size)

  Constructor for creating a new DIOBuf object.
- DIOBuf ∗ NewDIOBufFromChar (const char ∗ary, int size_array)

  Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input Ary.
- DIOBuf ∗ NewDIOBufFromBinStr (const char ∗ary)

  Constructor from a string argument like "101011011";.
- DIOBuf ∗ DIOBufReplaceString (DIOBuf ∗buf, char ∗ary, int size_array)

  Replaces the content of the buffer buf with the new array , of size size ∗.
- DIOBuf ∗ DIOBufReplaceBinString (DIOBuf ∗buf, char ∗bitstr)
- void DeleteDIOBuf (DIOBuf ∗buf)
- DIOBuf ∗ DIOBufResize (DIOBuf ∗buf, unsigned newsize)
- unsigned DIOBufSize (DIOBuf ∗buf)
- unsigned DIOBufByteSize (DIOBuf ∗buf)
- char ∗ DIOBufToString (DIOBuf ∗buf)

  Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.
- char ∗ DIOBufToHex (DIOBuf ∗buf)

  Creates a hex string representation of the DIOBuf buffer.
- char ∗ DIOBufToBinary (DIOBuf ∗buf)
- char ∗ DIOBufToInvertedBinary (DIOBuf ∗buf)

  Creates an inverted binary version of the original DIOBuf.
- AIORET_TYPE DIOBufSetIndex (DIOBuf ∗buf, int index, unsigned value)

  Sets the value of the DIOBuf buffer at index to the value specified.
- AIORET_TYPE DIOBufGetIndex (DIOBuf ∗buf, int index)

  Returns the bit value at the index specified.
- AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf ∗buf, unsigned index, char ∗value)
- AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf ∗buf, unsigned index, char value)

24.135.1 Detailed Description

A smart buffer for handling Bit values and performing Bit arithmatic. This alleviates the need to perform bitwise operations on unsigned chars or other primitive data types in programming languages.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.135.2 Function Documentation

DIOBuf∗ NewDIOBuf (unsigned *size*)

Constructor for creating a new DIOBuf object.

The parameter represents the number of *bits* that you want to have in your in your buffer. Typical values would be multiples of 8

Parameters

size	Preallocates the buffer to size

Returns

DIOBuf ∗ or Null if failure

DIOBuf∗ NewDIOBufFromChar (const char ∗ *ary,* int *size_array*)

Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input *Ary*.

Parameters

ary	
size_array	

Returns

DIOBuf if successful or NULL if there was an error. Will set errno to the reason in question but it will almost always be due to memory allocation problems.

DIOBuf∗ NewDIOBufFromBinStr (const char ∗ *ary*)

Constructor from a string argument like "101011011";.

Parameters

ary	String that contains 1's and 0's. I.E: "11110000"

Returns

DIOBuf if successful or NULL if there was an error.

DIOBuf∗ DIOBufReplaceString (DIOBuf ∗ *buf,* char ∗ *ary,* int *size_array*)

Replaces the content of the buffer buf with the new array , of size size ∗.

Parameters

buf	DIOBuf buffer one wishes to replace the content of
ary	Array of raw bytes values that will replace the original
size_array	The size, in bytes, of the *ary* that will be copied in

Returns

DIOBuf if successful or NULL if there was an error and *errno* will be set to the error in question

DIOBuf∗ DIOBufReplaceBinString (DIOBuf ∗ *buf*, char ∗ *bitstr*)

void DeleteDIOBuf (DIOBuf ∗ *buf*)

DIOBuf∗ DIOBufResize (DIOBuf ∗ *buf*, unsigned *newsize*)

unsigned DIOBufSize (DIOBuf ∗ *buf*)

unsigned DIOBufByteSize (DIOBuf ∗ *buf*)

char∗ DIOBufToString (DIOBuf ∗ *buf*)

Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.

Parameters

buf	DIOBuf one wished to print in string format

Returns

A string containing 1's and 0's if successful, NULL if failure and errno is set.

char∗ DIOBufToHex (DIOBuf ∗ *buf*)

Creates a hex string representation of the DIOBuf buffer.

This is useful for log message which require a more terse representation.

Parameters

buf	DIOBuf one wishes to convert to Hex

Returns

A Hex string , prefixed with "0x", that represents the hexidecimal representation of the DIOBuf buffer's contents. NULL indicates a failure and it sets the errno to the cause of the error.

char∗ DIOBufToBinary (DIOBuf ∗ *buf*)

char∗ DIOBufToInvertedBinary (DIOBuf ∗ *buf*)

Creates an inverted binary version of the original DIOBuf.

This is in contrast to just inverting the string of 1s to become 0s and vice versea. This is useful in

Parameters

buf	DIOBuf to invert

Returns

A binary string that represents the inverted value. NULL indicates a failure and it sets the errno

AIORET_TYPE DIOBufSetIndex (DIOBuf ∗ *buf*, int *index*, unsigned *value*)

Sets the value of the DIOBuf buffer at index to the value specified.

The value is required to be either a '0' or a '1', otherwise an error will be generated for this result.

Parameters

buf	
index	
value	A boolean value of either 0 or 1

Returns

success if >= AIOUSB_SUCCESS , < 0 otherwise

AIORET_TYPE DIOBufGetIndex (DIOBuf ∗ *buf,* int *index*)

Returns the bit value at the index specified.

Parameters

buf	DIOBuf we wish to inspect
index	Index of the bit we wish to examine

Returns

0 or 1 if successful, < 0 indicated a failure

AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char ∗ *value*)

AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char *value*)

24.136 lib/DIOBuf.h File Reference

```
#include "AIOTypes.h"
#include "AIOChannelMask.h"
#include <assert.h>
#include <math.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
```

Data Structures

- struct DIOBuf

  DIOBuf: A Smart structure for maintaining bit vectors and for providing human-readable functionality to make it easy to operate on said bit vectors.

Typedefs

- typedef unsigned char DIOBufferType

Functions

- DIOBuf ∗ NewDIOBuf (unsigned size)

  Constructor for creating a new DIOBuf object.
- DIOBuf ∗ NewDIOBufFromChar (const char ∗ary, int size_array)

  Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input Ary.
- DIOBuf ∗ NewDIOBufFromBinStr (const char ∗ary)

  Constructor from a string argument like "101011011";.
- void DeleteDIOBuf (DIOBuf ∗buf)
- DIOBuf ∗ DIOBufReplaceString (DIOBuf ∗buf, char ∗ary, int size_array)

  Replaces the content of the buffer buf with the new array , of size size ∗.
- DIOBuf ∗ DIOBufReplaceBinString (DIOBuf ∗buf, char ∗bitstr)
- char ∗ DIOBufToHex (DIOBuf ∗buf)

  Creates a hex string representation of the DIOBuf buffer.

- char ∗ DIOBufToBinary (DIOBuf ∗buf)
- char ∗ DIOBufToInvertedBinary (DIOBuf ∗buf)

  Creates an inverted binary version of the original DIOBuf.
- DIOBuf ∗ DIOBufResize (DIOBuf ∗buf, unsigned size)
- unsigned DIOBufSize (DIOBuf ∗buf)
- unsigned DIOBufByteSize (DIOBuf ∗buf)
- char ∗ DIOBufToString (DIOBuf ∗buf)

  Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.
- AIORET_TYPE DIOBufSetIndex (DIOBuf ∗buf, int index, unsigned value)

  Sets the value of the DIOBuf buffer at index to the value specified.
- AIORET_TYPE DIOBufGetIndex (DIOBuf ∗buf, int index)

  Returns the bit value at the index specified.
- AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf ∗buf, unsigned index, char ∗value)
- AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf ∗buf, unsigned index, char value)

24.136.1 Typedef Documentation

typedef unsigned char DIOBufferType

24.136.2 Function Documentation

DIOBuf∗ NewDIOBuf (unsigned *size*)

Constructor for creating a new DIOBuf object.

The parameter represents the number of *bits* that you want to have in your in your buffer. Typical values would be multiples of 8

Parameters

size	Preallocates the buffer to size

Returns

DIOBuf ∗ or Null if failure

DIOBuf∗ NewDIOBufFromChar (const char ∗ *ary,* int *size_array*)

Constructor for creating a new DIOBuf object but it accepts an array of bytes with size_array providing the length , or total number of bytes in the input *Ary*.

Parameters

ary	
size_array	

Returns

DIOBuf if successful or NULL if there was an error. Will set errno to the reason in question but it will almost always be due to memory allocation problems.

DIOBuf∗ NewDIOBufFromBinStr (const char ∗ *ary*)

Constructor from a string argument like "101011011";.

Parameters

ary	String that contains 1's and 0's. I.E: "11110000"

Returns

DIOBuf if successful or NULL if there was an error.

void DeleteDIOBuf (DIOBuf ∗ *buf*)

DIOBuf∗ DIOBufReplaceString (DIOBuf ∗ *buf,* char ∗ *ary,* int *size_array*)

Replaces the content of the buffer buf with the new array , of size size ∗.

Parameters

buf	DIOBuf buffer one wishes to replace the content of
ary	Array of raw bytes values that will replace the original
size_array	The size, in bytes, of the *ary* that will be copied in

Returns

DIOBuf if successful or NULL if there was an error and *errno* will be set to the error in question

DIOBuf∗ DIOBufReplaceBinString (DIOBuf ∗ *buf*, char ∗ *bitstr*)

char∗ DIOBufToHex (DIOBuf ∗ *buf*)

Creates a hex string representation of the DIOBuf buffer.

This is useful for log message which require a more terse representation.

Parameters

buf	DIOBuf one wishes to convert to Hex

Returns

A Hex string , prefixed with "0x", that represents the hexidecimal representation of the DIOBuf buffer's contents. NULL indicates a failure and it sets the errno to the cause of the error.

char∗ DIOBufToBinary (DIOBuf ∗ *buf*)

char∗ DIOBufToInvertedBinary (DIOBuf ∗ *buf*)

Creates an inverted binary version of the original DIOBuf.

This is in contrast to just inverting the string of 1s to become 0s and vice versea. This is useful in

Parameters

buf	DIOBuf to invert

Returns

A binary string that represents the inverted value. NULL indicates a failure and it sets the errno

DIOBuf∗ DIOBufResize (DIOBuf ∗ *buf*, unsigned *size*)

unsigned DIOBufSize (DIOBuf ∗ *buf*)

unsigned DIOBufByteSize (DIOBuf ∗ *buf*)

char∗ DIOBufToString (DIOBuf ∗ *buf*)

Converts the DIOBuf buf into a string of 1's and 0's representing the buf's value in binary.

Parameters

buf	DIOBuf one wished to print in string format

Returns

A string containing 1's and 0's if successful, NULL if failure and errno is set.

AIORET_TYPE DIOBufSetIndex (DIOBuf ∗ *buf*, int *index*, unsigned *value*)

Sets the value of the DIOBuf buffer at index to the value specified.

The value is required to be either a '0' or a '1', otherwise an error will be generated for this result.

Parameters

buf	
index	
value	A boolean value of either 0 or 1

Returns

success if >= AIOUSB_SUCCESS , < 0 otherwise

AIORET_TYPE DIOBufGetIndex (DIOBuf ∗ *buf,* int *index*)

Returns the bit value at the index specified.

Parameters

buf	DIOBuf we wish to inspect
index	Index of the bit we wish to examine

Returns

0 or 1 if successful, < 0 indicated a failure

AIORET_TYPE DIOBufGetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char ∗ *value*)

AIORET_TYPE DIOBufSetByteAtIndex (DIOBuf ∗ *buf,* unsigned *index,* char *value*)

24.137 lib/mocks/mock_aiocontbuf_get_data.c File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <libusb.h>
#include <unistd.h>
#include <sys/stat.h>
#include "aiousb.h"
#include <dlfcn.h>
```

Functions

- AIORET_TYPE aiocontbuf_get_bulk_data (AIOContinuousBuf ∗buf, USBDevice ∗usb, unsigned char endpoint, unsigned char ∗data, int datasize, int ∗bytes, unsigned timeout)

24.137.1 Function Documentation

AIORET_TYPE aiocontbuf_get_bulk_data (AIOContinuousBuf ∗ *buf,* USBDevice ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *datasize,* int ∗ *bytes,* unsigned *timeout*)

24.138 lib/mocks/mock_aiocontbuf_get_data_arduino.c File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <libusb.h>
#include <unistd.h>
#include <sys/stat.h>
#include "aiousb.h"
#include "AIOTypes.h"
#include "USBDevice.h"
#include "AIOUSB_Log.h"
#include "AIOUSB_Core.h"
#include <dlfcn.h>
```

Functions

- AIORET_TYPE aiocontbuf_get_bulk_data (AIOContinuousBuf ∗buf, USBDevice ∗usb, unsigned char endpoint, unsigned char ∗data, int datasize, int ∗bytes, unsigned timeout)
- AIORET_TYPE adc_get_bulk_data (ADCConfigBlock ∗config, USBDevice ∗usb, unsigned char endpoint, unsigned char ∗data, int datasize, int ∗bytes, unsigned timeout)
- void CloseAllDevices (void)

24.138.1 Function Documentation

AIORET_TYPE aiocontbuf_get_bulk_data (AIOContinuousBuf ∗ *buf,* USBDevice ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *datasize,* int ∗ *bytes,* unsigned *timeout*)

AIORET_TYPE adc_get_bulk_data (ADCConfigBlock ∗ *config,* USBDevice ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *datasize,* int ∗ *bytes,* unsigned *timeout*)

void CloseAllDevices (void)

24.139 lib/mocks/mock_capture_usb.c File Reference

This file will allow capturing of all USB traffic, in and out.

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include "AIOTypes.h"
#include "USBDevice.h"
#include "AIOUSB_Core.h"
#include "AIOUSB_Log.h"
#include <dlfcn.h>
```

Typedefs

- typedef AIOEither(∗ init_device)(USBDevice ∗usb, LIBUSBArgs ∗args)

Enumerations

- enum IO_DIRECTION {
  IN, OUT, IN, OUT,
  IN, OUT }

Functions

- int mock_usb_control_transfer (USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int mock_usb_bulk_transfer (USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int mock_usb_request (USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int mock_usb_reset_device (USBDevice ∗usbdev)
- int mock_usb_put_config (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int mock_usb_get_config (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- AIOEither InitializeUSBDevice (USBDevice ∗usb, LIBUSBArgs ∗args)

  Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.

Variables

- int(∗ orig_usb_control_transfer)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int(∗ orig_usb_bulk_transfer)(USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int(∗ orig_usb_request)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)

- int(∗ orig_usb_reset_device)(USBDevice ∗usbdev)
- int(∗ orig_usb_put_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int(∗ orig_usb_get_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- FILE ∗ outfile

24.139.1 Detailed Description

This file will allow capturing of all USB traffic, in and out.

Author

Jimi Damon `james.damon@accesio.com`

Date

Tue Feb 17 12:01:40 2015

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.139.2 Typedef Documentation

typedef AIOEither(∗ init_device)(USBDevice ∗usb, LIBUSBArgs ∗args)

24.139.3 Enumeration Type Documentation

enum IO_DIRECTION

Enumerator

IN

OUT

IN

OUT

IN

OUT

24.139.4 Function Documentation

int mock_usb_control_transfer (USBDevice ∗ *usbdev,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char ∗ *data,* uint16_t *wLength,* unsigned int *timeout*)

int mock_usb_bulk_transfer (USBDevice ∗ *dev_handle,* unsigned char *endpoint,* unsigned char ∗ *data,* int *length,* int ∗ *actual_length,* unsigned int *timeout*)

int mock_usb_request (USBDevice ∗ *usbdev,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char ∗ *data,* uint16_t *wLength,* unsigned int *timeout*)

int mock_usb_reset_device (USBDevice ∗ *usbdev*)

int mock_usb_put_config (USBDevice ∗ *usb,* ADCConfigBlock ∗ *configBlock*)

int mock_usb_get_config (USBDevice ∗ *usb,* ADCConfigBlock ∗ *configBlock*)

AIOEither InitializeUSBDevice (USBDevice ∗ *usb,* LIBUSBArgs ∗ *args*)

Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.

24.139.5 Variable Documentation

int(∗ orig_usb_control_transfer)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)

int(∗ orig_usb_bulk_transfer)(USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)

int(∗ orig_usb_request)(USBDevice ∗usbdev, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)

int(∗ orig_usb_reset_device)(USBDevice ∗usbdev)

int(∗ orig_usb_put_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)

int(∗ orig_usb_get_config)(USBDevice ∗usb, ADCConfigBlock ∗configBlock)

FILE∗ outfile

24.140 lib/mocks/mock_dio.c File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <libusb.h>
#include <unistd.h>
#include <sys/stat.h>
#include "AIOTypes.h"
#include "USBDevice.h"
#include "AIOUSB_Core.h"
#include <dlfcn.h>
```

Enumerations

- enum IO_DIRECTION {
  IN, OUT, IN, OUT,
  IN, OUT }

Functions

- void save_results (char ∗prefix, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int mock_usb_control_transfer (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int mock_usb_bulk_transfer (USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)

24.140.1 Enumeration Type Documentation

enum IO_DIRECTION

Enumerator

IN

OUT

IN

OUT

IN

OUT

24.140.2 Function Documentation

void save_results (char ∗ *prefix*, uint8_t *request_type*, uint8_t *bRequest*, uint16_t *wValue*, uint16_t *wIndex*, unsigned char ∗ *data*, uint16_t *wLength*, unsigned int *timeout*)

Improve this to allow the utility to make the directory in question and write the results there

int mock_usb_control_transfer (USBDevice ∗ *dev_handle*, uint8_t *request_type*, uint8_t *bRequest*, uint16_t *wValue*, uint16_t *wIndex*, unsigned char ∗ *data*, uint16_t *wLength*, unsigned int *timeout*)

int mock_usb_bulk_transfer (USBDevice ∗ *dev_handle*, unsigned char *endpoint*, unsigned char ∗ *data*, int *length*, int ∗ *actual_length*, unsigned int *timeout*)

24.141 lib/mocks/mock_usb_xfers.c File Reference

```
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#include "AIOTypes.h"
#include "USBDevice.h"
#include "AIOUSB_Core.h"
#include "AIOUSB_Log.h"
#include "libusb.h"
#include <assert.h>
#include <dlfcn.h>
```

Typedefs

- typedef AIORET_TYPE(∗ add_devices_fn)(libusb_device ∗∗deviceList, USBDevice ∗∗devs, int ∗size)

Enumerations

- enum IO_DIRECTION {
  IN, OUT, IN, OUT,
  IN, OUT }

Functions

- int mock_usb_control_transfer (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int mock_usb_bulk_transfer (USBDevice ∗usb, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int mock_usb_reset_device (USBDevice ∗usb)
- int mock_USBDevicePutADCConfigBlock (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int mock_USBDeviceFetchADCConfigBlock (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗deviceList, USBDevice ∗∗devs, int ∗size)

Variables

- IO_DIRECTION direction
- ADCConfigBlock ∗ KEEP = NULL

24.141.1 Typedef Documentation

typedef AIORET_TYPE(∗ add_devices_fn)(libusb_device ∗∗deviceList, USBDevice ∗∗devs, int ∗size)

24.141.2 Enumeration Type Documentation

enum IO_DIRECTION

Enumerator

IN

OUT

IN

OUT

IN

OUT

24.141.3 Function Documentation

int mock_usb_control_transfer (USBDevice ∗ *dev_handle*, uint8_t *request_type*, uint8_t *bRequest*, uint16_t *wValue*, uint16_t *wIndex*, unsigned char ∗ *data*, uint16_t *wLength*, unsigned int *timeout*)

int mock_usb_bulk_transfer (USBDevice ∗ *usb*, unsigned char *endpoint*, unsigned char ∗ *data*, int *length*, int ∗ *actual_length*, unsigned int *timeout*)

int mock_usb_reset_device (USBDevice ∗ *usb*)

int mock_USBDevicePutADCConfigBlock (USBDevice ∗ *usb*, ADCConfigBlock ∗ *configBlock*)

int mock_USBDeviceFetchADCConfigBlock (USBDevice ∗ *usb*, ADCConfigBlock ∗ *configBlock*)

AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗ *deviceList*, USBDevice ∗∗ *devs*, int ∗ *size*)

24.141.4 Variable Documentation

IO_DIRECTION direction

ADCConfigBlock∗ KEEP = NULL

24.142 lib/USBDevice.c File Reference

```
#include "AIOTypes.h"
#include "USBDevice.h"
#include "libusb.h"
#include "AIODeviceTable.h"
#include "AIOEither.h"
```

Functions

- AIOEither InitializeUSBDevice (USBDevice ∗usb, LIBUSBArgs ∗args)

  Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.
- USBDevice ∗ NewUSBDevice (libusb_device ∗dev, libusb_device_handle ∗handle)
- USBDevice ∗ CopyUSBDevice (USBDevice ∗usb)
- int USBDeviceClose (USBDevice ∗usb)
- AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗deviceList, USBDevice ∗∗devs, int ∗size)
- AIORET_TYPE AddDevice (int ∗size, int index, libusb_device ∗∗deviceList, USBDevice ∗∗devs, struct libusb_-device_descriptor ∗libusbDeviceDesc)
- int USBDeviceGetIdProduct (USBDevice ∗device)
- void DeleteUSBDevices (USBDevice ∗devices)
- void DeleteUSBDevice (USBDevice ∗dev)
- int USBDeviceSetDebug (USBDevice ∗usb, AIOUSB_BOOL debug)
- libusb_device_handle ∗ USBDeviceGetUSBDeviceHandle (USBDevice ∗usb)
- libusb_device_handle ∗ get_usb_device (USBDevice ∗dev)
- int USBDeviceFetchADCConfigBlock (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int USBDevicePutADCConfigBlock (USBDevice ∗usb, ADCConfigBlock ∗configBlock)

- int usb_control_transfer (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int usb_bulk_transfer (USBDevice ∗usb, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_-length, unsigned int timeout)
- int usb_request (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t w-Index, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int usb_reset_device (USBDevice ∗usb)

24.142.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.142.2 Function Documentation

AIOEither InitializeUSBDevice (USBDevice ∗ *usb*, LIBUSBArgs ∗ *args*)

Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.

USBDevice∗ NewUSBDevice (libusb_device ∗ *dev*, libusb_device_handle ∗ *handle*)

USBDevice∗ CopyUSBDevice (USBDevice ∗ *usb*)

int USBDeviceClose (USBDevice ∗ *usb*)

AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗ *deviceList*, USBDevice ∗∗ *devs*, int ∗ *size*)

AIORET_TYPE AddDevice (int ∗ *size*, int *index*, libusb_device ∗∗ *deviceList*, USBDevice ∗∗ *devs*, struct libusb_device_descriptor ∗ *libusbDeviceDesc*)

int USBDeviceGetIdProduct (USBDevice ∗ *device*)

void DeleteUSBDevices (USBDevice ∗ *devices*)

void DeleteUSBDevice (USBDevice ∗ *dev*)

int USBDeviceSetDebug (USBDevice ∗ *usb*, AIOUSB_BOOL *debug*)

libusb_device_handle∗ USBDeviceGetUSBDeviceHandle (USBDevice ∗ *usb*)

libusb_device_handle∗ get_usb_device (USBDevice ∗ *dev*)

int USBDeviceFetchADCConfigBlock (USBDevice ∗ *usb*, ADCConfigBlock ∗ *configBlock*)

int USBDevicePutADCConfigBlock (USBDevice ∗ *usb*, ADCConfigBlock ∗ *configBlock*)

int usb_control_transfer (USBDevice ∗ *dev_handle*, uint8_t *request_type*, uint8_t *bRequest*, uint16_t *wValue*, uint16_t *wIndex*, unsigned char ∗ *data*, uint16_t *wLength*, unsigned int *timeout*)

int usb_bulk_transfer (USBDevice ∗ *usb*, unsigned char *endpoint*, unsigned char ∗ *data*, int *length*, int ∗ *actual_length*, unsigned int *timeout*)

This function is intended to improve upon libusb_bulk_transfer() by receiving or transmitting packets until the entire transfer request has been satisfied; it intentionally restarts the timeout each time a packet is received, so the timeout parameter specifies the longest permitted delay between packets, not the total time to complete the transfer request

Note

even if we get a timeout, some data may have been transferred; if so, then this timeout is not an error; if we get a timeout and no data was transferred, then treat it as an error condition

int usb_request (USBDevice ∗ *dev_handle*, uint8_t *request_type*, uint8_t *bRequest*, uint16_t *wValue*, uint16_t *wIndex*, unsigned char ∗ *data*, uint16_t *wLength*, unsigned int *timeout*)

int usb_reset_device (USBDevice ∗ *usb*)

24.143 lib/USBDevice.h File Reference

```
#include <stdint.h>
#include <libusb.h>
#include <stdlib.h>
#include "ADCConfigBlock.h"
#include "AIOEither.h"
```

Data Structures

- struct USBDevice
- struct aiousb_libusb_args

Macros

- #define INTERNAL_METHOD(NAME, RETVAL,...) RETVAL (∗NAME)(__VA_ARGS__)

Typedefs

- typedef struct USBDevice USBDevice
- typedef struct aiousb_libusb_args LIBUSBArgs

Functions

- USBDevice ∗ NewUSBDevice (libusb_device ∗dev, libusb_device_handle ∗handle)
- void DeleteUSBDevice (USBDevice ∗dev)
- USBDevice ∗ CopyUSBDevice (USBDevice ∗usb)
- AIOEither InitializeUSBDevice (USBDevice ∗usb, LIBUSBArgs ∗args)

  Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.
- AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗deviceList, USBDevice ∗∗devs, int ∗size)
- void DeleteUSBDevices (USBDevice ∗devs)
- int USBDeviceClose (USBDevice ∗dev)
- int USBDeviceGetIdProduct (USBDevice ∗device)
- int USBDeviceFetchADCConfigBlock (USBDevice ∗device, ADCConfigBlock ∗config)
- int USBDevicePutADCConfigBlock (USBDevice ∗usb, ADCConfigBlock ∗configBlock)
- int usb_control_transfer (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t wIndex, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int usb_bulk_transfer (USBDevice ∗dev_handle, unsigned char endpoint, unsigned char ∗data, int length, int ∗actual_length, unsigned int timeout)
- int usb_request (USBDevice ∗dev_handle, uint8_t request_type, uint8_t bRequest, uint16_t wValue, uint16_t w-Index, unsigned char ∗data, uint16_t wLength, unsigned int timeout)
- int usb_reset_device (USBDevice ∗usb)
- libusb_device_handle ∗ get_usb_device (USBDevice ∗dev)
- libusb_device_handle ∗ USBDeviceGetUSBDeviceHandle (USBDevice ∗usb)

24.143.1 Macro Definition Documentation

#define INTERNAL_METHOD(***NAME,*** ***RETVAL,*** ***...*** **) RETVAL (∗NAME)(__VA_ARGS__)**

24.143.2 Typedef Documentation

typedef struct USBDevice USBDevice

typedef struct aiousb_libusb_args LIBUSBArgs

24.143.3 Function Documentation

USBDevice∗ NewUSBDevice (libusb_device ∗ *dev,* libusb_device_handle ∗ *handle*)

void DeleteUSBDevice (USBDevice ∗ *dev*)

USBDevice∗ CopyUSBDevice (USBDevice ∗ *usb*)

AIOEither InitializeUSBDevice (USBDevice ∗ *usb,* LIBUSBArgs ∗ *args*)

Wraps the initial IntializeUSBDevice, and records mock functions that will call the initial values.

AIORET_TYPE AddAllACCESUSBDevices (libusb_device ∗∗ *deviceList,* USBDevice ∗∗ *devs,* int ∗ *size*)

void DeleteUSBDevices (USBDevice ∗ *devs*)

int USBDeviceClose (USBDevice ∗ *dev*)

int USBDeviceGetIdProduct (USBDevice ∗ *device*)

int USBDeviceFetchADCConfigBlock (USBDevice ∗ *device,* ADCConfigBlock ∗ *config*)

int USBDevicePutADCConfigBlock (USBDevice ∗ *usb,* ADCConfigBlock ∗ *configBlock*)

int usb_control_transfer (USBDevice ∗ *dev_handle,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char ∗ *data,* uint16_t *wLength,* unsigned int *timeout*)

int usb_bulk_transfer (USBDevice ∗ *usb,* unsigned char *endpoint,* unsigned char ∗ *data,* int *length,* int ∗ *actual_length,* unsigned int *timeout*)

This function is intended to improve upon libusb_bulk_transfer() by receiving or transmitting packets until the entire transfer request has been satisfied; it intentionally restarts the timeout each time a packet is received, so the timeout parameter specifies the longest permitted delay between packets, not the total time to complete the transfer request

Note

even if we get a timeout, some data may have been transferred; if so, then this timeout is not an error; if we get a timeout and no data was transferred, then treat it as an error condition

int usb_request (USBDevice ∗ *dev_handle,* uint8_t *request_type,* uint8_t *bRequest,* uint16_t *wValue,* uint16_t *wIndex,* unsigned char ∗ *data,* uint16_t *wLength,* unsigned int *timeout*)

int usb_reset_device (USBDevice ∗ *usb*)

libusb_device_handle∗ get_usb_device (USBDevice ∗ *dev*)

libusb_device_handle∗ USBDeviceGetUSBDeviceHandle (USBDevice ∗ *usb*)

24.144 lib/wrappers/scilab/foo.c File Reference

```
#include "foo.h"
#include <stdio.h>
```

Functions

- void foo_something ()

24.144.1 Function Documentation

void foo_something ( )

24.145 lib/wrappers/scilab/foo.h File Reference

Functions

- void foo_something ()

24.145.1 Function Documentation

void foo_something ( )

24.146 samples/TestLib/aiocommon.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include <ctype.h>
#include "aiocommon.h"
#include "aiousb.h"
```

Macros

- #define DUMP 0x1000
- #define CNTS 0x1001
- #define JCONF 0x1002
- #define REPEAT 0x1003

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_aio_cmd_line (struct opts ∗options, int argc, char ∗argv[ ])

  Simple command line parser sets up testing features.
- void print_aio_usage (int argc, char ∗∗argv, struct option ∗options)
- AIORET_TYPE aio_list_devices (struct opts ∗options, int ∗indices, int num_devices)
- AIORET_TYPE aio_override_adcconfig_settings (ADCConfigBlock ∗config, struct opts ∗options)
- AIORET_TYPE aio_supply_default_command_line_settings (struct opts ∗options)
- AIORET_TYPE aio_override_aiobuf_settings (AIOContinuousBuf ∗buf, struct opts ∗options)

Variables

- struct opts AIO_OPTIONS

24.146.1 Macro Definition Documentation

#define DUMP 0x1000

#define CNTS 0x1001

#define JCONF 0x1002

#define REPEAT 0x1003

24.146.2 Function Documentation

struct channel_range∗ get_channel_range (char ∗ *optarg*)

void process_aio_cmd_line (struct opts ∗ *options,* int *argc,* char ∗ *argv[ ]*)

Simple command line parser sets up testing features.

void print_aio_usage (int *argc,* char ∗∗ *argv,* struct option ∗ *options*)

AIORET_TYPE aio_list_devices (struct **opts** ∗ *options,* int ∗ *indices,* int *num_devices*)

AIORET_TYPE aio_override_adcconfig_settings (**ADCConfigBlock** ∗ *config,* struct **opts** ∗ *options*)

AIORET_TYPE aio_supply_default_command_line_settings (struct **opts** ∗ *options*)

AIORET_TYPE aio_override_aiobuf_settings (**AIOContinuousBuf** ∗ *buf,* struct **opts** ∗ *options*)

24.146.3 Variable Documentation

struct **opts** AIO_OPTIONS

24.147 samples/TestLib/aiocommon.h File Reference

```
#include "aiousb.h"
#include <getopt.h>
#include <stdint.h>
```

Data Structures

- struct channel_range
- struct opts

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_aio_cmd_line (struct opts ∗options, int argc, char ∗argv[ ])

    Simple command line parser sets up testing features.
- void print_aio_usage (int argc, char ∗∗argv, struct option ∗options)
- AIORET_TYPE aio_list_devices (struct opts ∗options, int ∗indices, int num_devices)
- AIORET_TYPE aio_override_aiobuf_settings (AIOContinuousBuf ∗buf, struct opts ∗options)
- AIORET_TYPE aio_override_adcconfig_settings (ADCConfigBlock ∗config, struct opts ∗options)
- AIORET_TYPE aio_supply_default_command_line_settings (struct opts ∗options)

Variables

- struct opts AIO_OPTIONS

24.147.1 Function Documentation

struct **channel_range**∗ get_channel_range (char ∗ *optarg*)

void process_aio_cmd_line (struct **opts** ∗ *options,* int *argc,* char ∗ *argv[ ]*)

Simple command line parser sets up testing features.

void print_aio_usage (int *argc,* char ∗∗ *argv,* struct option ∗ *options*)

AIORET_TYPE aio_list_devices (struct **opts** ∗ *options,* int ∗ *indices,* int *num_devices*)

AIORET_TYPE aio_override_aiobuf_settings (**AIOContinuousBuf** ∗ *buf,* struct **opts** ∗ *options*)

AIORET_TYPE aio_override_adcconfig_settings (**ADCConfigBlock** ∗ *config,* struct **opts** ∗ *options*)

AIORET_TYPE aio_supply_default_command_line_settings (struct **opts** ∗ *options*)

24.147.2 Variable Documentation

struct **opts** AIO_OPTIONS

24.148 samples/TestLib/TestCaseSetup.cpp File Reference

```
#include "TestCaseSetup.h"
#include <stdlib.h>
#include <aiousb.h>
#include <AIOUSB_Core.h>
#include "AIOTypes.h"
```

Variables

- int CURRENT_DEBUG_LEVEL = 1

24.148.1 Variable Documentation

int CURRENT_DEBUG_LEVEL = 1

24.149 samples/TestLib/TestCaseSetup.h File Reference

```
#include <aiousb.h>
#include <exception>
#include <iostream>
#include <sstream>
#include <stdio.h>
#include <unistd.h>
#include <stdarg.h>
#include "AIOUSB_Core.h"
```

Data Structures

- class Error
- class TestCaseSetup

Macros

- #define ERROR_LEVEL 2<<1
- #define FATAL_LEVEL 2<<1
- #define ALERT_LEVEL 2<<2
- #define WARN_LEVEL 2<<3
- #define INFO_LEVEL 2<<4
- #define DEBUG_LEVEL 2<<5
- #define TRACE_LEVEL 2<<6
- #define LOG(X,...) printf(X, ##__VA_ARGS__);
- #define INFO(X,...)
- #define TRACE(X,...)
- #define DEBUG(X,...)
- #define ERROR(X,...)
- #define FATAL(X,...)
- #define TERSE_LOGGING (WARN_LEVEL | ERROR_LEVEL | INFO_LEVEL)
- #define VERBOSE_LOGGING (DEBUG_LEVEL | INFO_LEVEL | WARN_LEVEL | ERROR_LEVEL)
- #define THROW_ERROR(x) ThrowError(x , __LINE__)
- #define CHECK_RESULT(x) if(result != AIOUSB_SUCCESS) ThrowError(result,__LINE__);

Variables

- const int MAX_NAME_SIZE = 20
- const int DEF_MAX_CHANNELS = 128
- const int DEF_NUM_CHANNELS = 16
- const int DEF_CAL_CHANNEL = 5
- int CURRENT_DEBUG_LEVEL

24.149.1 Macro Definition Documentation

#define ERROR_LEVEL 2<<1

#define FATAL_LEVEL 2<<1

#define ALERT_LEVEL 2<<2

#define WARN_LEVEL 2<<3

#define INFO_LEVEL 2<<4

#define DEBUG_LEVEL 2<<5

#define TRACE_LEVEL 2<<6

#define LOG(*X,* ...) printf(X, ##__VA_ARGS__);

#define INFO(*X,* ...)

Value:

```
if( CURRENT_DEBUG_LEVEL & INFO_LEVEL ) \
                                printf(X, ##__VA_ARGS__);
```

#define TRACE(*X,* ...)

Value:

```
if( CURRENT_DEBUG_LEVEL & TRACE_LEVEL ) \
                                printf(X, ##__VA_ARGS__ );
```

#define DEBUG(*X,* ...)

Value:

```
if( CURRENT_DEBUG_LEVEL & DEBUG_LEVEL ) \
                                printf(X, ##__VA_ARGS__ );
```

#define ERROR(*X,* ...)

Value:

```
if( CURRENT_DEBUG_LEVEL & ERROR_LEVEL ) \
                                printf(X, ##__VA_ARGS__ );
```

#define FATAL(*X,* ...)

Value:

```
if( CURRENT_DEBUG_LEVEL & ERROR_LEVEL ) \
                                printf(X, ##__VA_ARGS__ );
```

#define TERSE_LOGGING (WARN_LEVEL | ERROR_LEVEL | INFO_LEVEL)

#define VERBOSE_LOGGING (DEBUG_LEVEL | INFO_LEVEL | WARN_LEVEL | ERROR_LEVEL)

#define THROW_ERROR(*x*) ThrowError(x , __LINE__)

#define CHECK_RESULT(*x*) if(result != AIOUSB_SUCCESS) ThrowError(result,__LINE__);

24.149.2 Variable Documentation

const int MAX_NAME_SIZE = 20

const int DEF_MAX_CHANNELS = 128

const int DEF_NUM_CHANNELS = 16

const int DEF_CAL_CHANNEL = 5

int CURRENT_DEBUG_LEVEL

24.150 samples/USB-AI16-16/bulk_acquire_sample.c File Reference

```
#include <aiousb.h>
#include <math.h>
#include <stdio.h>
#include <unistd.h>
#include <getopt.h>
```

Data Structures

- struct opts

Functions

- void process_cmd_line (struct opts *, int argc, char *argv[])
- int main (int argc, char **argv)
- void print_usage (int argc, char **argv, struct option *options)

24.150.1 Function Documentation

void process_cmd_line (struct opts * *options,* int *argc,* char * *argv[]*)

int main (int *argc,* char ** *argv*)

make sure counter is stopped

void print_usage (int *argc,* char ** *argv,* struct option * *options*)

24.151 samples/USB-AI16-16/burst_test.c File Reference

```
#include <stdio.h>
#include <aiousb.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <ctype.h>
#include <time.h>
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice *dev)

24.151.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.151.2 Function Documentation

AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗ *dev*)

24.152 samples/USB-AI16-16/continuous_mode.c File Reference

```
#include <aiousb.h>
#include "aiocommon.h"
```

Functions

- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)

24.152.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.152.2 Function Documentation

AIOUSB_BOOL fnd (AIOUSBDevice ∗ *dev*)

24.153 samples/USB-AI16-16/continuous_mode_callback.c File Reference

```
#include <stdio.h>
#include <aiousb.h>
#include <unistd.h>
#include <math.h>
#include <ctype.h>
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <signal.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[ ])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- AIORET_TYPE capture_data (AIOContinuousBuf ∗buf)
- int main (int argc, char ∗argv[ ])

Variables

- FILE ∗ fp

24.153.1 Function Documentation

struct channel_range∗ get_channel_range (char ∗ *optarg*)

void process_cmd_line (struct opts ∗ , int *argc,* char ∗ *argv[ ]*)

AIOUSB_BOOL fnd (AIOUSBDevice ∗ *dev*)

AIORET_TYPE capture_data (AIOContinuousBuf ∗ *buf*)

int main (int *argc,* char ∗ *argv[ ]*)

24.153.2 Variable Documentation

FILE∗ fp

24.154 samples/USB-AI16-16/continuous_mode_from_json_config.c File Reference

```
#include "aiocommon.h"
#include <aiousb.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[ ])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- AIORET_TYPE capture_data (AIOContinuousBuf ∗buf)
- int main (int argc, char ∗argv[ ])

Variables

- FILE ∗ fp

24.154.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.154.2 Function Documentation

struct **channel_range**∗ get_channel_range (char ∗ *optarg*)

void process_cmd_line (struct **opts** ∗ , int *argc,* char ∗ *argv[]*)

AIOUSB_BOOL fnd (**AIOUSBDevice** ∗ *dev*)

AIORET_TYPE capture_data (**AIOContinuousBuf** ∗ *buf*)

int main (int *argc,* char ∗ *argv[]*)

24.154.3 Variable Documentation

FILE∗ fp

24.155 samples/USB-AI16-16/daitest.cpp File Reference

```
#include <iostream>
#include <thread>
#include <chrono>
#include "aiousb.h"
#include <signal.h>
#include <unistd.h>
```

Functions

- void handle_signal (int signal)
- int main (int argc, char ∗argv[])

Variables

- int exit_sample = 0
- struct sigaction old_action

24.155.1 Function Documentation

void handle_signal (int *signal*)

int main (int *argc,* char ∗ *argv[]*)

24.155.2 Variable Documentation

int exit_sample = 0

struct sigaction old_action

24.156 samples/USB-AI16-16/dio_sample.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "aiocommon.h"
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗dev)
- int main (int argc, char ∗∗argv)

24.156.1 Function Documentation

AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗ *dev*)

int main (int *argc*, char ∗∗ *argv*)

Make all ports outputs and 0 value as the initial tristate

24.157 samples/USB-AI16-16/diotest.c File Reference

```
#include <stdio.h>
#include "aiousb.h"
```

Functions

- int main (int argc, char ∗argv[ ])

24.157.1 Function Documentation

int main (int *argc*, char ∗ *argv[ ]*)

24.158 samples/USB-AI16-16/diotest2.cpp File Reference

```
#include <iostream>
#include "aiousb.h"
```

Functions

- int main (int argc, char ∗argv[ ])

24.158.1 Function Documentation

int main (int *argc*, char ∗ *argv[ ]*)

24.159 samples/USB-AI16-16/HOLD/dirktest.c File Reference

```
#include <stdio.h>
#include <aiousb.h>
#include <unistd.h>
#include <math.h>
#include <ctype.h>
#include <AIODataTypes.h>
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <signal.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[ ])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- AIORET_TYPE capture_data (AIOContinuousBuf ∗buf)
- int main (int argc, char ∗argv[ ])

Variables

- FILE ∗ fp

24.159.1 Function Documentation

struct **channel_range**∗ get_channel_range (char ∗ *optarg*)

void process_cmd_line (struct **opts** ∗ , int *argc,* char ∗ *argv[]*)

AIOUSB_BOOL fnd (**AIOUSBDevice** ∗ *dev*)

AIORET_TYPE capture_data (**AIOContinuousBuf** ∗ *buf*)

int main (int *argc,* char ∗ *argv[]*)

24.159.2 Variable Documentation

FILE∗ fp

24.160 samples/USB-AI16-16/HOLD/julian_test.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "aiocommon.h"
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗dev)
- int main (int argc, char ∗∗argv)

24.160.1 Function Documentation

AIOUSB_BOOL find_ai_board (**AIOUSBDevice** ∗ *dev*)

int main (int *argc,* char ∗∗ *argv*)

24.161 samples/USB-AI16-16/HOLD/reverse_cal_table.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <exception>
#include <iostream>
#include "TestCaseSetup.h"
```

Functions

- void goDoIt (TestCaseSetup &t)
- int main (int argc, char ∗∗argv)

24.161.1 Function Documentation

void goDoIt (**TestCaseSetup** & *t*)

int main (int *argc,* char ∗∗ *argv*)

24.162 samples/USB-AI16-16/reverse_cal_table.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <exception>
#include <iostream>
#include "TestCaseSetup.h"
```

Functions

- void goDoIt (TestCaseSetup &t)
- int main (int argc, char **argv)

24.162.1 Function Documentation

void goDoIt (**TestCaseSetup** & *t*)

int main (int *argc,* char ** *argv*)

24.163 samples/USB-AI16-16/HOLD/sample_dio.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "aiocommon.h"
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice *dev)
- int main (int argc, char **argv)

24.163.1 Function Documentation

AIOUSB_BOOL find_ai_board (**AIOUSBDevice** * *dev*)

int main (int *argc,* char ** *argv*)

Make all ports outputs and 0 value as the initial tristate

24.164 samples/USB-AI16-16/HOLD/slow_receiver_test.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <getopt.h>
#include <vector>
```

Data Structures

- struct config_options

Functions

- struct config_options * process_cmd_line (int argc, char **argv)
- void print_usage (int argc, char **argv, struct option *options)
- int main (int argc, char **argv)

24.164.1 Function Documentation

struct **config_options** * process_cmd_line (int *argc,* char ** *argv*)

void print_usage (int *argc,* char ** *argv,* struct option * *options*) int main (int *argc,* char ∗∗ *argv*)

24.165 samples/USB-AI16-16/HOLD/test.c File Reference

```
#include <stdio.h>
#include <aiousb.h>
#include <unistd.h>
#include <math.h>
#include <ctype.h>
#include <AIODataTypes.h>
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <signal.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- AIORET_TYPE capture_data (AIOContinuousBuf ∗buf)
- int main (int argc, char ∗argv[])

Variables

- FILE ∗ fp

24.165.1 Function Documentation

struct **channel_range**∗ get_channel_range (char ∗ *optarg*)

void process_cmd_line (struct **opts** ∗ , int *argc,* char ∗ *argv[]*)

AIOUSB_BOOL fnd (**AIOUSBDevice** ∗ *dev*)

AIORET_TYPE capture_data (**AIOContinuousBuf** ∗ *buf*)

int main (int *argc,* char ∗ *argv[]*)

24.165.2 Variable Documentation

FILE∗ fp

24.166 samples/USB-AI16-16/java/extcal/src/main/java/com/accesio/extcal.cpp File Reference

```
#include <iostream>
#include <iterator>
#include <iomanip>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <aiousb.h>
#include <AIOUSB_Core.h>
#include <USBDeviceManager.hpp>
#include <USB_AI16_Family.hpp>
```

Functions

- int main (int argc, char ∗argv[])

24.166.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Author

Jimi Damon jdamon@accesio.com

Version

Format:

h

24.166.2 Function Documentation

int main (int *argc,* char ∗ *argv[]*)

24.167 samples/USB-AI16-16/jni/read_channels_test.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗dev)
- int main (int argc, char ∗∗argv)

24.167.1 Function Documentation

AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗ *dev*)

int main (int *argc,* char ∗∗ *argv*)

these functions are not needed IF you use the function AIOProcessCmdLine(). It will make the following call sequence

```
* retval = AIOUSB_Init();
* if ( retval != AIOUSB_SUCCESS ) {
*     fprintf(stderr,"Error calling AIOUSB_Init(): %d\n", (int)retval );
*     exit(retval );
* }
* AIOUSB_ListDevices();
*
```

this is the preferred form over

```
*        retval = AIOProcessCmdline( options, argc, argv );
* @verbatim
*        in the case where you want your program to process
*        extra options on the command line as the options not
*        parsed by AIOProcessCommandLine will be retained
*        in argv.
*
```

Copy the modified config settings back to the device ave config to the device

24.168 samples/USB-AI16-16/read_channels_test.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice *dev)
- int main (int argc, char **argv)

24.168.1 Function Documentation

AIOUSB_BOOL find_ai_board (AIOUSBDevice * *dev*)

int main (int *argc*, char ** *argv*)

these functions are not needed IF you use the function AIOProcessCmdLine(). It will make the following call sequence

```
* retval = AIOUSB_Init();
* if ( retval != AIOUSB_SUCCESS ) {
*     fprintf(stderr,"Error calling AIOUSB_Init(): %d\n", (int)retval );
*     exit(retval );
* }
* AIOUSB_ListDevices();
*
```

this is the preferred form over

```
*        retval = AIOProcessCmdline( options, argc, argv );
* @verbatim
*        in the case where you want your program to process
*        extra options on the command line as the options not
*        parsed by AIOProcessCommandLine will be retained
*        in argv.
*
```

Copy the modified config settings back to the device ave config to the device

24.169 samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <exception>
#include <iostream>
#include "TestCaseSetup.h"
```

Data Structures

- struct options

Functions

- struct options get_options (int argc, char **argv)
- int main (int argc, char **argv)

24.169.1 Function Documentation

struct options get_options (int *argc*, char ** *argv*)

int main (int *argc*, char ** *argv*)

24.170 samples/USB-AI16-16/sample.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
```

Functions

- int main (int argc, char ∗∗argv)

24.170.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

call GetDevices() to obtain "list" of devices found on the bus

print list of all devices found on the bus

demonstrate automatic A/D calibration

24.171 samples/USB-AO16-16/sample.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
```

Functions

- int main (int argc, char ∗∗argv)

24.171.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

24.172 samples/USB-DA12-8A/sample.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
```

Functions

- int main (int argc, char ∗∗argv)

24.172.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

24.173 samples/USB-DIO-16/sample.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Functions

- int main (int argc, char ∗∗argv)

24.173.1 Function Documentation

int main (int *argc*, char ∗∗ *argv*)

24.174 samples/USB-AI16-16/simp_test.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <exception>
#include <iostream>
#include "TestCaseSetup.h"
```

Functions

- int main (int argc, char ∗∗argv)

24.174.1 Function Documentation

int main (int *argc*, char ∗∗ *argv*)

24.175 samples/USB-AI16-16/simple_continuous_with_json.c File Reference

Sample that demonstrates data ac.

```
#include <stdio.h>
#include <aiousb.h>
#include <unistd.h>
#include <math.h>
#include <ctype.h>
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <signal.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[])
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- int main (int argc, char ∗argv[])

Variables

- FILE ∗ fp

24.175.1 Detailed Description

Sample that demonstrates data ac.

Author

Jimi Damon `james.damon@accesio.com`

Date

Thu Nov 12 10:54:48 2015

24.175.2 Function Documentation

struct **channel_range**∗ get_channel_range (char ∗ *optarg*)

void process_cmd_line (struct opts ∗ , int *argc*, char ∗ *argv[]*)

AIOUSB_BOOL fnd (AIOUSBDevice ∗ *dev*)

int main (int *argc*, char ∗ *argv[]*)

Start with the NewAIOContinousBufFromJSON("{'aiocontin

24.175.3 Variable Documentation

FILE∗ fp

24.176 samples/USB-AI16-16/start_stop_continuous.c File Reference

```
#include <stdio.h>
#include <aiousb.h>
#include <unistd.h>
#include <math.h>
#include <ctype.h>
#include "AIOCountsConverter.h"
#include "AIOUSB_Log.h"
#include "aiocommon.h"
#include <getopt.h>
#include <signal.h>
```

Functions

- struct channel_range ∗ get_channel_range (char ∗optarg)
- void process_cmd_line (struct opts ∗, int argc, char ∗argv[])
- void run_acquisition (AIOContinuousBuf ∗buf, struct opts ∗options)
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- int main (int argc, char ∗argv[])

Variables

- FILE ∗ fp

24.176.1 Function Documentation

struct channel_range∗ get_channel_range (char ∗ *optarg*)

void process_cmd_line (struct opts ∗ , int *argc*, char ∗ *argv[]*)

void run_acquisition (AIOContinuousBuf ∗ *buf*, struct opts ∗ *options*)

AIOUSB_BOOL fnd (AIOUSBDevice ∗ *dev*)

int main (int *argc*, char ∗ *argv[]*)

Start with the NewAIOContinousBufFromJSON("{'aiocontin

24.176.2 Variable Documentation

FILE∗ fp

24.177 samples/USB-AI16-16/test_fastscan.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <exception>
#include <iostream>
#include "TestCaseSetup.h"
```

Functions

- int main (int argc, char ∗∗argv)

Variables

- int CURRENT_DEBUG_LEVEL

24.177.1 Function Documentation

int main (int *argc*, char ∗∗ *argv*)

24.177.2 Variable Documentation

int CURRENT_DEBUG_LEVEL

24.178 samples/USB-ARB1/stream_test.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include "aiocommon.h"
```

Functions

- AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗dev)
- int main (int argc, char ∗∗argv)

24.178.1 Function Documentation

AIOUSB_BOOL find_ai_board (AIOUSBDevice ∗ *dev*)

int main (int *argc*, char ∗∗ *argv*)

24.179 samples/USB-DA12-8A/SampleClass.cpp File Reference

```
#include <iostream>
#include <iterator>
#include <iomanip>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <USBDeviceManager.hpp>
#include <USB_DA12_8A_Family.hpp>
```

Functions

- int main (int argc, char ∗argv[])

24.179.1 Function Documentation

int main (int *argc*, char ∗ *argv[]*)

24.180 samples/USB-DIO-16/receiver.cpp File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Functions

- int main (int argc, char ∗∗argv)

24.180.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

24.181 samples/USB-DIO-16/standalone_receiver.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
```

Functions

- int main (int argc, char ∗∗argv)

24.181.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

24.182 samples/USB-DIO-32/daisample.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "AIOTypes.h"
```

Functions

- int main (int argc, char ∗∗argv)

24.182.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

24.183 samples/USB-DIO-32/read_and_write_sample.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "AIOTypes.h"
```

Functions

- int main (int argc, char ∗∗argv)

24.183.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

See Also

Compilation
CmakeCompilation

< GetDevices

24.184 samples/USB-DIO-96/read_and_write_sample.c File Reference

```
#include "AIOTypes.h"
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "AIOChannelMask.h"
```

Data Structures

- struct DeviceInfo

Macros

- #define BITS_PER_BYTE 8
- #define DEVICES_REQUIRED 1
- #define MAX_DIO_BYTES 12
- #define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)
- #define MAX_NAME_SIZE 20

Enumerations

- enum EXIT_CODE {
  SUCCESS = 0, USB_ERROR = -1, NO_DEVICE_FOUND = -2, SUCCESS = 0,
  USB_ERROR = -1, NO_DEVICE_FOUND = -2 }

Functions

- AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗dev)
- int main (int argc, char ∗∗argv)

24.184.1 Macro Definition Documentation

#define BITS_PER_BYTE 8

#define DEVICES_REQUIRED 1

#define MAX_DIO_BYTES 12

#define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)

#define MAX_NAME_SIZE 20

24.184.2 Enumeration Type Documentation

enum EXIT_CODE

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

See Also

Compilation
CmakeCompilation

Enumerator

SUCCESS
USB_ERROR
NO_DEVICE_FOUND
SUCCESS
USB_ERROR
NO_DEVICE_FOUND

24.184.3 Function Documentation

AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗ *dev*)

int main (int *argc*, char ∗∗ *argv*)

< GetDevices

24.185 samples/USB-DIO-32/sample3.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "AIOTypes.h"
#include "AIOChannelMask.h"
```

Functions

- int main (int argc, char ∗∗argv)

  LIBUSB Overview LIBUSB (http://www.libusb.org/) must be installed on the Linux box (the AIOUSB code was developed using libusb version 1.0.3).

24.185.1 Function Documentation

int main (int *argc,* char ∗∗ *argv*)

LIBUSB Overview LIBUSB (http://www.libusb.org/) must be installed on the Linux box (the AIOUSB code was developed using libusb version 1.0.3).

After installing libusb, it may also be necessary to set an environment variable so that the libusb and aiousb header files can be located:

```
export CPATH=/usr/local/include/libusb-1.0/:/usr/local/include/aiousb/
```

Once libusb is installed properly, it should be possible to compile the sample program using the simple command:

```
make
```

Alternatively, one can "manually" compile the sample program using the command:

```
g++ sample.cpp -laiousb -lusb-1.0 -o sample
```

or, to enable debug features

```
g++ -ggdb sample.cpp -laiousbdbg -lusb-1.0 -o sample
```

< GetDevices

24.186 samples/USB-DIO-48/read_and_write.c File Reference

AIOUSB sample program that writes and reads from a USB-DIO-48.

```
#include "aiocommon.h"
#include <aiousb.h>
```

Functions

- AIOUSB_BOOL find_dio (AIOUSBDevice ∗dev)
- void CHECK_RESULT (AIORET_TYPE retval, char ∗errmsg)
- int main (int argc, char ∗∗argv)

24.186.1 Detailed Description

AIOUSB sample program that writes and reads from a USB-DIO-48.

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

All the API functions that DO NOT begin "AIOUSB_" are standard API functions, largely documented in `http-://accesio.com/MANUALS/USB%20Software%20Reference.pdf`. The functions that DO begin with "-AIOUSB_" are "extended" API functions added to the Linux implementation. Source code lines in this sample program that are prefixed with the comment "/ ∗ API ∗ /" highlight calls to the AIOUSB API.

24.186.2 Function Documentation

AIOUSB_BOOL find_dio (AIOUSBDevice ∗ *dev*)

void CHECK_RESULT (AIORET_TYPE *retval,* char ∗ *errmsg*)

int main (int *argc,* char ∗∗ *argv*)

24.187 samples/USB-DIO-96/dio96_read_write.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Macros

- #define BITS_PER_BYTE 8
- #define DEVICES_REQUIRED 1
- #define MAX_DIO_BYTES 12
- #define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)
- #define MAX_NAME_SIZE 20
- #define PORT_C 1<<2
- #define PORT_B 1<<1
- #define PORT_A 1
- #define MAKE_MASK(GROUP, PORT) ((PORT << (GROUP ∗ 3)))

Functions

- AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗dev)
- char ∗ show_byte (unsigned char)
- int main (int argc, char ∗∗argv)

24.187.1 Macro Definition Documentation

#define BITS_PER_BYTE 8

#define DEVICES_REQUIRED 1

#define MAX_DIO_BYTES 12

#define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)

#define MAX_NAME_SIZE 20

#define PORT_C 1<<2

#define PORT_B 1<<1

#define PORT_A 1

#define MAKE_MASK(*GROUP, PORT*) ((PORT << (GROUP ∗ 3)))

24.187.2 Function Documentation

AIOUSB_BOOL find_dio_96 (AIOUSBDevice * *dev*)

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

See Also

Compilation
CmakeCompilation

char * show_byte (unsigned char *val*)

int main (int *argc,* char ** *argv*)

Configure Groups 1 and 3 and all ports A-C for being an output

Due to the nature of DIO_Configure using a Mask as the third argument, this logic is a bit inverted. Using DIO_Configure, you must specify a '1' in the mask where you want a Low voltage to occur, and you must specify a '0' in the mask where you want a High Voltage to occur.

```
* Write the following port patterns with 1
* indicating Off ( or low ) voltage.
*
* DIO_Configure Mask                | Output Signal (1=High,0=Low)
* ================================================================
*
* Group 1 Port A   00100100  corresponds to   11011011 Volts
*         Port B   10101010        "          01010101 Volts
*         Port C   11110000        "          00001111 Volts
*
* Group 3 Port A   00001111        "          11110000 Volts
*         Port B   00111100        "          11000011 Volts
*         Port C   11000011        "          00111100 Volts
*
*
```

< Data[3] is the start of Group 1, Port A

< Group 1 Port B

< Group 1 Port C

24.188 samples/USB-DIO-96/mytest.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Macros

- #define BITS_PER_BYTE 8
- #define DEVICES_REQUIRED 1
- #define MAX_DIO_BYTES 12
- #define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)
- #define MAX_NAME_SIZE 20
- #define PORT_C 1<<2
- #define PORT_B 1<<1
- #define PORT_A 1
- #define MAKE_MASK(GROUP, PORT) ((PORT << (GROUP ∗ 3)))

Functions

- AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗dev)
- char ∗ show_byte (unsigned char)
- int main (int argc, char ∗∗argv)

24.188.1 Macro Definition Documentation

#define BITS_PER_BYTE 8

#define DEVICES_REQUIRED 1

#define MAX_DIO_BYTES 12

#define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)

#define MAX_NAME_SIZE 20

#define PORT_C 1<<2

#define PORT_B 1<<1

#define PORT_A 1

#define MAKE_MASK(*GROUP, PORT*) ((PORT << (GROUP ∗ 3)))

24.188.2 Function Documentation

AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗ *dev*)

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

See Also

Compilation
CmakeCompilation

char∗ show_byte (unsigned char *val*)

int main (int *argc,* char ∗∗ *argv*)

Configure Groups 1 and 3 and all ports A-C for being an output

Due to the nature of DIO_Configure using a Mask as the third argument, this logic is a bit inverted. Using DIO_Configure, you must specify a '1' in the mask where you want a Low voltage to occur, and you must specify a '0' in the mask where you want a High Voltage to occur.

```
* Write the following port patterns with 1
* indicating Off ( or low ) voltage.
*
* DIO_Configure Mask                | Output Signal (1=High,0=Low)
* ================================================================
*
* Group 1 Port A   00100100  corresponds to   11011011 Volts
*         Port B   10101010        "          01010101 Volts
*         Port C   11110000        "          00001111 Volts
*
* Group 3 Port A   00001111        "          11110000 Volts
*         Port B   00111100        "          11000011 Volts
*         Port C   11000011        "          00111100 Volts
*
*
```

< Data[3] is the start of Group 1, Port A

< Group 1 Port B

< Group 1 Port C

24.189 samples/USB-DIO-96/tmp.c File Reference

```
#include <aiousb.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Macros

- #define BITS_PER_BYTE 8
- #define DEVICES_REQUIRED 1
- #define MAX_DIO_BYTES 12
- #define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)
- #define MAX_NAME_SIZE 20
- #define PORT_C 1<<2
- #define PORT_B 1<<1
- #define PORT_A 1
- #define MAKE_MASK(GROUP, PORT) ((PORT << (GROUP ∗ 3)))

Functions

- AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗dev)
- char ∗ show_byte (unsigned char)
- int main (int argc, char ∗∗argv)

24.189.1 Macro Definition Documentation

#define BITS_PER_BYTE 8

#define DEVICES_REQUIRED 1

#define MAX_DIO_BYTES 12

#define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)

#define MAX_NAME_SIZE 20

#define PORT_C 1<<2

#define PORT_B 1<<1

#define PORT_A 1

#define MAKE_MASK(*GROUP*, *PORT*) ((PORT << (GROUP * 3)))

24.189.2 Function Documentation

AIOUSB_BOOL find_dio_96 (AIOUSBDevice * *dev*)

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

See Also

Compilation
CmakeCompilation

char* show_byte (unsigned char *val*)

int main (int *argc*, char ** *argv*)

Configure Groups 1 and 3 and all ports A-C for being an output

Due to the nature of DIO_Configure using a Mask as the third argument,

```
* Write the following port patterns with 1
* indicating Off ( or low ) voltage.
*
* DIO_Configure Data                | Output Signal (1=High,0=Low)
* ============================================================
*
* Group 1 Port A   00100100  corresponds to   11011011 Volts
*         Port B   10101010        "          01010101 Volts
*         Port C   11110000        "          00001111 Volts
*
* Group 3 Port A   00001111        "          11110000 Volts
*         Port B   00111100        "          11000011 Volts
*         Port C   11000011        "          00111100 Volts
*
*
```

< Data[3] is the start of Group 1, Port A

< Group 1 Port B

< Group 1 Port C

24.190 samples/USB-DIO-96/write_sample.c File Reference

```
#include "AIOTypes.h"
#include "AIOChannelMask.h"
#include "aiousb.h"
#include "aiocommon.h"
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Data Structures

- struct DeviceInfo

Macros

- #define BITS_PER_BYTE 8
- #define DEVICES_REQUIRED 1
- #define MAX_DIO_BYTES 12
- #define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)
- #define MAX_NAME_SIZE 20

Enumerations

- enum EXIT_CODE {
  SUCCESS = 0, USB_ERROR = -1, NO_DEVICE_FOUND = -2, SUCCESS = 0,
  USB_ERROR = -1, NO_DEVICE_FOUND = -2 }

Functions

- AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗dev)
- AIOUSB_BOOL fnd (AIOUSBDevice ∗dev)
- int main (int argc, char ∗∗argv)

24.190.1 Macro Definition Documentation

#define BITS_PER_BYTE 8

#define DEVICES_REQUIRED 1

#define MAX_DIO_BYTES 12

#define MASK_BYTES ((MAX_DIO_BYTES + BITS_PER_BYTE - 1) / BITS_PER_BYTE)

#define MAX_NAME_SIZE 20

24.190.2 Enumeration Type Documentation

enum EXIT_CODE

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

See Also

Compilation
CmakeCompilation

Enumerator

SUCCESS

USB_ERROR

NO_DEVICE_FOUND

SUCCESS

USB_ERROR

NO_DEVICE_FOUND

24.190.3 Function Documentation

AIOUSB_BOOL find_dio_96 (AIOUSBDevice ∗ *dev*)

AIOUSB_BOOL fnd (AIOUSBDevice ∗ *dev*)

int main (int *argc,* char ∗∗ *argv*)

< GetDevices

24.191 samples/USB-IDIO-16_8/idio_sample.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "aiousb.h"
```

Macros

- #define RATE_LIMIT(product)

Functions

- int main (int argc, char ∗argv[])

24.191.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.191.2 Macro Definition Documentation

#define RATE_LIMIT(*product*)

Value:

```
do {                                                            \
        if( product >= USB_IIRO_16 && product <= USB_IIRO_4 )       \
            sleep(1);                                               \
    } while( 0 );                                                   \
```

24.191.3 Function Documentation

int main (int *argc,* char ∗ *argv[]*)

24.192 samples/USB-IDIO-16_8/idio_sample2.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "aiousb.h"
```

Macros

- #define RATE_LIMIT(product)

Functions

- int main (int argc, char ∗argv[])

24.192.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.192.2 Macro Definition Documentation

#define RATE_LIMIT(*product*)

Value:

```
do {                                                            \
        if( product >= USB_IIRO_16 && product <= USB_IIRO_4 )       \
            sleep(1);                                               \
    } while( 0 );                                                   \
```

24.192.3 Function Documentation

int main (int *argc,* char ∗ *argv[]*)

24.193 samples/USB-IDIO-16_8/perftest.c File Reference

```
#include "aiousb.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
```

Functions

- int main (int argc, char ∗argv[])

  $Date

24.193.1 Function Documentation

int main (int *argc,* char ∗ *argv[]*)

$Date

Format:

ad

$ $Author

Format:

an <ae>

$ $Release

Format:

h

$ Sample program to run the USB-IDIO-16

24.194 samples/USB-IIRO-16_8/iiro_sample.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include "aiousb.h"
#include "aiocommon.h"
```

Macros

- #define RATE_LIMIT(product)

Functions

- AIOUSB_BOOL find_idio (AIOUSBDevice ∗dev)
- int main (int argc, char ∗argv[])

24.194.1 Detailed Description

Author

Format:

an <ae>

Date

Format:

ad

Version

Format:

h

24.194.2 Macro Definition Documentation

#define RATE_LIMIT(***product*** **)**

Value:

```
do {                                                          \
        if( product >= USB_IIRO_16 && product <= USB_IIRO_4 )     \
            sleep(1);                                             \
    } while( 0 );                                                 \
```

24.194.3 Function Documentation

AIOUSB_BOOL find_idio (**AIOUSBDevice** ∗ ***dev*** **)**

int main (**int** ***argc,*** **char** ∗ ***argv[]*** **)**

< Call AIOUSB_Init() first

< Quickly list USB devices on the bus

24.195 /media/jdamon/Development/Documents/Projects/aiousb_patrick_mcbride_issue/AIOUSB/README.md File Reference

24.196 lib/wrappers/README.md File Reference

24.197 samples/USB-AI16-16/android/read_channels_test/README.md File Reference

24.198 samples/USB-AI16-16/android/README.md File Reference

24.199 samples/USB-AI16-16/java/extcal/README.md File Reference

24.200 samples/USB-AI16-16/java/read_channels_test/native-utils/README.md File Reference

24.201 samples/USB-AI16-16/java/read_channels_test/README.md File Reference

24.202 samples/USB-AI16-16/java/README.md File Reference

Index