

ACCES I/O Products USB Library

Generated by Doxygen 1.8.6

Tue Nov 29 2016 10:19:14

Contents

1	Main Page	1
1.1	AIOUSB driver library	1
1.1.1	Prerequisites	1
1.2	Building on Linux/ MacOS / BSD / *NIX systems	2
1.2.1	GNU make build	2
1.2.2	Build with CMake	2
1.2.3	Installation	2
1.2.4	How to build Wrapper languages	3
	CMake	3
	Regular Make system for building wrapper language support	3
2	Compiling and Installation	5
2.1	Compiling from source on Linux/MacOS X / UNIX	5
2.2	Compiling from source on Windows	5
2.3	Installing the Compiled libraries	5
3	AIOUSB C library	7
3.1	Overview	7
3.2	Other stuff	7
3.3	Compiling sample	7
4	AIOUSB C++ Class library	9
4.1	Introduction	9
4.2	Packaging	9
4.3	Deprecated	9
5	C/C++ Samples	11
5.1	Overview	11
5.2	C and C++ samples for USB based acquisition cards	11
5.3	USB-AI16-16	11
5.3.1	extcal.c	11
5.3.2	continuous_mode.c	12
5.3.3	continuous_mode_from_json_config.c	12
5.3.4	burst_test.c	12
	Overview	12
	Parts of the sample	12
	Command line parsing	12
	Create a new AIOContinuousBuf	13
	Set the device index for the AIOContinuousBuf	13
	Initialize the AIOContinuousBuf	13
	Set the Clock rate / Data Acquisition speed	13
	Start the continuous mode callback	13

Acquire data until completed	13
5.3.5 bulk_aquire_test.c	14
5.4 USB-IDIO-16 and USB-IDIO-8	14
5.4.1 idio_sample.c	14
5.4.2 idio_sample2.c	14
5.5 USB-IIRO-16 and USB-IIRO-8	14
5.5.1 iiro_sample.c	14
5.6 USB-DIO-32	15
5.6.1 Overview	15
5.6.2 C/C++ Language Sample	15
Building	15
Executing	15
5.6.3 Java Sample	16
Building	16
Executing	16
5.7 USB-DIO-16	17
5.7.1 Overview	17
5.7.2 Basic AIOUSB Sample	17
5.7.3 Building	17
5.7.4 Executing	18
5.7.5 AIOUSB Java Sample	18
Building	18
Executing	18
5.8 USB-AO16-16	19
5.8.1 Overview	19
Building	19
Executing	19
5.9 USB-DA12-8A	20
5.9.1 Overview	20
5.9.2 AIOUSB C Sample	20
Building	20
Executing	20
5.9.3 AIOUSB C++ Sample	21
Building	21
Executing	21
5.9.4 AIOUSB Java Sample	21
Building	21
Executing	22
6 Wrappers	23
6.1 Overview	23
6.2 Building Wrappers	23
7 Firmware	25
7.1 Introduction	25
7.2 Overview	25
7.3 Automatic Device Initialization	25
7.4 Copy Firmware Files To Share Directory	25
7.5 Copy Udev Rules File To System Directory	26
7.6 Troubleshooting The Udev Rules File	26

7.7	Manually Uploading Firmware to USB Devices	26
7.8	Minimum Required Files	26
8	LIBUSB Overview	29
8.1	Overview	29
8.2	Libusb Other Stuff	29
8.3	Compiling sample	29
9	README	31
10	How to run read_channels_test_java	33
11	How to run	35
12	How to run extcal.java	37
13	Native Utils	39
14	How to run read_channels_test_java	41
15	How to run	43
16	Todo List	45
17	Deprecated List	47
18	Namespace Index	49
18.1	Namespace List	49
19	Hierarchical Index	51
19.1	Class Hierarchy	51
20	Data Structure Index	53
20.1	Data Structures	53
21	File Index	57
21.1	File List	57
22	Namespace Documentation	61
22.1	AIOTUSB Namespace Reference	61
22.1.1	Function Documentation	62
	operator<<	62
	operator<<	62
	operator<<	62
	operator<<	62
23	Data Structure Documentation	63
23.1	ad_gain_pairs Struct Reference	63
23.1.1	Field Documentation	63
	gain	63
	name	63
23.2	ADCCfgBlock Struct Reference	63
23.2.1	Field Documentation	63
	device	63
	size	64
	registers	64

timeout	64
mux_settings	64
clock_rate	64
discardFirstSample	64
debug	64
testing	64
23.3 ADRange Struct Reference	64
23.3.1 Field Documentation	64
minVolts	64
range	64
23.4 AI16_DataPoint Class Reference	64
23.4.1 Detailed Description	65
23.4.2 Constructor & Destructor Documentation	65
AI16_DataPoint	65
23.4.3 Member Function Documentation	65
getChannel	65
getRange	65
getRangeText	65
isDifferentialMode	65
getCounts	66
getVolts	66
toString	66
23.4.4 Friends And Related Function Documentation	66
AI16_DataPointArray	66
AI16_DataSet	66
AnalogInputSubsystem	66
std::vector< AI16_DataPoint >	66
23.4.5 Field Documentation	66
counts	66
channel	66
range	66
differentialMode	66
23.5 AI16_DataPointArray Class Reference	66
23.5.1 Constructor & Destructor Documentation	66
AI16_DataPointArray	66
23.6 AI16_DataSet Class Reference	67
23.6.1 Detailed Description	67
23.6.2 Constructor & Destructor Documentation	67
AI16_DataSet	67
~AI16_DataSet	67
23.6.3 Member Function Documentation	68
getSubsystem	68
getPoints	68
getTimeStamp	68
getCalMode	68
getTriggerMode	68
getOverSample	68
isDiscardFirstSample	69
print	69

23.6.4	Friends And Related Function Documentation	69
	AnalogInputSubsystem	69
23.6.5	Field Documentation	69
	subsystem	69
	points	69
	timeStamp	69
	calMode	69
	triggerMode	69
	overSample	69
	discardFirstSample	69
23.7	AI16_InputRange Class Reference	69
23.7.1	Constructor & Destructor Documentation	70
	AI16_InputRange	70
	AI16_InputRange	70
	~AI16_InputRange	70
23.7.2	Member Function Documentation	70
	setRange	70
23.7.3	Friends And Related Function Documentation	70
	AnalogInputSubsystem	70
	AI16_DataPoint	70
23.8	aio_channel_range Struct Reference	70
23.8.1	Field Documentation	70
	start	70
	end	70
	gain	70
23.9	aio_counts_converter Struct Reference	71
23.9.1	Field Documentation	71
	num_oversamples	71
	num_channels	71
	num_scans	71
	unit_size	71
	scan_count	71
	channel_count	71
	os_count	71
	converted_count	71
	sum	71
	buf	71
	continue_conversion	71
	gain_ranges	71
	Convert	71
	ConvertFifo	71
	discardFirstSample	71
23.10	aio_either_val Struct Reference	71
23.10.1	Field Documentation	72
	number	72
	object	72
23.11	aio_ret_value Struct Reference	72
23.11.1	Field Documentation	72
	left	72

errmsg	72
right	72
type	72
size	72
23.12AIOArgument Struct Reference	72
23.12.1 Field Documentation	72
threaded	72
debug	72
size	72
actual_size	72
config	72
23.13AIOArguments Struct Reference	73
23.13.1 Field Documentation	73
device_args	73
number_arguments	73
23.14AIOBuf Struct Reference	73
23.14.1 Field Documentation	73
size	73
endpos	73
type	73
defined	73
23.15aiobuf_iterator Struct Reference	73
23.15.1 Field Documentation	73
buf	73
loc	73
next	73
23.16AIOChannelMask Struct Reference	74
23.16.1 Field Documentation	74
signal_indices	74
signal_index	74
active_signals	74
signals	74
number_signals	74
pos	74
size	74
strrep	74
strrepsmall	74
23.17AIOChannelRangeTmp Struct Reference	74
23.17.1 Field Documentation	74
start_channel	74
end_channel	74
gaincode	74
23.18AIOCmd Struct Reference	75
23.18.1 Field Documentation	75
stop_scan	75
stop_scan_arg	75
channel	75
num_scans	75
num_channels	75

num_samples	75
23.19AIOCommandLineOptions Struct Reference	75
23.19.1 Field Documentation	76
pass_through	76
num_scans	76
default_num_scans	76
num_channels	76
default_num_channels	76
num_oversamples	76
default_num_oversamples	76
gain_code	76
clock_rate	76
default_clock_rate	76
outfile	76
reset	76
debug_level	76
number_ranges	76
verbose	76
start_channel	76
default_start_channel	76
end_channel	76
default_end_channel	76
index	76
block_size	76
with_timing	76
slow_acquire	76
buffer_size	76
rate_limit	76
physical	76
counts	76
calibration	76
repeat_number	76
aiobuf_json	76
default_aiobuf_json	77
adcconfig_json	77
ranges	77
23.20AIOContinuousBuf Struct Reference	77
23.20.1 Detailed Description	77
23.20.2 Field Documentation	78
callback	78
worker	78
lock	78
tattr	78
work	78
DeviceIndex	78
fifo	78
buffer	78
countsbuf	78
unit_size	78

hz	78
base_size	78
size	78
num_oversamples	78
num_channels	78
num_scans	78
scans_read	78
start_scanning	78
block_size	78
bytes_processed	78
counter_control	78
timeout	78
exitcode	78
testing	78
debug	78
mask	78
status	78
type	78
PushN	78
PopN	78
23.21 AIODeviceInfo Struct Reference	79
23.21.1 Field Documentation	79
PID	79
NameSize	79
Name	79
DIOBytes	79
Counters	79
23.22 AIODeviceQuery Struct Reference	79
23.22.1 Field Documentation	79
productID	79
nameSize	79
name	79
numDIOBytes	80
numCounters	80
index	80
23.23 aioerror Struct Reference	80
23.23.1 Field Documentation	80
retval	80
error_message	80
23.24 AIOFifo Struct Reference	80
23.24.1 Detailed Description	80
23.24.2 Field Documentation	80
AIO_FIFO_INTERFACE	80
LOCKING_MECHANISM	81
23.25 AIOGainRange Struct Reference	81
23.25.1 Field Documentation	81
min	81
max	81
23.26 AIOProductGroup Struct Reference	81

23.26.1 Detailed Description	81
23.27 AIOPProductRange Struct Reference	81
23.27.1 Detailed Description	81
23.28 aioub _libusb_args Struct Reference	81
23.28.1 Field Documentation	82
dev	82
handle	82
deviceDesc	82
23.29 AIIODevice Struct Reference	82
23.29.1 Field Documentation	83
usb_device	83
bOpen	83
deviceIndex	83
isInit	83
PID	83
DIOConfigBits	83
discardFirstSample	83
commTimeout	83
miscClockHz	83
ProductID	83
DIOBytes	83
Counters	83
Tristates	83
bGateSelectable	83
RootClock	83
bGetName	84
ConfigBytes	84
ImmDACs	84
bDACStream	84
bDACDIOSStream	84
bDACSlowWaveStream	84
bDACDIOClock	84
DACsUsed	84
bADCStream	84
ADCChannels	84
ADCMUXChannels	84
RangeShift	84
ADCChannelsPerGroup	84
bDIOSStream	84
StreamingBlockSize	84
bDIODebounce	84
bDIOSPI	84
bSetCustomClocks	84
WDGBytes	84
bClearFIFO	84
ImmADCs	84
bDACBoardRange	84
bDACChannelCal	84
FlashSectors	84

bDACOpen	84
bDACClosing	84
bDACAborting	84
bDACStarted	84
DACData	84
PendingDACData	84
hDACDataMutex	84
hDACDataSem	84
bDIOOpen	84
bDIORead	85
bDeviceWasHere	85
LastDIOData	85
cachedName	85
cachedSerialNumber	85
cachedConfigBlock	85
workerBusy	85
workerStatus	85
workerResult	85
FastITConfig	85
FastITBakConfig	85
FastITConfig_size	85
ADBuf	85
ADBuf_size	85
testing	85
valid	85
23.30aiousboption Struct Reference	85
23.31AIOWDGConfig Struct Reference	85
23.31.1 Field Documentation	86
bufsize	86
L	86
wdgbuf	86
timeout	86
23.32AnalogInputSubsystem Class Reference	86
23.32.1 Detailed Description	89
23.32.2 Constructor & Destructor Documentation	89
AnalogInputSubsystem	89
~AnalogInputSubsystem	89
23.32.3 Member Function Documentation	89
setScanRange	89
print	89
getNumChannels	90
getNumMUXChannels	90
getChannelsPerGroup	90
isAutoCalPresent	90
getRangeText	90
isAutoConfig	91
setAutoConfig	91
readConfig	91
writeConfig	91

isDiscardFirstSample	92
setDiscardFirstSample	92
getCalMode	92
setCalMode	92
getTriggerMode	93
setTriggerMode	93
getRange	93
getRange	93
setRange	94
setRange	94
isDifferentialMode	94
isDifferentialMode	94
setDifferentialMode	95
setDifferentialMode	95
setRangeAndDiffMode	96
setRangeAndDiffMode	96
setRangeAndDiffMode	96
getOverSample	96
setOverSample	96
setCalibrationTable	97
setCalibrationTable	97
getStreamingBlockSize	97
setStreamingBlockSize	97
getClock	98
setClock	98
calibrate	98
calibrate	98
read	99
readCounts	99
readCounts	99
readVolts	101
readVolts	101
readBulkStart	101
readBulkSamplesAvailable	102
readBulkNext	102
clearFIFO	102
countsToVolts	102
countsToVolts	103
voltsToCounts	103
voltsToCounts	103
23.32.4 Friends And Related Function Documentation	103
USB_AI16_Family	103
23.32.5 Field Documentation	104
CAL_MODE_NORMAL	104
CAL_MODE_GROUND	104
CAL_MODE_REFERENCE	104
TRIG_MODE_CTR0_EXT	104
TRIG_MODE_FALLING_EDGE	104
TRIG_MODE_SCAN	104

TRIG_MODE_EXTERNAL	104
TRIG_MODE_TIMER	104
RANGE_0_10V	104
RANGE_10V	104
RANGE_0_5V	104
RANGE_5V	104
RANGE_0_2V	104
RANGE_2V	104
RANGE_0_1V	105
RANGE_1V	105
MIN_COUNTS	105
MAX_COUNTS	105
CAL_TABLE_WORDS	105
RANGE_TEXT	105
NUM_CONFIG_REGISTERS	105
NUM_MUX_CONFIG_REGISTERS	105
NUM_GAIN_CODE_REGISTERS	105
REG_GAIN_CODE	105
REG_CAL_MODE	105
REG_TRIG_MODE	105
REG_START_END	105
REG_OVERSAMPLE	105
REG_MUX_START_END	105
DIFFERENTIAL_MODE	105
MAX_OVERSAMPLE	105
TRIG_MODE_VALID_MASK	105
AUTO_CAL_UNKNOWN	106
AUTO_CAL_NOT_PRESENT	106
AUTO_CAL_PRESENT	106
MAX_CHANNELS	106
numChannels	106
numMUXChannels	106
channelsPerGroup	106
configBlockSize	106
autoCalFeature	106
inputRange	106
differentialMode	106
calMode	106
triggerMode	106
startChannel	106
endChannel	106
overSample	106
readBulkBuffer	106
readBulkSamplesRequested	106
readBulkSamplesRetrieved	106
autoConfig	106
23.33AnalogIORange Class Reference	106
23.33.1 Detailed Description	107
23.33.2 Constructor & Destructor Documentation	107

AnalogIORange	107
AnalogIORange	107
~AnalogIORange	107
23.33.3 Member Function Documentation	107
getRange	107
setRange	107
setCountRange	108
setVoltRange	109
countsToVolts	109
voltsToCounts	109
23.33.4 Field Documentation	109
range	109
minCounts	109
maxCounts	109
rangeCounts	109
minVolts	109
maxVolts	109
rangeVolts	110
23.34 AnalogOutputSubsystem Class Reference	110
23.34.1 Detailed Description	110
23.34.2 Constructor & Destructor Documentation	110
AnalogOutputSubsystem	110
~AnalogOutputSubsystem	110
23.34.3 Member Function Documentation	110
print	110
getNumChannels	111
writeCounts	111
writeCounts	111
23.34.4 Field Documentation	111
numChannels	111
minCounts	111
maxCounts	111
23.35 AO16_AnalogOutputSubsystem Class Reference	111
23.35.1 Detailed Description	112
23.35.2 Constructor & Destructor Documentation	113
AO16_AnalogOutputSubsystem	113
~AO16_AnalogOutputSubsystem	113
23.35.3 Member Function Documentation	113
getRangeText	113
getRange	113
setRange	113
writeVolts	113
writeVolts	114
countsToVolts	115
voltsToCounts	115
23.35.4 Friends And Related Function Documentation	115
USB_AO16_Family	115
23.35.5 Field Documentation	115
RANGE_0_5V	115

RANGE_5V	115
RANGE_0_10V	115
RANGE_10V	115
MIN_COUNTS	115
MAX_COUNTS	115
RANGE_TEXT	116
outputRange	116
23.36AO16_OutputRange Class Reference	116
23.36.1 Constructor & Destructor Documentation	116
AO16_OutputRange	116
AO16_OutputRange	116
~AO16_OutputRange	116
23.36.2 Member Function Documentation	116
setRange	116
23.36.3 Friends And Related Function Documentation	117
AO16_AnalogOutputSubsystem	117
23.37BoolArray Class Reference	117
23.37.1 Constructor & Destructor Documentation	117
BoolArray	117
23.38BulkAcquireWorkerParams Struct Reference	117
23.38.1 Field Documentation	117
DeviceIndex	117
BufSize	117
pBuf	117
23.39channel_range Struct Reference	117
23.39.1 Field Documentation	118
start_channel	118
end_channel	118
gaincode	118
23.40cJSON Struct Reference	118
23.40.1 Field Documentation	118
next	118
prev	118
child	118
type	118
valuestring	118
valueint	118
valuedouble	118
string	118
23.41cJSON_Hooks Struct Reference	118
23.41.1 Field Documentation	118
malloc_fn	118
free_fn	118
23.42config_options Struct Reference	119
23.42.1 Field Documentation	119
targetSerialNumber	119
framePoints	119
buffer_size	119
clock_rate	119

number_channels	119
write_clock_rate	119
23.43configuration Struct Reference	119
23.43.1 Field Documentation	119
type	119
timeout	119
discard_first_sample	119
device_index	119
number_scans	119
calibration	120
scan_type	120
calibration_file	120
debug	120
output_file	120
file_handle	120
file_name	120
configure	120
run	120
23.44Counter Class Reference	120
23.44.1 Detailed Description	121
23.44.2 Constructor & Destructor Documentation	121
Counter	121
23.44.3 Member Function Documentation	121
getDeviceIndex	121
setMode	121
setCount	121
setModeAndCount	121
readCount	122
readCountAndStatus	122
readCountAndSetModeAndCount	122
23.44.4 Friends And Related Function Documentation	122
CounterSubsystem	122
23.44.5 Field Documentation	122
MODE_TERMINAL_COUNT	122
MODE_ONE_SHOT	122
MODE_RATE_GENERATOR	123
MODE_SQUARE_WAVE	123
MODE_SW_TRIGGERED	123
MODE_HW_TRIGGERED	123
parent	123
counterIndex	123
23.45CounterList Class Reference	123
23.46CounterSubsystem Class Reference	123
23.46.1 Detailed Description	124
23.46.2 Constructor & Destructor Documentation	124
CounterSubsystem	124
~CounterSubsystem	124
23.46.3 Member Function Documentation	124
print	124

getNumCounterBlocks	124
getNumCounters	124
getCounter	125
readCounts	125
selectGate	125
startClock	125
stopClock	126
23.46.4 Friends And Related Function Documentation	126
Counter	126
USB_AI16_Family	126
USB_CTR_15_Family	126
USB_DIO_32_Family	126
23.46.5 Field Documentation	126
COUNTERS_PER_BLOCK	126
numCounterBlocks	126
numCounters	126
counters	126
23.47CStringArray Struct Reference	126
23.48DA12_AnalogOutputSubsystem Class Reference	127
23.48.1 Detailed Description	128
23.48.2 Constructor & Destructor Documentation	128
DA12_AnalogOutputSubsystem	128
~DA12_AnalogOutputSubsystem	128
23.48.3 Member Function Documentation	128
getRangeText	128
getRange	128
getRange	128
setRange	129
setRange	129
setRange	129
writeVolts	130
writeVolts	131
countsToVolts	131
voltsToCounts	131
23.48.4 Friends And Related Function Documentation	131
USB_DA12_8A_Family	131
USB_DA12_8E_Family	131
23.48.5 Field Documentation	131
RANGE_0_2_5V	131
RANGE_2_5V	132
RANGE_0_5V	132
RANGE_5V	132
RANGE_0_10V	132
RANGE_10V	132
MIN_COUNTS	132
MAX_COUNTS	132
RANGE_TEXT	132
outputRange	132
23.49DA12_OutputRange Class Reference	132

23.49.1 Constructor & Destructor Documentation	133
DA12_OutputRange	133
DA12_OutputRange	133
~DA12_OutputRange	133
23.49.2 Member Function Documentation	133
setRange	133
23.49.3 Friends And Related Function Documentation	133
DA12_AnalogOutputSubsystem	133
23.50 DeviceInfo Struct Reference	133
23.50.1 Field Documentation	133
outputMask	133
readBuffer	133
writeBuffer	133
name	133
productID	133
nameSize	134
numDIOBytes	134
numCounters	134
serialNumber	134
index	134
23.51 DeviceProperties Struct Reference	134
23.51.1 Detailed Description	134
23.51.2 Field Documentation	134
Name	134
SerialNumber	134
ProductID	134
DIOPorts	135
Counters	135
Tristates	135
RootClock	135
DACChannels	135
ADCChannels	135
ADCMUXChannels	135
ADCChannelsPerGroup	135
23.52 DeviceSubsystem Class Reference	135
23.52.1 Detailed Description	135
23.52.2 Constructor & Destructor Documentation	136
DeviceSubsystem	136
~DeviceSubsystem	136
23.52.3 Member Function Documentation	136
getDeviceIndex	136
print	136
getParent	136
23.52.4 Field Documentation	136
parent	136
23.53 DigitalIOSubsystem Class Reference	136
23.53.1 Detailed Description	137
23.53.2 Constructor & Destructor Documentation	137
DigitalIOSubsystem	137

~DigitalIOSubsystem	137
23.53.3 Member Function Documentation	137
bitsToBytes	137
bytesToBits	137
print	137
getNumPorts	137
getNumChannels	138
getNumTristateGroups	138
getNumTristateChannels	138
configure	138
configure	138
getConfiguration	139
read	139
read	139
write	140
write	140
23.53.4 Friends And Related Function Documentation	140
USB_AI16_Family	140
USB_AO16_Family	140
USB_DIO_Family	140
USB_DIO_16_Family	140
USB_DIO_32_Family	140
23.53.5 Field Documentation	140
numPorts	140
numChannels	140
numTristateGroups	140
numTristateChannels	140
writeValues	141
23.54DIOBuf Struct Reference	141
23.54.1 Detailed Description	141
23.54.2 Field Documentation	141
size	141
buffer	141
strbuf	141
strbuf_size	141
23.55DIOStreamSubsystem Class Reference	142
23.55.1 Detailed Description	142
23.55.2 Constructor & Destructor Documentation	142
DIOStreamSubsystem	142
~DIOStreamSubsystem	142
23.55.3 Member Function Documentation	142
print	142
getStreamingBlockSize	143
setStreamingBlockSize	143
getClock	143
setClock	143
stopClock	144
open	144
close	144

read	144
write	144
clearFIFO	145
23.55.4 Friends And Related Function Documentation	145
USB_DIO_16_Family	145
23.55.5 Field Documentation	145
clockHz	145
23.56DoubleArray Class Reference	145
23.56.1 Constructor & Destructor Documentation	145
DoubleArray	145
23.57Error Class Reference	145
23.57.1 Constructor & Destructor Documentation	146
Error	146
Error	146
23.57.2 Member Function Documentation	146
what	146
23.57.3 Field Documentation	146
message	146
23.58IllegalArgumentException Class Reference	146
23.58.1 Detailed Description	146
23.58.2 Constructor & Destructor Documentation	146
IllegalArgumentException	146
23.59IntArray Class Reference	146
23.59.1 Constructor & Destructor Documentation	146
IntArray	146
23.60lookup Struct Reference	147
23.60.1 Field Documentation	147
value	147
str	147
strvalue	147
23.61mux_settings Struct Reference	147
23.61.1 Field Documentation	147
ADCChannelsPerGroup	147
ADCMUXChannels	147
defined	147
23.62new_aio_fifo Struct Reference	147
23.62.1 Field Documentation	148
AIO_FIFO_INTERFACE	148
LOCKING_MECHANISM	148
Push	148
PushN	148
Pop	148
PopN	148
23.63OperationFailedException Class Reference	148
23.63.1 Detailed Description	148
23.63.2 Constructor & Destructor Documentation	148
OperationFailedException	148
OperationFailedException	148
23.64options Struct Reference	148

23.64.1 Field Documentation	149
maxcount	149
use_maxcount	149
23.65opts Struct Reference	149
23.65.1 Field Documentation	149
num_scans	149
default_num_scans	149
num_channels	149
default_num_channels	149
num_oversamples	150
default_num_oversamples	150
gain_code	150
clock_rate	150
default_clock_rate	150
outfile	150
reset	150
debug_level	150
number_ranges	150
verbose	150
start_channel	150
default_start_channel	150
end_channel	150
default_end_channel	150
index	150
block_size	150
with_timing	150
slow_acquire	150
buffer_size	150
rate_limit	150
physical	150
counts	150
calibration	150
repeat	150
aiobuf_json	150
default_aiobuf_json	150
adcconfig_json	150
ranges	150
num_scans	150
clock_speed	150
cal_channel	150
max_channels	150
clock_scale	150
calibration_enabled	150
23.66OutputVoltagePoint Class Reference	151
23.66.1 Detailed Description	151
23.66.2 Constructor & Destructor Documentation	151
OutputVoltagePoint	151
OutputVoltagePoint	151
23.66.3 Field Documentation	151

channel	151
volts	151
23.67OutputVoltagePointArray Class Reference	151
23.67.1 Constructor & Destructor Documentation	152
OutputVoltagePointArray	152
23.68ProductIDName Struct Reference	152
23.68.1 Field Documentation	152
id	152
name	152
23.69rangelookup Struct Reference	152
23.69.1 Field Documentation	152
minvalue	152
maxvalue	152
23.70StringArray Class Reference	152
23.70.1 Constructor & Destructor Documentation	152
StringArray	152
23.71TestCaseSetup Class Reference	153
23.71.1 Constructor & Destructor Documentation	154
~TestCaseSetup	154
TestCaseSetup	154
TestCaseSetup	154
23.71.2 Member Function Documentation	154
findDevice	154
findDevice	154
doSomething	154
setCurrentDeviceIndex	154
doBulkConfigBlock	154
doPreSetup	154
doSetAutoCalibration	154
doVerifyGroundCalibration	154
doVerifyReferenceCalibration	154
doDemonstrateReadVoltages	155
doScanSingleChannel	155
doPreReadImmediateVoltages	155
doCSVReadVoltages	155
doCSVWithGetChannelV	155
doCleanupAfterBulk	155
doDACDirect	155
doDACDirectSetup	155
writeBuffer	155
setMaxCount	155
ThrowError	155
doFastITScanSetup	155
doFastITScan	155
doGetBuffer	155
THROW_IF_ERROR	155
envGetInteger	155
envGetDouble	155
doTestSetAutoCalibration	155

doGenericVendorWrite	155
doBulkAcquire	155
doBulkAcquire	155
doDisplayBulkResults	156
resetCPU	156
getVolts	156
getCounts	156
getGainCodes	156
setupVoltageParameters	156
TEST_ADC_BulkPoll	156
23.71.3 Field Documentation	156
productID	156
nameSize	156
numDIOBytes	156
numCounters	156
DeviceIndex	156
deviceFound	156
CAL_CHANNEL	156
MAX_CHANNELS	156
NUM_CHANNELS	156
counts	156
volts	156
gainCodes	156
configBlock	156
number_oversamples	156
block_size	156
clock_speed	156
maxcounts	156
calibration_enabled	156
dataBuf	156
23.72UCharArray Class Reference	157
23.72.1 Constructor & Destructor Documentation	157
UCharArray	157
23.73USB_AI16_Family Class Reference	157
23.73.1 Detailed Description	158
23.73.2 Constructor & Destructor Documentation	158
USB_AI16_Family	158
~USB_AI16_Family	158
23.73.3 Member Function Documentation	158
initialize	158
getSupportedProductNames	158
getSupportedProductIDs	158
isSupportedProductID	158
print	158
adc	159
dio	159
ctr	159
23.73.4 Friends And Related Function Documentation	159
USBDeviceManager	159

23.73.5 Field Documentation	159
supportedProductIDs	159
analogInputSubsystem	159
digitalIOSubsystem	159
counterSubsystem	159
23.74USB_AIO16_Family Class Reference	159
23.74.1 Detailed Description	160
23.74.2 Constructor & Destructor Documentation	161
USB_AIO16_Family	161
~USB_AIO16_Family	161
23.74.3 Member Function Documentation	161
initialize	161
getSupportedProductNames	161
getSupportedProductIDs	161
isSupportedProductID	161
print	161
adc	161
dac	162
dio	162
ctr	162
23.74.4 Friends And Related Function Documentation	162
USBDeviceManager	162
23.74.5 Field Documentation	162
supportedProductIDs	162
analogInputSubsystem	162
analogOutputSubsystem	162
digitalIOSubsystem	162
counterSubsystem	162
23.75USB_AO16_Family Class Reference	162
23.75.1 Detailed Description	163
23.75.2 Constructor & Destructor Documentation	163
USB_AO16_Family	163
~USB_AO16_Family	163
23.75.3 Member Function Documentation	163
initialize	163
getSupportedProductNames	163
getSupportedProductIDs	164
isSupportedProductID	164
print	164
dac	164
dio	164
23.75.4 Friends And Related Function Documentation	164
USBDeviceManager	164
23.75.5 Field Documentation	164
supportedProductIDs	164
analogOutputSubsystem	164
digitalIOSubsystem	164
23.76USB_CTR_15_Family Class Reference	165
23.76.1 Detailed Description	165

23.76.2 Constructor & Destructor Documentation	166
USB_CTR_15_Family	166
~USB_CTR_15_Family	166
23.76.3 Member Function Documentation	166
initialize	166
getSupportedProductNames	166
getSupportedProductIDs	166
isSupportedProductID	166
print	166
ctr	166
23.76.4 Friends And Related Function Documentation	167
USBDeviceManager	167
23.76.5 Field Documentation	167
supportedProductIDs	167
counterSubsystem	167
23.77USB_DA12_8A_Family Class Reference	167
23.77.1 Detailed Description	168
23.77.2 Constructor & Destructor Documentation	168
USB_DA12_8A_Family	168
~USB_DA12_8A_Family	168
23.77.3 Member Function Documentation	168
initialize	168
getSupportedProductNames	168
getSupportedProductIDs	168
isSupportedProductID	168
print	168
dac	169
23.77.4 Friends And Related Function Documentation	169
USBDeviceManager	169
23.77.5 Field Documentation	169
supportedProductIDs	169
analogOutputSubsystem	169
23.78USB_DA12_8E_Family Class Reference	169
23.78.1 Detailed Description	170
23.78.2 Constructor & Destructor Documentation	170
USB_DA12_8E_Family	170
~USB_DA12_8E_Family	170
23.78.3 Member Function Documentation	170
initialize	170
getSupportedProductNames	170
getSupportedProductIDs	170
isSupportedProductID	170
print	171
dac	171
23.78.4 Friends And Related Function Documentation	171
USBDeviceManager	171
23.78.5 Field Documentation	171
supportedProductIDs	171
analogOutputSubsystem	171

23.79USB_DIO_16_Family Class Reference	171
23.79.1 Detailed Description	172
23.79.2 Constructor & Destructor Documentation	172
USB_DIO_16_Family	172
~USB_DIO_16_Family	172
23.79.3 Member Function Documentation	172
initialize	172
getSupportedProductNames	172
getSupportedProductIDs	173
isSupportedProductID	173
print	173
dio	173
diostream	173
23.79.4 Friends And Related Function Documentation	173
USBDeviceManager	173
23.79.5 Field Documentation	173
supportedProductIDs	173
digitalIOSubsystem	173
dioStreamSubsystem	173
23.80USB_DIO_32_Family Class Reference	174
23.80.1 Detailed Description	174
23.80.2 Constructor & Destructor Documentation	175
USB_DIO_32_Family	175
~USB_DIO_32_Family	175
23.80.3 Member Function Documentation	175
initialize	175
getSupportedProductNames	175
getSupportedProductIDs	175
isSupportedProductID	175
print	175
dio	175
ctr	176
23.80.4 Friends And Related Function Documentation	176
USBDeviceManager	176
23.80.5 Field Documentation	176
supportedProductIDs	176
digitalIOSubsystem	176
counterSubsystem	176
23.81USB_DIO_Family Class Reference	176
23.81.1 Detailed Description	177
23.81.2 Constructor & Destructor Documentation	177
USB_DIO_Family	177
~USB_DIO_Family	177
23.81.3 Member Function Documentation	177
initialize	177
getSupportedProductNames	177
getSupportedProductIDs	177
isSupportedProductID	177
print	178

dio	178
23.81.4 Friends And Related Function Documentation	178
USBDeviceManager	178
23.81.5 Field Documentation	178
supportedProductIDs	178
digitalIOSubsystem	178
23.82USBDevice Struct Reference	178
23.82.1 Field Documentation	179
usb_control_transfer	179
usb_bulk_transfer	179
usb_request	179
usb_reset_device	179
usb_put_config	179
usb_get_config	179
timeout	179
device	179
deviceHandle	179
deviceDesc	179
debug	179
usblp_attached	179
iface	179
verbose	179
conf	179
origconf	179
altset	179
23.83USBDeviceArray Class Reference	179
23.83.1 Constructor & Destructor Documentation	179
USBDeviceArray	179
23.84USBDeviceBase Class Reference	179
23.84.1 Detailed Description	180
23.84.2 Constructor & Destructor Documentation	181
USBDeviceBase	181
~USBDeviceBase	181
23.84.3 Member Function Documentation	181
clearFIFO	181
getMiscClock	181
setMiscClock	181
getStreamingBlockSize	181
setStreamingBlockSize	181
print	181
getDeviceIndex	181
getProductID	181
getName	181
getSerialNumber	181
getCommTimeout	182
setCommTimeout	182
reset	182
customEEPROMWrite	182
customEEPROMRead	183

23.84.4 Friends And Related Function Documentation	183
USBDeviceManager	183
DIOStreamSubsystem	183
AnalogInputSubsystem	183
23.84.5 Field Documentation	183
CUSTOM_EEPROM_SIZE	183
CLEAR_FIFO_METHOD_IMMEDIATE	183
CLEAR_FIFO_METHOD_AUTO	183
CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT	183
CLEAR_FIFO_METHOD_WAIT	183
deviceIndex	183
productID	183
name	183
serialNumber	183
23.85USBDeviceManager Class Reference	184
23.85.1 Detailed Description	185
23.85.2 Constructor & Destructor Documentation	185
USBDeviceManager	185
~USBDeviceManager	185
23.85.3 Member Function Documentation	185
emptyDeviceList	185
print	185
printDevices	185
getAIOUSBVersion	185
getAIOUSBVersionDate	185
productIDToName	186
productIDToName	186
productNameToID	186
productNameToID	187
listDevices	187
getResultCodeAsString	187
isOpen	187
open	187
close	188
scanForDevices	188
getDeviceByProductID	188
getDeviceByProductID	188
getDeviceByProductID	189
getDeviceBySerialNumber	189
23.85.4 Field Documentation	189
VERSION_NUMBER	189
VERSION_DATE	189
MIN_PRODUCT_ID	189
MAX_PRODUCT_ID	189
deviceList	189
OPEN_PATTERN	189
openStatus	189
MESSAGE_NOT_OPEN	189
23.86ushort_array Struct Reference	190

23.86.1 Field Documentation	190
size	190
23.87UShortArray Class Reference	190
23.87.1 Constructor & Destructor Documentation	190
UShortArray	190
24 File Documentation	191
24.1 deprecated/classlib/AI16_DataPoint.cpp File Reference	191
24.2 deprecated/classlib/AI16_DataPoint.hpp File Reference	191
24.3 deprecated/classlib/AI16_DataSet.cpp File Reference	191
24.4 deprecated/classlib/AI16_DataSet.hpp File Reference	191
24.5 deprecated/classlib/AI16_InputRange.cpp File Reference	192
24.6 deprecated/classlib/AI16_InputRange.hpp File Reference	192
24.7 deprecated/classlib/AnalogInputSubsystem.cpp File Reference	192
24.7.1 Detailed Description	192
24.8 deprecated/classlib/AnalogInputSubsystem.hpp File Reference	193
24.9 deprecated/classlib/AnalogIORange.cpp File Reference	193
24.10deprecated/classlib/AnalogIORange.hpp File Reference	193
24.11deprecated/classlib/AnalogOutputSubsystem.cpp File Reference	193
24.12deprecated/classlib/AnalogOutputSubsystem.hpp File Reference	194
24.13deprecated/classlib/AO16_AnalogOutputSubsystem.cpp File Reference	194
24.14deprecated/classlib/AO16_AnalogOutputSubsystem.hpp File Reference	194
24.15deprecated/classlib/AO16_OutputRange.cpp File Reference	194
24.16deprecated/classlib/AO16_OutputRange.hpp File Reference	194
24.17deprecated/classlib/Counter.cpp File Reference	195
24.17.1 Detailed Description	195
24.18deprecated/classlib/Counter.hpp File Reference	195
24.19deprecated/classlib/CounterSubsystem.cpp File Reference	196
24.19.1 Detailed Description	196
24.20deprecated/classlib/CounterSubsystem.hpp File Reference	196
24.21deprecated/classlib/CppCommon.h File Reference	196
24.22deprecated/classlib/DA12_AnalogOutputSubsystem.cpp File Reference	196
24.22.1 Detailed Description	197
24.23deprecated/classlib/DA12_AnalogOutputSubsystem.hpp File Reference	197
24.24deprecated/classlib/DA12_OutputRange.cpp File Reference	197
24.25deprecated/classlib/DA12_OutputRange.hpp File Reference	197
24.26deprecated/classlib/DeviceSubsystem.cpp File Reference	198
24.27deprecated/classlib/DeviceSubsystem.hpp File Reference	198
24.28deprecated/classlib/DigitalIOSubsystem.cpp File Reference	198
24.28.1 Detailed Description	198
24.29deprecated/classlib/DigitalIOSubsystem.hpp File Reference	198
24.30deprecated/classlib/DIOStreamSubsystem.cpp File Reference	199
24.30.1 Detailed Description	199
24.31deprecated/classlib/DIOStreamSubsystem.hpp File Reference	199
24.32deprecated/classlib/OutputVoltagePoint.hpp File Reference	199
24.33deprecated/classlib/README.doc File Reference	200
24.34lib/wrappers/README.doc File Reference	200
24.35Firmware/README.doc File Reference	200
24.36samples/USB-AI16-16/README.doc File Reference	200
24.37samples/USB-AO16-16/README.doc File Reference	200

24.38samples/USB-DA12-8A/README.doc File Reference 200

24.39samples/USB-DIO-16/README.doc File Reference 200

24.40samples/USB-DIO-32/README.doc File Reference 200

24.41samples/USB-IDIO-16_8/README.doc File Reference 200

24.42samples/USB-IIRO-16_8/README.doc File Reference 200

24.43deprecated/classlib/USB_AI16_Family.cpp File Reference 200

24.44deprecated/classlib/USB_AI16_Family.hpp File Reference 200

24.45deprecated/classlib/USB_AIO16_Family.cpp File Reference 201

24.46deprecated/classlib/USB_AIO16_Family.hpp File Reference 201

24.47deprecated/classlib/USB_AO16_Family.cpp File Reference 201

24.48deprecated/classlib/USB_AO16_Family.hpp File Reference 201

24.49deprecated/classlib/USB_CTR_15_Family.cpp File Reference 202

24.50deprecated/classlib/USB_CTR_15_Family.hpp File Reference 202

24.51deprecated/classlib/USB_DA12_8A_Family.cpp File Reference 202

24.52deprecated/classlib/USB_DA12_8A_Family.hpp File Reference 202

24.53deprecated/classlib/USB_DA12_8E_Family.cpp File Reference 203

24.54deprecated/classlib/USB_DA12_8E_Family.hpp File Reference 203

24.55deprecated/classlib/USB_DIO_16_Family.cpp File Reference 203

24.56deprecated/classlib/USB_DIO_16_Family.hpp File Reference 203

24.57deprecated/classlib/USB_DIO_32_Family.cpp File Reference 204

24.58deprecated/classlib/USB_DIO_32_Family.hpp File Reference 204

24.59deprecated/classlib/USB_DIO_Family.cpp File Reference 204

24.60deprecated/classlib/USB_DIO_Family.hpp File Reference 204

24.61deprecated/classlib/USBDeviceBase.cpp File Reference 205

24.62deprecated/classlib/USBDeviceBase.hpp File Reference 205

24.63deprecated/classlib/USBDeviceManager.cpp File Reference 206

 24.63.1 Detailed Description 206

24.64deprecated/classlib/USBDeviceManager.hpp File Reference 206

 24.64.1 Detailed Description 207

24.65doc/aiousb.doc File Reference 207

24.66doc/firmware.doc File Reference 207

24.67doc/index.doc File Reference 207

24.68doc/install.doc File Reference 207

24.69doc/java.doc File Reference 207

24.70doc/libusb.doc File Reference 207

24.71doc/samples.doc File Reference 207

24.72doc/wrappers.doc File Reference 207

24.73lib/ADCConfigBlock.c File Reference 207

 24.73.1 Function Documentation 208

 ADCConfigBlockCopy 208

 DeleteADCConfigBlock 208

 ADCConfigBlockGetAIOUSBDevice 208

 ADCConfigBlockSetAIOUSBDevice 208

 ADCConfigBlockSetDevice 208

 ADCConfigBlockInitializeDefault 208

 ADCConfigBlockInitializeFromAIOUSBDevice 208

 ADCConfigBlockSetSize 209

 ADCConfigBlockGetSize 209

 ADCConfigBlockSetTesting 209

ADCConfigBlockSetDebug	209
ADCConfigBlockSetRangeSingle	209
ADCConfigBlockSetRegister	209
ADCConfigBlockGetTesting	209
ADCConfigBlockGetDebug	209
ADCConfigBlockInit	209
ADCConfigBlockInitForCounterScan	209
ADC_VerifyAndCorrectConfigBlock	209
ADCConfigBlockSetAllGainCodeAndDiffMode	209
ADCConfigBlockGetGainCode	209
ADCConfigBlockSetGainCode	209
ADCConfigBlockSetEndChannel	209
ADCConfigBlockSetChannelRange	209
ADCConfigBlockSetStartChannel	209
ADCConfigBlockSetScanRange	209
ADCConfigBlockSetCalMode	209
ADCConfigBlockGetCalMode	209
ADCConfigBlockGetStartChannel	210
ADCConfigBlockGetEndChannel	210
ADCConfigBlockGetOversample	210
ADCConfigBlockSetOversample	210
ADCConfigBlockGetTimeout	210
ADCConfigBlockSetTimeout	210
ADCConfigBlockGetTriggerMode	210
ADCConfigBlockSetTriggerMode	210
ADCConfigBlockSetDifferentialMode	210
ADCConfigBlockSetReference	210
ADCConfigBlockSetTriggerEdge	210
get_gain_code	210
get_cal_mode	210
ADCConfigBlockToYAML	210
ADCConfigBlockToJson	210
ADCConfigBlockSetScanAllChannels	210
ADCConfigBlockSetTriggerReference	210
is_all_digits	210
ADCConfigBlockGetJSONValueOrDefault	210
ADCConfigBlockGetJSONValueOrInt	211
NewADCConfigBlockFromJSON	211
ADCConfigBlockSetClockRate	211
ADCConfigBlockGetClockRate	211
24.74lib/ADCConfigBlock.h File Reference	211
24.74.1 Typedef Documentation	212
AIOUSBDevice	212
ADCMuxSettings	212
ADCConfigBlock	212
ADConfigBlock	212
24.74.2 Function Documentation	212
ADCConfigBlockInit	212
ADCConfigBlockInitForCounterScan	212

ADCConfigBlockInitializeDefault	212
ADC_VerifyAndCorrectConfigBlock	212
ADCConfigBlockSetAllGainCodeAndDiffMode	212
ADCConfigBlockSetRegister	212
ADCConfigBlockGetGainCode	212
ADCConfigBlockSetGainCode	212
ADCConfigBlockSetClockRate	212
ADCConfigBlockGetClockRate	213
ADCConfigBlockSetScanRange	213
ADCConfigBlockSetStartChannel	213
ADCConfigBlockSetEndChannel	213
ADCConfigBlockSetChannelRange	213
ADCConfigBlockSetCalMode	213
ADCConfigBlockGetCalMode	213
ADCConfigBlockToYAML	213
ADCConfigBlockGetStartChannel	213
ADCConfigBlockGetEndChannel	213
ADCConfigBlockGetOversample	213
ADCConfigBlockSetOversample	213
ADCConfigBlockGetTimeout	213
ADCConfigBlockSetTimeout	213
ADCConfigBlockGetTriggerMode	213
ADCConfigBlockSetTriggerMode	213
ADCConfigBlockSetReference	213
ADCConfigBlockSetTriggerEdge	214
ADCConfigBlockSetDifferentialMode	214
ADCConfigBlockSetRangeSingle	214
ADCConfigBlockCopy	214
ADCConfigBlockSetDevice	214
ADCConfigBlockSetAIOUSBDevice	214
ADCConfigBlockGetAIOUSBDevice	214
ADCConfigBlockInitializeFromAIOUSBDevice	214
ADCConfigBlockSetTesting	214
ADCConfigBlockGetTesting	214
ADCConfigBlockSetSize	214
ADCConfigBlockGetSize	214
ADCConfigBlockSetDebug	214
ADCConfigBlockGetDebug	214
ADCConfigBlockToJSON	214
NewADCConfigBlockFromJSON	214
DeleteADCConfigBlock	214
is_all_digits	214
24.75lib/AIOBuf.c File Reference	214
24.75.1 Detailed Description	215
24.75.2 Function Documentation	215
NewAIOBuf	215
DeleteAIOBuf	215
AIOBufGetSize	215
AIOBufGetTotalSize	215

AIOBufGetType	215
AIOBufGetRaw	215
AIOBufGetTypeSize	215
AIOBufRead	215
AIOBufWrite	215
AIOBufGetIterator	215
AIOBufIteratorIsValid	215
AIOBufIteratorNext	215
AIOBufIteratorGetValue	215
24.76lib/AIOBuf.h File Reference	215
24.76.1 Typedef Documentation	216
AIOBuf	216
AIOBufIterator	216
24.76.2 Enumeration Type Documentation	216
AIOBufType	216
24.76.3 Function Documentation	216
NewAIOBuf	216
DeleteAIOBuf	216
AIOBufGetSize	216
AIOBufGetType	216
AIOBufGetRaw	216
AIOBufRead	216
AIOBufWrite	216
AIOBufGetIterator	216
AIOBufIteratorGetValue	217
AIOBufIteratorIsValid	217
AIOBufIteratorNext	217
24.77lib/AIOChannelMask.c File Reference	217
24.77.1 Function Documentation	217
NewAIOChannelMask	217
DeleteAIOChannelMask	218
AIOChannelMaskIndices	218
AIOChannelMaskNextIndex	218
AIOChannelMaskSetMaskFromInt	218
AIOChannelMaskSetMaskAtIndex	218
AIOChannelMaskGetMaskAtIndex	218
AIOChannelMaskGetSize	218
AIOChannelMaskNumberChannels	218
AIOChannelMaskNumberSignals	218
AIOChannelMaskSetMaskFromStr	218
NewAIOChannelMaskFromStr	219
NewAIOChannelMaskFromChr	219
AIOChannelMaskToString	219
AIOChannelMaskToStringAtIndex	219
AIOChannelMaskGetMask	219
24.78lib/AIOChannelMask.h File Reference	219
24.78.1 Macro Definition Documentation	220
BIT_LENGTH	220
24.78.2 Typedef Documentation	220

aio_channel_obj	220
24.78.3 Function Documentation	220
NewAIOChannelMask	220
DeleteAIOChannelMask	220
NewAIOChannelMaskFromStr	220
NewAIOChannelMaskFromChr	221
AIOChannelMaskToString	221
AIOChannelMaskToStringAtIndex	221
AIOChannelMaskGetMask	221
AIOChannelMaskGetMaskAtIndex	221
AIOChannelMaskNumberChannels	221
AIOChannelMaskNumberSignals	221
AIOChannelMaskGetSize	221
AIOChannelMaskIndices	221
AIOChannelMaskNextIndex	222
AIOChannelMaskSetMaskFromInt	222
AIOChannelMaskSetMaskAtIndex	222
AIOChannelMaskSetMaskFromStr	222
24.79lib/AIOChannelRange.c File Reference	222
24.79.1 Enumeration Type Documentation	222
STATE	222
24.79.2 Function Documentation	223
NewAIOChannelRangeFromStr	223
DeleteAIOChannelRange	223
lookup_voltage_range	223
AIOChannelRangeToStr	223
AIOChannelRangeGetStart	223
AIOChannelRangeGetEnd	223
AIOChannelRangeGetGain	223
24.79.3 Variable Documentation	223
aio_channel_range_error	223
24.80lib/AIOChannelRange.h File Reference	223
24.80.1 Macro Definition Documentation	223
LENGTH_AD_GAIN_CODE_STRINGS	223
24.80.2 Typedef Documentation	223
AIOChannelRange	223
24.80.3 Function Documentation	224
NewAIOChannelRangeFromStr	224
DeleteAIOChannelRange	224
AIOChannelRangeToStr	224
AIOChannelRangeGetStart	224
AIOChannelRangeGetEnd	224
AIOChannelRangeGetGain	224
24.80.4 Variable Documentation	224
AD_GAIN_CODE_STRINGS	224
aio_channel_range_error	224
24.81lib/AIOCmd.c File Reference	224
24.81.1 Function Documentation	224
NewAIOCmdFromJSON	224

NewAIOCmd	224
DeleteAIOCmd	224
24.82lib/AIOCmd.h File Reference	224
24.82.1 Detailed Description	225
24.82.2 Typedef Documentation	225
AIOCmd	225
24.82.3 Function Documentation	225
NewAIOCmdFromJSON	225
NewAIOCmd	225
DeleteAIOCmd	225
24.83lib/AIOCommandLine.c File Reference	225
24.83.1 Function Documentation	226
AIO_CMDLINE_DEFAULT_OPTIONS	226
AIO_CMDLINE_SCRIPTING_OPTIONS	226
AIO_CMDLINE_CLEAR_OPTIONS	226
AIOProcessCommandLine	226
AIOProcessCmdline	226
AIOPrintUsage	226
NewDefaultAIOCommandLineOptions	227
NewAIOCommandLineOptionsFromDefaultOptions	227
DeleteAIOCommandLineOptions	227
AIOCommandLineListDevices	227
AIOCommandLineOptionsListDevices	227
AIOCommandLineOverrideADCCConfigBlock	227
AIOCommandLineOptionsOverrideADCCConfigBlock	227
AIOGetChannelRange	227
AIO_SCRIPTING_OPTIONS	227
AIO_CMDLINE_OPTIONS	227
24.83.2 Variable Documentation	228
opterr	228
optind	228
AIO_DEFAULT_CMDLINE_OPTIONS	228
AIO_DEFAULT_SCRIPTING_OPTIONS	228
24.84lib/AIOCommandLine.h File Reference	228
24.84.1 Macro Definition Documentation	229
DUMP	229
CNTS	229
JCONF	229
REPEAT	229
24.84.2 Typedef Documentation	229
AIOChannelRangeTmp	229
AIOCommandLineOptions	229
24.84.3 Enumeration Type Documentation	229
DeviceEnum	229
24.84.4 Function Documentation	230
NewDefaultAIOCommandLineOptions	230
NewAIOCommandLineOptionsFromDefaultOptions	230
AIO_CMDLINE_DEFAULT_OPTIONS	230
AIO_CMDLINE_SCRIPTING_OPTIONS	230

AIO_CMDLINE_CLEAR_OPTIONS	230
AIOProcessCmdline	230
AIOProcessCommandLine	230
AIOGetChannelRange	230
AIOPrintUsage	230
DeleteAIOCommandLineOptions	230
AIOCommandLineOptionsListDevices	231
AIOCommandLineOptionsOverrideADCConfigBlock	232
AIO_SCRIPTING_OPTIONS	232
AIO_CMDLINE_OPTIONS	232
AIOCommandLineListDevices	232
AIOCommandLineOverrideADCConfigBlock	232
24.84.5 Variable Documentation	232
AIO_DEFAULT_CMDLINE_OPTIONS	232
AIO_DEFAULT_SCRIPTING_OPTIONS	232
24.85lib/AIOConfiguration.c File Reference	232
24.85.1 Function Documentation	233
NewAIOConfiguration	233
DeleteAIOConfiguration	233
AIOConfigurationInitialize	233
AIOArgumentInitialize	233
AIOConfigurationSetTimeout	233
AIOConfigurationSetDebug	233
24.86lib/AIOConfiguration.h File Reference	233
24.86.1 Typedef Documentation	233
AIOConfiguration	233
ADCConfiguration	233
AIOContinuousBufConfiguration	233
24.86.2 Enumeration Type Documentation	233
ConfigurationType	234
ADCScanType	234
ADCSetCalFunction	234
24.86.3 Function Documentation	234
AIOConfigurationSetDebug	234
AIOConfigurationSetTimeout	234
NewAIOConfiguration	234
AIOConfigurationInitialize	234
AIOArgumentInitialize	234
24.87lib/AIOContinuousBuffer.c File Reference	234
24.87.1 Detailed Description	237
24.87.2 Typedef Documentation	238
RangeValueLookup	238
24.87.3 Function Documentation	238
ConvertCountsToVoltsFunction	238
RawCountsWorkFunction	239
AIOContinuousBufForceTerminateAcquisitionOverrun	239
AIOContinuousBufForceTerminateAcquisition	239
NewAIOContinuousBufForCounts	239
NewAIOContinuousBuf	239

AIOContinuousBufGetNumberChannels	239
AIOContinuousBufSetNumberChannels	239
AIOContinuousBufSetBaseSize	239
AIOContinuousBufGetBaseSize	239
AIOContinuousBufGetBufferSize	239
NewAIOContinuousBufForVolts	239
AIOContinuousBuf_InitConfiguration	239
AIOContinuousBufPushN	239
AIOContinuousBufPopN	239
AIOContinuousBufInitADCConfigBlock	239
AIOContinuousBufInitConfiguration	239
AIOContinuousBuf_SendPreConfig	240
AIOContinuousBufSendPreConfig	240
DeleteAIOContinuousBuf	240
AIOContinuousBufSetCountsBuffer	240
AIOContinuousBufSetVoltsBuffer	240
AIOContinuousBufSetStreamingBlockSize	240
AIOContinuousBufGetStreamingBlockSize	240
AIOContinuousBufGetADCConfigBlock	240
AIOContinuousBuf_SetCallback	240
AIOContinuousBufSetCallback	240
AIOContinuousBufSetNumberScans	240
AIOContinuousBufGetNumberScans	240
AIOContinuousBuf_BufSizeForCounts	240
AIOContinuousBufGetUnitSize	240
AIOContinuousBufSetUnitSize	240
AIOContinuousBufReset	240
AIOContinuousBufGetReadPosition	240
AIOContinuousBufGetWritePosition	240
AIOContinuousBufGetRemainingSize	240
AIOContinuousBufGetSize	240
AIOContinuousBufGetSizeNumElements	240
AIOContinuousBufGetStatus	240
AIOContinuousBufPending	240
AIOContinuousBufGetScansRead	240
AIOContinuousBufGetExitCode	240
AIOContinuousBufGetRunStatus	240
AIOContinuousBufCountScansAvailable	240
AIOContinuousBufCountsAvailable	240
AIOContinuousBufGetDataAvailable	240
AIOContinuousBufReadIntegerScanCounts	240
AIOContinuousBufGetNumberOfScansToRead	242
AIOContinuousBufSetNumberOfScansToRead	242
AIOContinuousBufReadIntegerNumberOfScans	242
AIOContinuousBufReadSingle	242
AIOContinuousBufReadCompleteScanCounts	242
AIOContinuousBufGetCallback	242
AIOContinuousBufSetClock	242
AIOContinuousBufGetClock	242

AIOContinuousBufStart	242
AIOContinuousBufStopAcquisition	242
AIOContinuousBufSetChannelMask	242
AIOContinuousBuf_NumberSignals	243
AIOContinuousBufNumberSignals	243
AIOContinuousBuf_NumberChannels	243
AIOContinuousBufNumberChannels	243
AIOContinuousBufWrite	243
AIOContinuousBufWriteCounts	243
AIOContinuousBufGetNumberSamplesPerScan	243
AIOContinuousBufGetTotalSamplesExpected	243
StartStreaming	243
SetConfig	243
ResetCounters	243
AIOContinuousBufLoadCounters	243
AIOContinuousBufCleanup	243
AIOContinuousBufPreSetup	243
AIOContinuousBufNumberSamplesAvailable	243
AIOContinuousBufNumberWriteSamplesRemaining	243
AIOContinuousBufReadNSamples	243
AIOContinuousBufInitiateCallbackAcquisition	243
number_to_read	243
continue_running	243
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction	244
AIOContinuousBufCallbackStart	244
AIOContinuousBufResetDevice	244
AIOContinuousBufRead	244
AIOContinuousBufLock	244
AIOContinuousBufUnlock	245
AIOContinuousBufSimpleSetupConfig	245
AIOContinuousBufEnd	245
AIOContinuousBuf_SetTesting	245
AIOContinuousBufSetTesting	245
AIOContinuousBufGetTesting	245
AIOContinuousBufSetDefaultModeForCounterScan	245
AIOContinuousBufSetDebug	245
AIOContinuousBufGetDebug	245
AIOContinuousBuf_SetDeviceIndex	245
AIOContinuousBufSetDeviceIndex	245
AIOContinuousBuf_SaveConfig	245
AIOContinuousBufSaveConfig	245
AIOContinuousBuf_SetStartAndEndChannel	245
AIOContinuousBufSetStartAndEndChannel	245
AIOContinuousBuf_SetChannelRangeGain	245
AIOContinuousBuf_SetChannelRange	245
AIOContinuousBufSetChannelRange	245
AIOContinuousBufSetTimeout	245
AIOContinuousBufGetTimeout	245
AIOContinuousBuf_SetOversample	245

AIOContinuousBufSetOversample	245
AIOContinuousBufSetOverSample	245
AIOContinuousBuf_GetOverSample	245
AIOContinuousBufGetOversample	245
AIOContinuousBuf_SetAllGainCodeAndDiffMode	245
AIOContinuousBufSetAllGainCodeAndDiffMode	245
AIOContinuousBuf_SetDiscardFirstSample	245
AIOContinuousBufSetDiscardFirstSample	245
AIOContinuousBuf_GetDeviceIndex	245
AIOContinuousBufGetDeviceIndex	245
GetJSONValueOrDefault	246
NewAIOContinuousBufFromJSON	246
AIOContinuousBufToJSON	246
24.87.4 Variable Documentation	246
TrueFalse	246
BaseSizeRange	246
24.88lib/AIOContinuousBuffer.h File Reference	246
24.88.1 Detailed Description	248
24.88.2 Macro Definition Documentation	249
ROOTCLOCK	249
24.88.3 Typedef Documentation	249
AIOUSB_WorkFn	249
AIOContinuousBuf	249
24.88.4 Enumeration Type Documentation	249
AIO_CONT_BUF_TYPE	249
24.88.5 Function Documentation	249
NewAIOContinuousBuf	249
NewAIOContinuousBufForCounts	249
NewAIOContinuousBufForVolts	249
DeleteAIOContinuousBuf	249
AIOContinuousBufInitConfiguration	249
AIOContinuousBufInitADCConfigBlock	250
AIOContinuousBufGetCallback	250
AIOContinuousBufSetCallback	250
AIOContinuousBufSetStreamingBlockSize	250
AIOContinuousBufGetStreamingBlockSize	250
AIOContinuousBufGetADCConfigBlock	250
AIOContinuousBufSetNumberChannels	250
AIOContinuousBufGetNumberChannels	250
AIOContinuousBufGetOversample	250
AIOContinuousBufSetOversample	250
AIOContinuousBufNumberChannels	250
AIOContinuousBufSetBaseSize	250
AIOContinuousBufGetBaseSize	250
AIOContinuousBufGetBufferSize	250
AIOContinuousBufSetUnitSize	250
AIOContinuousBufGetUnitSize	250
AIOContinuousBufSetTesting	250
AIOContinuousBufGetTesting	250

AIOContinuousBufSendPreConfig	250
AIOContinuousBufSetStartAndEndChannel	250
AIOContinuousBufSetAllGainCodeAndDiffMode	250
AIOContinuousBufGetDeviceIndex	250
AIOContinuousBufSetDiscardFirstSample	251
AIOContinuousBufSetChannelMask	251
AIOContinuousBufNumberSignals	252
AIOContinuousBufSetChannelRange	252
AIOContinuousBufSaveConfig	252
AIOContinuousBufSetDeviceIndex	252
AIOContinuousBufResetDevice	252
AIOContinuousBufSetTimeout	252
AIOContinuousBufGetTimeout	252
AIOContinuousBufSetDebug	252
AIOContinuousBufGetDebug	252
AIOContinuousBufGetNumberScans	252
AIOContinuousBufSetNumberScans	252
AIOContinuousBufNumberWriteSamplesRemaining	252
AIOContinuousBufNumberSamplesAvailable	252
AIOContinuousBufGetNumberSamplesPerScan	252
AIOContinuousBufGetTotalSamplesExpected	252
AIOContinuousBufReset	252
AIOContinuousBufPushN	252
AIOContinuousBufPopN	252
AIOContinuousBufLock	252
AIOContinuousBufUnlock	252
AIOContinuousBufCallbackStart	252
AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction	253
AIOContinuousBufStopAcquisition	253
AIOContinuousBufInitiateCallbackAcquisition	253
AIOContinuousBufGetReadPosition	253
AIOContinuousBufGetWritePosition	253
AIOContinuousBufGetRemainingSize	253
AIOContinuousBufGetStatus	253
AIOContinuousBufGetExitCode	253
AIOContinuousBufGetRunStatus	253
AIOContinuousBufPending	253
AIOContinuousBufGetScansRead	253
AIOContinuousBufReadIntegerScanCounts	253
AIOContinuousBufReadCompleteScanCounts	254
AIOContinuousBufReadIntegerNumberOfScans	254
AIOContinuousBufSetCountsBuffer	254
AIOContinuousBufSetVoltsBuffer	254
AIOContinuousBufCountScansAvailable	254
AIOContinuousBufSetClock	254
AIOContinuousBufGetClock	254
AIOContinuousBufEnd	254
AIOContinuousBufSimpleSetupConfig	254
AIOContinuousBufRead	254

AIOContinuousBufWrite	254
AIOContinuousBufWriteCounts	255
AIOContinuousBufCleanup	255
AIOContinuousBufToJSON	255
NewAIOContinuousBufFromJSON	255
24.89lib/AIOCountsConverter.c File Reference	255
24.89.1 Detailed Description	255
24.89.2 Function Documentation	256
default_out	256
enhanced_out	256
NewAIOCountsConverterWithBuffer	256
NewAIOCountsConverterWithScanLimiter	256
NewAIOCountsConverter	256
DeleteAIOCountsConverter	256
AIOCountsConverterReset	256
AIOCountsConverterConvertNScans	256
AIOCountsConverterConvertAllAvailableScans	256
Convert	256
AIOCountsConverterConvertFifo	256
AIOCountsConverterConvert	256
NewAIOGainRangeFromADCConfigBlock	256
DeleteAIOGainRange	256
24.90lib/AIOCountsConverter.h File Reference	256
24.90.1 Detailed Description	257
24.90.2 Typedef Documentation	257
AIOCountsConverter	257
24.90.3 Function Documentation	257
NewAIOCountsConverterWithBuffer	257
NewAIOCountsConverter	257
NewAIOCountsConverterFromAIOContinuousBuf	257
NewAIOCountsConverterWithScanLimiter	258
AIOCountsConverterReset	258
DeleteAIOCountsConverter	258
AIOCountsConverterConvertNScans	258
AIOCountsConverterConvertAllAvailableScans	258
AIOCountsConverterConvert	258
AIOCountsConverterConvertFifo	258
NewAIOGainRangeFromADCConfigBlock	258
DeleteAIOGainRange	258
24.91lib/AIODEviceInfo.c File Reference	258
24.91.1 Function Documentation	258
NewAIODEviceInfo	258
DeleteAIODEviceInfo	258
AIODEviceInfoGetName	258
AIODEviceInfoGetCounters	258
AIODEviceInfoGetDIOBytes	258
AIODEviceInfoGet	258
24.92lib/AIODEviceInfo.h File Reference	259
24.92.1 Detailed Description	259

24.92.2 Typedef Documentation	259
AIODeviceInfo	259
24.92.3 Function Documentation	259
NewAIODeviceInfo	259
DeleteAIODeviceInfo	259
AIODeviceInfoGetName	259
AIODeviceInfoGet	259
AIODeviceInfoGetCounters	259
AIODeviceInfoGetDIOBytes	259
24.93lib/AIODeviceQuery.c File Reference	260
24.93.1 Detailed Description	260
24.93.2 Function Documentation	260
NewAIODeviceQuery	260
DeleteAIODeviceQuery	261
AIODeviceQueryToStr	261
AIODeviceQueryToRepr	261
AIODeviceQueryGetProductID	261
AIODeviceQueryGetIndex	261
AIODeviceQueryNameSize	261
AIODeviceQueryGetName	262
AIODeviceQueryGetNumDIOBytes	262
AIODeviceQueryGetNumCounters	262
24.94lib/AIODeviceQuery.h File Reference	262
24.94.1 Detailed Description	263
24.94.2 Typedef Documentation	263
AIODeviceQuery	263
24.94.3 Function Documentation	263
NewAIODeviceQuery	263
DeleteAIODeviceQuery	263
AIODeviceQueryToStr	264
AIODeviceQueryToRepr	264
AIODeviceQueryGetProductID	264
AIODeviceQueryNameSize	264
AIODeviceQueryGetName	264
AIODeviceQueryGetNumDIOBytes	264
AIODeviceQueryGetNumCounters	265
AIODeviceQueryGetIndex	265
24.95lib/AIODeviceTable.c File Reference	265
24.95.1 Macro Definition Documentation	266
NUM_PROD_NAMES	266
24.95.2 Function Documentation	266
AIOUSB_SetInit	266
AIODeviceTableInit	266
AIOUSB_IsInit	266
AIOUSB_InitTest	266
AIOUSB_Cleanup	266
QueryDeviceInfo	266
ProductIDToName	266
ProductNameToID	267

GetDevices	267
AIODeviceTableGetUSBDeviceAtIndex	267
GetSafeDeviceName	267
AIOUSB_EnsureOpen	267
AIODeviceTableGetDeviceAtIndex	268
AIODeviceTableGetAIOUSBDeviceAtIndex	268
AIOUSBGetError	268
AIODeviceTableAddDeviceToDeviceTable	268
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice	268
ClearAIODeviceTable	268
AIODeviceTableSetDeviceID	268
AIOUSB_GetAllDevices	268
AIODeviceTablePopulateTableTest	268
CloseAllDevices	268
AIODeviceTableClearDevices	268
ClearDevices	268
AIODeviceTablePopulateTable	268
AIOUSB_Init	268
AIOUSB_Exit	268
AIOUSB_Reset	268
24.95.3 Variable Documentation	268
deviceTable	268
AIOUSB_INIT_PATTERN	268
aiousbInit	268
24.96lib/AIODeviceTable.h File Reference	269
24.96.1 Function Documentation	269
AIODeviceTableAddDeviceToDeviceTable	269
AIODeviceTableAddDeviceToDeviceTableWithUSBDevice	270
AIODeviceTablePopulateTable	270
AIODeviceTablePopulateTableTest	270
AIODeviceTableClearDevices	270
ClearDevices	270
AIODeviceTableGetDeviceAtIndex	270
AIODeviceTableGetAIOUSBDeviceAtIndex	270
AIODeviceTableGetUSBDeviceAtIndex	270
QueryDeviceInfo	270
GetDevices	270
GetSafeDeviceName	271
ProductIDToName	271
ProductNameToID	271
AIOUSB_Init	271
AIOUSB_EnsureOpen	271
AIOUSB_IsInit	271
AIOUSB_Exit	271
AIOUSB_Reset	271
AIODeviceTableInit	271
ClearAIODeviceTable	271
CloseAllDevices	272
AIOUSB_GetAllDevices	272

AIOUSBGetError	272
24.96.2 Variable Documentation	272
deviceTable	272
AIOUSB_INIT_PATTERN	272
24.97lib/AIOEither.c File Reference	272
24.97.1 Macro Definition Documentation	272
LOOKUP	272
AIO_EITHER_CHECK_VALUE	272
AIO_EITHER_GET_VALUE	273
24.97.2 Function Documentation	273
AIOEitherClear	273
AIOEitherSetRight	273
AIOEitherGetRight	273
AIOEitherSetLeft	273
AIOEitherGetLeft	273
AIOEitherHasError	273
AIOEitherToString	273
AIOEitherToInt	273
AIOEitherToShort	273
AIOEitherToUnsigned	273
AIOEitherToDouble	273
AIOEitherToAIONumber	273
AIOEitherToAIORetType	273
24.98lib/AIOEither.h File Reference	273
24.98.1 Detailed Description	274
24.98.2 Macro Definition Documentation	274
AIO_ERROR_VALUE	274
24.98.3 Typedef Documentation	274
AIO_EITHER_VALUE_ITEM	274
AIOEither	274
24.98.4 Enumeration Type Documentation	274
AIO_EITHER_TYPE	274
24.98.5 Function Documentation	275
AIOEitherClear	275
AIOEitherSetRight	275
AIOEitherGetRight	275
AIOEitherSetLeft	275
AIOEitherGetLeft	275
AIOEitherHasError	275
AIOEitherToString	275
AIOEitherToInt	275
AIOEitherToShort	275
AIOEitherToUnsigned	275
AIOEitherToDouble	275
AIOEitherToAIONumber	275
AIOEitherToAIORetType	275
24.99lib/AIOFifo.c File Reference	275
24.99.1 Detailed Description	276
24.99.2 Macro Definition Documentation	276

LOOKUP	. 276
24.99.3 Function Documentation	. 276
delta	. 276
AIOFifoWriteSizeRemaining	. 276
AIOFifoWriteSizeRemainingNumElements	. 276
AIOFifoGetSize	. 276
AIOFifoGetSizeNumElements	. 276
rdelta	. 277
AIOFifoReadSize	. 277
AIOFifoReadSizeNumElements	. 277
AIOFifoResize	. 277
AIOFifoInitialize	. 277
NewAIOFifo	. 277
AIOFifoAllOrNoneInitialize	. 277
NewAIOFifoAllOrNone	. 277
AIOFifoReset	. 277
AIOFifoGetRefSize	. 277
Push	. 277
PushN	. 277
Pop	. 277
PopN	. 277
NewAIOFifoTYPE	. 277
DeleteAIOFifoTYPE	. 277
DeleteAIOFifo	. 277
increment	. 277
AIOFifoRead	. 277
AIOFifoWrite	. 277
AIOFifoWriteAllOrNone	. 277
AIOFifoReadAllOrNone	. 277
AIOFifoReadPosition	. 277
AIOFifoWritePosition	. 277
TEMPLATE_AIOFIFO_API	. 277
TEMPLATE_AIOFIFO_API	. 277
24.100b/AIOFifo.h File Reference	. 277
24.100.1 Macro Definition Documentation	. 278
LOCKING_MECHANISM	. 278
GRAB_RESOURCE	. 278
RELEASE_RESOURCE	. 278
AIO_FIFO_INTERFACE	. 278
TEMPLATE_AIOFIFO_INTERFACE	. 279
TEMPLATE_AIOFIFO_API	. 279
24.100.2 Typedef Documentation	. 279
AIOFifo	. 279
TYPE	. 279
INPUT_TYPE	. 279
AIOFifoTYPE	. 279
24.100.3 Function Documentation	. 279
TEMPLATE_AIOFIFO_INTERFACE	. 279
TEMPLATE_AIOFIFO_INTERFACE	. 279

NewAIOFifo	279
DeleteAIOFifo	279
AIOFifoReset	279
AIOFifoRead	279
AIOFifoWrite	279
AIOFifoWriteAllOrNone	279
AIOFifoReadAllOrNone	280
AIOFifoGetRefSize	280
NewAIOFifoTYPE	280
Push	280
PushN	280
PopN	280
AIOFifoWriteSizeRemaining	280
AIOFifoWriteSizeRemainingNumElements	280
AIOFifoReadSize	280
AIOFifoReadSizeNumElements	280
AIOFifoGetSize	280
AIOFifoGetSizeNumElements	280
AIOFifoResize	280
AIOFifoReadPosition	280
AIOFifoWritePosition	280
24.101b/AIOList.c File Reference	280
24.101. Function Documentation	280
intToString	280
Deleteint	280
TAIL_Q_LIST_IMPLEMENTATION	281
TAIL_Q_LIST_IMPLEMENTATION	281
intlistFirst	281
Newintlist	281
Deleteintlist	281
intlistToString	281
intlistSize	281
intlistInsert	281
24.101.2variable Documentation	281
newone	281
24.102b/AIOList.h File Reference	281
24.102.1Macro Definition Documentation	281
TAIL_Q_LIST_TYPE	281
TAIL_Q_LIST_ENTRY_TYPE	281
TAIL_Q_LIST	281
TAIL_Q_LIST_IMPLEMENTATION	282
foreach_int	282
foreach_CStringArray_p	282
24.102.2Function Documentation	282
TAIL_Q_LIST	282
TAIL_Q_LIST	282
TAIL_Q_LIST_TYPE	282
TAIL_Q_LIST_ENTRY_TYPE	282
Newintlist	282

Deleteintlist	282
intlistToString	282
intlistSize	282
intlistFirst	282
intlistInsert	282
24.108b/AIOProductTypes.c File Reference	282
24.103. Function Documentation	283
AIO_PRODUCT_CONSTANT	283
AIO_PRODUCT_CONSTANT	283
AIO_PRODUCT_CONSTANT	283
AIO_PRODUCT_CONSTANT	283
AIO_PRODUCT_CONSTANT	283
AIO_PRODUCT_CONSTANT	283
AIO_PRODUCT_CONSTANT	283
AIO_PRODUCT_CONSTANT	283
NewAIOProductRange	283
DeleteAIOProductRange	283
AIOProductRangeStart	283
AIOProductRangeEnd	283
NewAIOProductGroup	283
DeleteAIOProductGroup	283
AIOProductGroupContains	283
groupcpy	283
24.108b/AIOProductTypes.h File Reference	284
24.104. Macro Definition Documentation	284
NUMARGS	284
AIO_RANGE	284
AIO_PRODUCT_GROUP	284
AIO_PRODUCT_CONSTANT	284
AIO_PRODUCT_EXTERN	285
24.104. Typedef Documentation	285
AIOProductRange	285
AIOProductGroup	285
24.104. Function Documentation	285
NewAIOProductRange	285
DeleteAIOProductRange	285
AIOProductRangeStart	285
AIOProductRangeEnd	285
NewAIOProductGroup	285
DeleteAIOProductGroup	285
AIOProductGroupContains	285
groupcpy	285
AIO_PRODUCT_EXTERN	285
AIO_PRODUCT_EXTERN	285
AIO_PRODUCT_EXTERN	285
AIO_PRODUCT_EXTERN	285
AIO_PRODUCT_EXTERN	285
AIO_PRODUCT_EXTERN	285
AIO_PRODUCT_EXTERN	285

AIO_PRODUCT_EXTERN	285
24.105b/AIOTuple.c File Reference	285
24.106b/AIOTuple.h File Reference	286
24.106.1Macro Definition Documentation	286
Aiotuple2_type	286
AIO_CHAR_ARRAY	286
Aiotuple2_ptr	286
Aiotuple2	286
Aiotuple2_to_str	286
24.106.2Function Documentation	286
Aiotuple2_type	286
Aiotuple2_type	286
Aiotuple2_AIORET_TYPE__CStringArray_pToString	286
DeleteAiotuple2_AIORET_TYPE__CStringArray_p	286
24.107b/AIOTypes.h File Reference	286
24.107.1Detailed Description	290
24.107.2Macro Definition Documentation	291
HAS_PTHREAD	291
EXPORTED_FUNCTION	291
CREATE_ENUM	291
CREATE_ENUM_W_START	291
LAST_ENUM	291
FIRST_ENUM	291
MIN_VALUE	291
MAX_VALUE	291
VALID_ENUM	291
ERR_UNLESS_VALID_ENUM	291
VALID_PRODUCT	291
GCC_VERSION	291
ACCES_DEPRECATED	291
LAMBDA	291
MIN	291
MAX	291
AUR_CBUF_SETUP	291
AUR_CBUF_EXIT	291
NUMBER_CHANNELS	291
foreach_array	292
AIO_MAKE_ERROR	292
AIOUSB_ERROR_VALUE	292
AIO_ASSERT	292
AIO_ASSERT_RET	292
AIO_ASSERT_AIORET_TYPE	292
AIO_ASSERT_NO_RETURN	292
AIO_ASSERT_EXIT	292
AIO_ASSERT_ERR_NO_RETURN	292
AIO_ASSERT_VALID_DATA	292
AIO_ASSERT_USB	292
AIO_ASSERT_DIOBUF	292
AIO_ASSERT_CHANNELMASK	292

AIO_ASSERT_CONFIG	292
AIO_ASSERT_AIOCONTBUF	292
AIO_ASSERT_AIOEITHER	292
AIO_ERROR	292
AIO_ERROR_VALID_DATA	292
AIO_ERROR_VALID_DATA_RETVAL	292
AIO_ERROR_VALID_AIORET_TYPE	292
AIO_ERROR_AIOEITHER_VALID_DATA	292
AIO_ERROR_VALID_DATA_W_CODE	292
AIO_ERROR_VALID_DATA_WITH_CODE	292
G_STMT_START	292
G_STMT_END	293
G_BREAKPOINT	293
EXIT_FN_IF_NO_VALID_USB	293
AIOUSB_ERROR_OFFSET	293
LIBUSB_RESULT_TO_AIOUSB_RESULT	293
AIOUSB_RESULT_TO_LIBUSB_RESULT	293
ROOTCLOCK	293
24.107.3 Typedef Documentation	293
AIORET_TYPE	293
AIORESULT	293
COUNTS	293
Ushort_Array	293
AIOBufferType	293
AIOUSB_BOOL	293
AIO_NUMBER	293
EnumStringLookup	293
24.107.4 Enumeration Type Documentation	293
AIO_SCAN_TYPE	293
THREAD_STATUS	293
AIOContinuousBufMode	294
anonymous enum	294
AIOUSB_BOOL_VAL	294
ProductIDS	294
anonymous enum	296
DACRange	296
FIFO_Method	297
ResultCode	297
anonymous enum	298
ADRegister	299
anonymous enum	299
ADGainCode	299
VENDOR_REQUEST	300
anonymous enum	301
ADCalMode	301
AIOCommandCode	301
24.108b/aiousb.h File Reference	301
24.108.1 Detailed Description	302
24.109b/AIOUSB_ADC.c File Reference	302

24.109.1 Detailed Description	304
24.109.2 Macro Definition Documentation	305
DEVICE_SAMPLE_BUFFER_SIZE	305
STREAMING_PNA_DEFINITIONS	305
24.109.3 Function Documentation	305
adc_get_bulk_data	305
ADC_ResetDevice	305
ADC_GetConfigSize	305
ADC_GetConfigRegisters	305
ADC_ReadADConfigBlock	305
ReadConfigBlock	305
WriteConfigBlock	305
ADC_Acquire_Reference_Counts	305
AIOUSB_GetScan	305
AIOUSB_ArrayCountsToVolts	306
AIOUSB_ArrayVoltsToCounts	306
AIOUSB_VoltsToCounts	306
ADC_GetChannelV	307
ADC_GetScanV	307
ADC_GetScan	307
ADC_GetConfig	308
adcblock_valid_trigger_settings	308
adcblock_valid_channel_settings	308
valid_config_block	308
adcblock_valid_size	308
ADC_SetConfig	308
ADC_CopyConfig	308
ADC_RangeAll	308
ADC_Range1	309
ADC_ADMode	310
ADC_SetOversample	310
ADC_GetOversample	310
ADC_SetAllGainCodeAndDiffMode	310
ADC_GetMaxClockRate	310
ADC_ClockRateForADCProduct	310
ADC_SetScanLimits	310
ADC_SetCal	311
ADC_CanCalibrate	311
ADC_QueryCal	311
ADC_Initialize	311
ADC_BulkAcquire	311
CreateSmartBuffer	312
ADC_BulkPoll	312
ADC_CreateFastITConfig	312
ADC_GetADConfigBlock_Registers	312
ADC_ClearFastITConfig	312
ADC_CreateADBuf	312
ADC_ClearADBuf	312
ADC_InitFastITScanV	312

ADC_ResetFastITScanV	312
ADC_SetFastITScanVChannels	313
ADC_Debug_Register_Settings	313
ADConfigBlockToYAML	313
ADC_GetFastITScanV	313
ADC_GetITScanV	313
AIOUSB_IsDiscardFirstSample	313
AIOUSB_SetDiscardFirstSample	313
AIOUSB_Copy_Config_Block	314
AIOUSB_Validate_ADC_Device	314
GetHiRef	314
DoLoadCalTable	314
AIOUSB_SetRangeSingle	314
ConfigureAndBulkAcquire	314
AIOUSB_ADC_InternalCal	314
AIOUSB_SetRegister	314
AIOUSB_GetRegister	314
AIOUSB_SetAllGainCodeAndDiffMode	314
AIOUSB_GetGainCode	315
AIOUSB_SetGainCode	315
AIOUSB_IsDifferentialMode	316
AIOUSB_SetDifferentialMode	316
AIOUSB_GetCalMode	316
AIOUSB_SetCalMode	316
AIOUSB_GetTriggerMode	316
AIOUSB_SetTriggerMode	316
AIOUSB_GetStartChannel	316
AIOUSB_GetEndChannel	316
AIOUSB_SetScanRange	316
AIOUSB_GetOversample	316
AIOUSB_SetOversample	316
AIOUSB_ADC_ExternalCal	316
24.109.4variable Documentation	317
adRanges	317
dRef	317
24.110b/AIOUSB_ADC.h File Reference	318
24.110.1Function Documentation	319
ADC_GetScanV	319
ADC_RangeAll	319
ADC_GetChannelV	319
ADC_GetScan	320
AIOUSB_GetScan	320
ADC_GetConfig	321
ADC_SetConfig	321
ADC_Range1	321
ADC_ADMODE	321
ADC_SetScanLimits	322
ADC_GetMaxClockRate	322
ADC_ClockRateForADCProduct	322

ADC_SetCal	322
ADC_QueryCal	322
ADC_CanCalibrate	322
ADC_Initialize	322
ADC_BulkAcquire	323
ADC_BulkPoll	323
ADC_InitFastITScanV	323
ADC_CreateFastITConfig	323
ADC_ResetFastITScanV	323
ADC_SetFastITScanVChannels	323
ADC_GetFastITScanV	323
ADC_GetITScanV	324
ADC_GetOversample	324
ADC_SetOversample	324
WriteConfigBlock	324
ReadConfigBlock	324
AIOUSB_SetAllGainCodeAndDiffMode	324
AIOUSB_GetGainCode	324
AIOUSB_SetGainCode	324
AIOUSB_IsDifferentialMode	325
AIOUSB_ADC_ExternalCal	325
ADC_SetAllGainCodeAndDiffMode	326
AIOUSB_SetDifferentialMode	326
AIOUSB_GetCalMode	326
AIOUSB_SetCalMode	326
AIOUSB_SetOversample	326
AIOUSB_GetOversample	326
AIOUSB_GetTriggerMode	326
AIOUSB_SetTriggerMode	326
AIOUSB_GetStartChannel	326
AIOUSB_GetEndChannel	326
AIOUSB_SetScanRange	326
AIOUSB_SetStreamingBlockSize	326
AIOUSB_IsDiscardFirstSample	326
AIOUSB_SetDiscardFirstSample	326
CreateSmartBuffer	327
AIOUSB_ADC_InternalCal	327
ADC_GetADConfigBlock_Registers	327
AIOUSB_SetRegister	327
AIOUSB_GetRegister	327
24.111b/AIOUSB_Core.c File Reference	327
24.111.1Detailed Description	328
24.111.2Macro Definition Documentation	329
BACKTRACE_DEBUG	329
AIOUSB_ENABLE_MUTEX	329
24.111.3Function Documentation	329
AIOUSB_Lock	329
AIOUSB_UnLock	329
AIOUSB_ResetChip	329

AIOUSB_Validate_Lock	329
AIOUSB_Validate	329
ResolveDeviceIndex	329
DeviceTableAtIndex	329
DeviceTableAtIndex_Lock	329
AIOUSB_GetDevice	329
AIOUSB_GetConfigBlock	329
AIOUSB_GetStreamingBlockSize	329
AIOUSB_SetStreamingBlockSize	330
AIOUSB_ClearFIFO	330
AIOUSB_GetVersion	330
AIOUSB_GetVersionDate	330
AIOUSB_GetMiscClock	330
AIOUSB_SetMiscClock	330
AIOUSB_GetCommTimeout	330
AIOUSB_SetCommTimeout	330
AIOUSB_Validate_Device	330
AIOUSB_InitConfigBlock	330
AIOUSB_ADC_LoadCalTable	330
AIOUSB_ADC_SetCalTable	330
GenericVendorWrite	330
GenericVendorRead	330
24.111.4Variable Documentation	330
aio_errno	330
productIDNameTable	331
24.112b/AIOUSB_Core.h File Reference	331
24.112.1Detailed Description	332
24.112.2Macro Definition Documentation	333
PUBLIC_EXTERN	333
PRIVATE	333
PROD_NAME_SIZE	333
24.112.3Typedef Documentation	333
AIOOption	333
AIOError	333
24.112.4Function Documentation	333
ADC_ResetDevice	333
AIOUSB_GetDeviceSerialNumber	333
AIOUSB_ResetChip	333
AIOUSB_Lock	333
AIOUSB_UnLock	333
AIOUSB_InitTest	333
AIOUSB_Validate	333
AIOUSB_Validate_Lock	333
DeviceTableAtIndex	333
DeviceTableAtIndex_Lock	333
AIOUSB_GetDevice	333
AIOUSB_GetConfigBlock	333
AIOUSB_SetMiscClock	333
AIOUSB_GetMiscClock	333

AIOUSB_SetCommTimeout	334
AIOUSB_GetCommTimeout	334
AIOUSB_GetVersion	334
AIOUSB_GetVersionDate	334
AIOUSB_GetResultCodeAsString	334
AIOUSB_VoltsToCounts	334
AIOUSB_ADC_LoadCalTable	334
AIOUSB_ADC_SetCalTable	334
AIOUSB_ClearFIFO	334
AIOUSB_GetStreamingBlockSize	334
AIOUSB_InitConfigBlock	334
GenericVendorRead	335
GenericVendorWrite	335
AIOUSB_Validate_Device	335
24.112.5Variable Documentation	335
aio_errno	335
adRanges	335
AIOUSB_INIT_PATTERN	335
aiousblnit	335
24.113b/AIOUSB_CTR.c File Reference	335
24.113.Detailed Description	336
24.113.2Macro Definition Documentation	336
RETURN_IF_INVALID_INPUT	336
JUMP_IF_INVALID_INPUT	336
JUMP_IF_NO_VALID_USB	336
24.113.3Function Documentation	337
CTR_8254Mode	337
CTR_8254Load	337
CTR_8254ModeLoad	337
CTR_8254ReadModeLoad	337
CTR_8254Read	337
CTR_8254ReadAll	337
CTR_8254ReadStatus	337
CTR_CalculateCountersForClock	337
CTR_StartOutputFreq	337
CTR_8254SelectGate	337
CTR_8254ReadLatched	337
24.114b/AIOUSB_CTR.h File Reference	337
24.114.Function Documentation	338
CTR_CalculateCountersForClock	338
CTR_8254Mode	338
CTR_8254Load	338
CTR_8254ModeLoad	338
CTR_8254ReadModeLoad	338
CTR_8254Read	338
CTR_8254ReadAll	338
CTR_8254ReadStatus	338
CTR_StartOutputFreq	338
CTR_8254SelectGate	338

CTR_8254ReadLatched	338
24.115b/AIOUSB_CustomEEPROM.c File Reference	338
24.115.1Detailed Description	339
24.115.2Macro Definition Documentation	339
EXIT_FN_IF_NO_VALID_USB	339
24.115.3Function Documentation	339
CustomEEPROMWrite	339
CustomEEPROMRead	339
24.116b/AIOUSB_CustomEEPROM.h File Reference	339
24.116.1Function Documentation	340
CustomEEPROMWrite	340
CustomEEPROMRead	340
24.117b/AIOUSB_DAC.c File Reference	340
24.117.1Detailed Description	340
24.117.2Function Documentation	341
DACDirect	341
DACMultiDirect	341
DACSetBoardRange	342
DACOutputOpen	342
DACOutputClose	342
DACOutputCloseNoEnd	342
DACOutputSetCount	342
DACOutputFrame	342
DACOutputFrameRaw	342
DACOutputStart	342
DACOutputSetInterlock	342
24.118b/AIOUSB_DAC.h File Reference	342
24.118.1Function Documentation	342
DACDirect	342
DACMultiDirect	342
DACSetBoardRange	343
DACOutputOpen	343
DACOutputClose	343
DACOutputCloseNoEnd	343
DACOutputSetCount	343
DACOutputFrame	343
DACOutputFrameRaw	343
DACOutputStart	343
DACOutputSetInterlock	343
24.119b/AIOUSB_DIO.c File Reference	343
24.119.1Detailed Description	344
24.119.2Macro Definition Documentation	344
GET_ENDPOINT	344
24.119.3Function Documentation	344
MASK_BYTES_SIZE	344
TRISTATE_BYTES_SIZE	344
aiousb_htons	344
DIO_ConfigureWithDIOBuf	345
DIO_Configure	345

DIO_ConfigureEx	345
DIO_ConfigurationQuery	345
DIO_WriteAll	345
DIO_Write8	345
DIO_Write1	345
DIO_ReadAll	345
DIO_ReadIntoDIOBuf	345
DIO_ReadAllToDIOBuf	345
DIO_ReadAllToCharStr	345
DIO_Read8	345
DIO_Read1	345
DIO_StreamOpen	345
DIO_StreamClose	345
DIO_StreamSetClocks	345
pow_of_minsize	345
DIO_StreamFrame	345
24.120b/AIOUSB_DIO.h File Reference	346
24.120. Detailed Description	346
24.120. Function Documentation	347
DIO_ConfigureWithDIOBuf	347
DIO_Configure	347
DIO_ConfigureEx	347
DIO_ConfigurationQuery	347
DIO_WriteAll	347
DIO_Write8	347
DIO_Write1	347
DIO_ReadAllToDIOBuf	347
DIO_ReadIntoDIOBuf	347
DIO_ReadAll	347
DIO_ReadAllToCharStr	347
DIO_Read8	347
DIO_Read1	347
DIO_StreamOpen	347
DIO_StreamClose	347
DIO_StreamSetClocks	347
DIO_StreamFrame	347
24.121b/AIOUSB_Log.c File Reference	347
24.121. Variable Documentation	348
cont_thread	348
message_lock	348
outfile	348
AIOUSB_DEBUG_LEVEL	348
24.122b/AIOUSB_Log.h File Reference	348
24.122. Macro Definition Documentation	349
GREEN	349
RED	349
MAGENTA	349
CYAN	349
AIO_DEVEL_STR	349

AIO_DEBUG_STR	349
AIO_WARN_STR	349
AIO_INFO_STR	349
AIO_ERROR_STR	349
AIO_FATAL_STR	349
AIO_RESET_STR	349
AIOUSB_LOG	349
AIOUSB_DEVEL	349
AIOUSB_DEBUG	349
AIOUSB_WARN	349
AIOUSB_INFO	349
AIOUSB_ERROR	349
AIOUSB_FATAL	349
24.122.2 Enumeration Type Documentation	349
AIO_DEBUG_LEVEL	349
24.122.3 Variable Documentation	349
AIOUSB_DEBUG_LEVEL	349
LOG_LEVEL	349
cont_thread	350
message_lock	350
outfile	350
24.123a/AIOUSB_Properties.c File Reference	350
24.123. Detailed Description	350
24.123.2 Macro Definition Documentation	351
RESULT_TEXT_SIZE	351
NUM_RESULT_CODES	351
24.123.3 Function Documentation	351
non_usb_supported_device	351
AIOUSB_GetDeviceByProductID	351
AIOUSB_GetDeviceSerialNumber	351
GetDeviceSerialNumber	351
GetDeviceBySerialNumber	351
AIOUSB_FindDevices	351
AIOUSB_FindDeviceIndicesByGroup	351
AIOUSB_FindDevicesByGroup	351
AIOUSB_GetDeviceProperties	351
AIOUSB_GetResultCodeAsString	351
AIOUSB_ShowDevices	352
AIOUSB_ListDevices	352
24.124a/AIOUSB_Properties.h File Reference	352
24.124. Detailed Description	352
24.124.2 Enumeration Type Documentation	352
AIODisplayType	352
24.124.3 Function Documentation	352
AIOUSB_GetDeviceByProductID	352
GetDeviceBySerialNumber	353
GetDeviceSerialNumber	353
AIOUSB_GetDeviceSerialNumber	353
AIOUSB_GetDeviceProperties	353

AIOUSB_GetResultCodeAsString	353
AIOUSB_ListDevices	353
AIOUSB_ShowDevices	353
AIOUSB_FindDevices	353
AIOUSB_FindDevicesByGroup	353
AIOUSB_FindDeviceIndicesByGroup	353
24.125b/AIOUSB_USB.c File Reference	353
24.126b/AIOUSB_USB.h File Reference	354
24.126.1Macro Definition Documentation	354
usb_control_xfer	354
usb_bulk_xfer	354
usb_open	354
usb_close	354
usb_free_devices	354
usb_get_devices	354
24.127b/AIOUSB_WDG.c File Reference	354
24.127.1Function Documentation	354
NewWDGConfig	354
doSomething	354
DeleteWDGConfig	354
WDG_SetConfig	355
WDG_GetStatus	355
WDG_Pet	355
24.128b/AIOUSB_WDG.h File Reference	355
24.128.1Enumeration Type Documentation	356
WDGVals	356
24.128.2Function Documentation	356
NewWDGConfig	356
DeleteWDGConfig	356
WDG_SetConfig	356
WDG_GetStatus	356
WDG_Pet	356
24.129b/AIOUSBDevice.c File Reference	357
24.129.1Function Documentation	357
NewAIOUSBDevice	357
NewAIOUSBDeviceFromJSON	357
AIOUSBDeviceToJSON	357
DeleteAIOUSBDevice	357
AIOUSBDeviceInitializeWithProductID	357
AIOUSBDeviceSetTimeout	357
AIOUSBDeviceGetTimeout	357
AIOUSBDeviceCopyADCCConfigBlock	357
AIOUSBDeviceSetADCCConfigBlock	357
AIOUSBDeviceSize	357
AIOUSBDeviceGetADCCConfigBlock	357
AIOUSBDeviceWriteADCCConfig	357
AIOUSBDeviceSetTesting	357
AIOUSBDeviceGetStreamingBlockSize	357
AIOUSBDeviceGetTesting	357

AIOUSBDeviceGetUSBHandle	357
AIOUSBDeviceSetUSBHandle	357
AIOUSBDeviceGetUSBHandleFromDeviceIndex	358
AIOUSBDeviceGetDiscardFirstSample	358
AIOUSBDeviceSetDiscardFirstSample	358
24.130b/AIOUSBDevice.h File Reference	358
24.130.1 Typedef Documentation	358
DeviceDescriptor	358
24.130.2 Function Documentation	358
AIOUSBDeviceToJSON	358
NewAIOUSBDeviceFromJSON	358
AIOUSBDeviceInitializeWithProductID	358
AIOUSBDeviceGetUSBHandle	358
AIOUSBDeviceGetUSBHandleFromDeviceIndex	358
AIOUSBDeviceSetUSBHandle	359
AIOUSBDeviceSetADCConfigBlock	359
AIOUSBDeviceGetADCConfigBlock	359
AIOUSBDeviceCopyADCConfigBlock	359
AIOUSBDeviceSetTesting	359
AIOUSBDeviceSize	359
AIOUSBDeviceGetTesting	359
AIOUSBDeviceGetStreamingBlockSize	359
AIOUSBDeviceGetDiscardFirstSample	359
AIOUSBDeviceSetDiscardFirstSample	359
AIOUSBDeviceSetTimeout	359
AIOUSBDeviceGetTimeout	359
AIOUSBDeviceWriteADCConfig	359
24.131lib/cJSON.c File Reference	359
24.131.1 Function Documentation	360
cJSON_GetErrorPtr	360
cJSON_InitHooks	360
cJSON_Delete	360
cJSON_ParseWithOpts	360
cJSON_Parse	360
cJSON_Print	360
cJSON_PrintUnformatted	360
cJSON_AsInteger	360
cJSON_GetArraySize	360
cJSON_GetArrayItem	360
cJSON_GetObjectItem	360
cJSON_AddItemToArray	360
cJSON_AddItemToObject	360
cJSON_AddItemReferenceToArray	360
cJSON_AddItemReferenceToObject	360
cJSON_DetachItemFromArray	360
cJSON_DeleteItemFromArray	360
cJSON_DetachItemFromObject	360
cJSON_DeleteItemFromObject	360
cJSON_ReplaceItemInArray	360

cJSON_ReplacetemInObject	360
cJSON_CreateNull	360
cJSON_CreateTrue	360
cJSON_CreateFalse	360
cJSON_CreateBool	360
cJSON_CreateNumber	360
cJSON_CreateString	360
cJSON_CreateArray	360
cJSON_CreateObject	360
cJSON_CreateIntArray	361
cJSON_CreateFloatArray	361
cJSON_CreateDoubleArray	361
cJSON_CreateStringArray	361
cJSON_Duplicate	361
cJSON_Minify	361
24.132b/cJSON.h File Reference	361
24.132.1Macro Definition Documentation	362
cJSON_False	362
cJSON_True	362
cJSON_NULL	362
cJSON_Number	362
cJSON_String	362
cJSON_Array	362
cJSON_Object	362
cJSON_IsReference	362
cJSON_AddNullToObject	362
cJSON_AddTrueToObject	362
cJSON_AddFalseToObject	362
cJSON_AddBoolToObject	362
cJSON_AddNumberToObject	362
cJSON_AddStringToObject	362
cJSON_SetIntValue	362
24.132.2Typedef Documentation	362
cJSON	362
cJSON_Hooks	362
24.132.3Function Documentation	362
cJSON_InitHooks	362
cJSON_Parse	362
cJSON_Print	362
cJSON_PrintUnformatted	362
cJSON_Delete	363
cJSON_AsInteger	363
cJSON_GetArraySize	363
cJSON_GetArrayItem	363
cJSON_GetObjectItem	363
cJSON_GetErrorPtr	363
cJSON_CreateNull	363
cJSON_CreateTrue	363
cJSON_CreateFalse	363

cJSON_CreateBool	363
cJSON_CreateNumber	363
cJSON_CreateString	363
cJSON_CreateArray	363
cJSON_CreateObject	363
cJSON_CreateIntArray	363
cJSON_CreateFloatArray	363
cJSON_CreateDoubleArray	363
cJSON_CreateStringArray	363
cJSON_AddItemToArray	363
cJSON_AddItemToObject	363
cJSON_AddItemReferenceToArray	363
cJSON_AddItemReferenceToObject	363
cJSON_DetachItemFromArray	363
cJSON_DeleteItemFromArray	363
cJSON_DetachItemFromObject	363
cJSON_DeleteItemFromObject	363
cJSON_ReplaceItemInArray	363
cJSON_ReplaceItemInObject	363
cJSON_Duplicate	363
cJSON_ParseWithOpts	363
cJSON_Minify	363
24.133b/CStringArray.c File Reference	364
24.133. Function Documentation	364
NewCStringArrayWithStrings	364
NewCStringArray	364
NewCStringArrayFromCArgs	364
DeleteCStringArray	364
CopyCStringArray	364
CStringArrayToString	364
CStringArrayToStringWithDelimiter	364
24.134b/CStringArray.h File Reference	364
24.134. Macro Definition Documentation	365
STRING_ARRAY	365
24.134. Typedef Documentation	365
CStringArray	365
24.134. Function Documentation	365
NewCStringArray	365
NewCStringArrayWithStrings	365
NewCStringArrayFromCArgs	365
DeleteCStringArray	365
CopyCStringArray	365
CStringArrayToString	365
CStringArrayToStringWithDelimiter	365
24.135b/DIOBuf.c File Reference	365
24.135. Detailed Description	366
24.135. Function Documentation	366
NewDIOBuf	366
NewDIOBufFromChar	366

NewDIOBufFromBinStr	366
DIOBufReplaceString	366
DIOBufReplaceBinString	367
DeleteDIOBuf	367
DIOBufResize	367
DIOBufSize	367
DIOBufByteSize	367
DIOBufToString	367
DIOBufToHex	367
DIOBufToBinary	367
DIOBufToInvertedBinary	367
DIOBufSetIndex	367
DIOBufGetIndex	368
DIOBufGetByteAtIndex	368
DIOBufSetByteAtIndex	368
24.136b/DIOBuf.h File Reference	368
24.136.1 Typedef Documentation	369
DIOBufferType	369
24.136.2 Function Documentation	369
NewDIOBuf	369
NewDIOBufFromChar	369
NewDIOBufFromBinStr	369
DeleteDIOBuf	369
DIOBufReplaceString	369
DIOBufReplaceBinString	370
DIOBufToHex	370
DIOBufToBinary	370
DIOBufToInvertedBinary	370
DIOBufResize	370
DIOBufSize	370
DIOBufByteSize	370
DIOBufToString	370
DIOBufSetIndex	370
DIOBufGetIndex	371
DIOBufGetByteAtIndex	371
DIOBufSetByteAtIndex	371
24.137b/mocks/mock_aiocontbuf_get_data.c File Reference	371
24.137.1 Function Documentation	371
aiocontbuf_get_bulk_data	371
24.138b/mocks/mock_aiocontbuf_get_data_arduino.c File Reference	371
24.138.1 Function Documentation	372
aiocontbuf_get_bulk_data	372
adc_get_bulk_data	372
CloseAllDevices	372
24.139b/mocks/mock_capture_usb.c File Reference	372
24.139.1 Detailed Description	373
24.139.2 Typedef Documentation	373
init_device	373
24.139.3 Enumeration Type Documentation	373

IO_DIRECTION	373
24.139.4Function Documentation	373
mock_usb_control_transfer	373
mock_usb_bulk_transfer	373
mock_usb_request	373
mock_usb_reset_device	374
mock_usb_put_config	374
mock_usb_get_config	374
InitializeUSBDevice	374
24.139.5Variable Documentation	374
orig_usb_control_transfer	374
orig_usb_bulk_transfer	374
orig_usb_request	374
orig_usb_reset_device	374
orig_usb_put_config	374
orig_usb_get_config	374
outfile	374
24.140b/mocks/mock_dio.c File Reference	374
24.140.1Enumeration Type Documentation	374
IO_DIRECTION	374
24.140.2Function Documentation	375
save_results	375
mock_usb_control_transfer	375
mock_usb_bulk_transfer	375
24.141b/mocks/mock_usb_xfers.c File Reference	375
24.141.1Typedef Documentation	376
add_devices_fn	376
24.141.2Enumeration Type Documentation	376
IO_DIRECTION	376
24.141.3Function Documentation	376
mock_usb_control_transfer	376
mock_usb_bulk_transfer	376
mock_usb_reset_device	376
mock_USBDevicePutADCConfigBlock	376
mock_USBDeviceFetchADCConfigBlock	376
AddAllACCESUSBDevices	376
24.141.4Variable Documentation	376
direction	376
KEEP	376
24.142b/USBDevice.c File Reference	376
24.142.1Detailed Description	377
24.142.2Function Documentation	377
InitializeUSBDevice	377
NewUSBDevice	377
CopyUSBDevice	377
USBDeviceClose	377
AddAllACCESUSBDevices	377
AddDevice	377
USBDeviceGetIdProduct	377

DeleteUSBDevices	377
DeleteUSBDevice	377
USBDeviceSetDebug	377
USBDeviceGetUSBDeviceHandle	377
get_usb_device	377
USBDeviceFetchADCConfigBlock	377
USBDevicePutADCConfigBlock	377
usb_control_transfer	377
usb_bulk_transfer	378
usb_request	378
usb_reset_device	378
24.143b/USBDevice.h File Reference	378
24.143.1Macro Definition Documentation	379
INTERNAL_METHOD	379
24.143.2Typedef Documentation	379
USBDevice	379
LIBUSBArgs	379
24.143.3Function Documentation	379
NewUSBDevice	379
DeleteUSBDevice	379
CopyUSBDevice	379
InitializeUSBDevice	379
AddAllACCESUSBDevices	379
DeleteUSBDevices	379
USBDeviceClose	379
USBDeviceGetIdProduct	379
USBDeviceFetchADCConfigBlock	379
USBDevicePutADCConfigBlock	379
usb_control_transfer	379
usb_bulk_transfer	379
usb_request	379
usb_reset_device	379
get_usb_device	379
USBDeviceGetUSBDeviceHandle	379
24.144b/wrappers/scilab/foo.c File Reference	379
24.144.1Function Documentation	380
foo_something	380
24.145b/wrappers/scilab/foo.h File Reference	380
24.145.1Function Documentation	380
foo_something	380
24.146samples/TestLib/aiocommon.c File Reference	380
24.146.1Macro Definition Documentation	380
DUMP	380
CNTS	380
JCONF	380
REPEAT	380
24.146.2Function Documentation	380
get_channel_range	380
process_aio_cmd_line	380

print_aio_usage	381
aio_list_devices	381
aio_override_adconfig_settings	381
aio_supply_default_command_line_settings	381
aio_override_aiobuf_settings	381
24.146.3Variable Documentation	381
AIO_OPTIONS	381
24.147.3Examples/TestLib/aiocommon.h File Reference	381
24.147.4Function Documentation	381
get_channel_range	381
process_aio_cmd_line	381
print_aio_usage	381
aio_list_devices	381
aio_override_aiobuf_settings	381
aio_override_adconfig_settings	381
aio_supply_default_command_line_settings	381
24.147.2Variable Documentation	381
AIO_OPTIONS	381
24.148.3Examples/TestLib/TestCaseSetup.cpp File Reference	382
24.148.4Variable Documentation	382
CURRENT_DEBUG_LEVEL	382
24.149.3Examples/TestLib/TestCaseSetup.h File Reference	382
24.149.4Macro Definition Documentation	383
ERROR_LEVEL	383
FATAL_LEVEL	383
ALERT_LEVEL	383
WARN_LEVEL	383
INFO_LEVEL	383
DEBUG_LEVEL	383
TRACE_LEVEL	383
LOG	383
INFO	383
TRACE	383
DEBUG	383
ERROR	383
FATAL	383
TERSE_LOGGING	383
VERBOSE_LOGGING	383
THROW_ERROR	383
CHECK_RESULT	383
24.149.2Variable Documentation	383
MAX_NAME_SIZE	383
DEF_MAX_CHANNELS	384
DEF_NUM_CHANNELS	384
DEF_CAL_CHANNEL	384
CURRENT_DEBUG_LEVEL	384
24.150.3Examples/USB-AI16-16/bulk_acquire_sample.c File Reference	384
24.150.4Function Documentation	384
process_cmd_line	384

main	384
print_usage	384
24.154samples/USB-AI16-16/burst_test.c File Reference	384
24.151.1Detailed Description	384
24.151.2Function Documentation	385
find_ai_board	385
24.155samples/USB-AI16-16/continuous_mode.c File Reference	385
24.152.1Detailed Description	385
24.152.2Function Documentation	385
fnd	385
24.153samples/USB-AI16-16/continuous_mode_callback.c File Reference	385
24.153.1Function Documentation	386
get_channel_range	386
process_cmd_line	386
fnd	386
capture_data	386
main	386
24.153.2Variable Documentation	386
fp	386
24.154samples/USB-AI16-16/continuous_mode_from_json_config.c File Reference	386
24.154.1Detailed Description	386
24.154.2Function Documentation	387
get_channel_range	387
process_cmd_line	387
fnd	387
capture_data	387
main	387
24.154.3Variable Documentation	387
fp	387
24.155samples/USB-AI16-16/daitest.cpp File Reference	387
24.155.1Function Documentation	387
handle_signal	387
main	387
24.155.2Variable Documentation	387
exit_sample	387
old_action	387
24.156samples/USB-AI16-16/dio_sample.c File Reference	387
24.156.1Function Documentation	388
find_ai_board	388
main	388
24.157samples/USB-AI16-16/diotest.c File Reference	388
24.157.1Function Documentation	388
main	388
24.158samples/USB-AI16-16/diotest2.cpp File Reference	388
24.158.1Function Documentation	388
main	388
24.159samples/USB-AI16-16/HOLD/dirctest.c File Reference	388
24.159.1Function Documentation	389
get_channel_range	389

process_cmd_line	389
fnf	389
capture_data	389
main	389
24.159.2/variable Documentation	389
fp	389
24.160samples/USB-AI16-16/HOLD/julian_test.c File Reference	389
24.160. Function Documentation	389
find_ai_board	389
main	389
24.161samples/USB-AI16-16/HOLD/reverse_cal_table.cpp File Reference	389
24.161. Function Documentation	389
goDolt	389
main	389
24.162samples/USB-AI16-16/reverse_cal_table.cpp File Reference	389
24.162. Function Documentation	390
goDolt	390
main	390
24.163samples/USB-AI16-16/HOLD/sample_dio.c File Reference	390
24.163. Function Documentation	390
find_ai_board	390
main	390
24.164samples/USB-AI16-16/HOLD/slow_receiver_test.cpp File Reference	390
24.164. Function Documentation	390
process_cmd_line	390
print_usage	390
main	391
24.165samples/USB-AI16-16/HOLD/test.c File Reference	391
24.165. Function Documentation	391
get_channel_range	391
process_cmd_line	391
fnf	391
capture_data	391
main	391
24.165.2/variable Documentation	391
fp	391
24.166samples/USB-AI16-16/java/extcal/src/main/java/com/accesio/extcal.cpp File Reference	391
24.166. Detailed Description	392
24.166.2 Function Documentation	392
main	392
24.167samples/USB-AI16-16/jni/read_channels_test.c File Reference	392
24.167. Function Documentation	392
find_ai_board	392
main	392
24.168samples/USB-AI16-16/read_channels_test.c File Reference	393
24.168. Function Documentation	393
find_ai_board	393
main	393
24.169samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp File Reference	393

24.169.	Function Documentation	393
	get_options	393
	main	393
24.170.	samples/USB-AI16-16/sample.cpp File Reference	394
24.170.	Function Documentation	394
	main	394
24.171.	samples/USB-AO16-16/sample.cpp File Reference	394
24.171.	Function Documentation	394
	main	394
24.172.	samples/USB-DA12-8A/sample.cpp File Reference	394
24.172.	Function Documentation	394
	main	394
24.173.	samples/USB-DIO-16/sample.cpp File Reference	394
24.173.	Function Documentation	395
	main	395
24.174.	samples/USB-AI16-16/simp_test.cpp File Reference	395
24.174.	Function Documentation	395
	main	395
24.175.	samples/USB-AI16-16/simple_continuous_with_json.c File Reference	395
24.175.	Detailed Description	395
24.175.	Function Documentation	395
	get_channel_range	395
	process_cmd_line	396
	fnd	396
	main	396
24.175.3/	variable Documentation	396
	fp	396
24.176.	samples/USB-AI16-16/start_stop_continuous.c File Reference	396
24.176.	Function Documentation	396
	get_channel_range	396
	process_cmd_line	396
	run_acquisition	396
	fnd	396
	main	396
24.176.2/	variable Documentation	396
	fp	396
24.177.	samples/USB-AI16-16/test_fastscan.cpp File Reference	396
24.177.	Function Documentation	397
	main	397
24.177.2/	variable Documentation	397
	CURRENT_DEBUG_LEVEL	397
24.178.	samples/USB-ARB1/stream_test.c File Reference	397
24.178.	Function Documentation	397
	find_ai_board	397
	main	397
24.179.	samples/USB-DA12-8A/SampleClass.cpp File Reference	397
24.179.	Function Documentation	397
	main	397
24.180.	samples/USB-DIO-16/receiver.cpp File Reference	397

24.180.	Function Documentation	398
	main	398
24.181.	samples/USB-DIO-16/standalone_receiver.c File Reference	398
24.181.	Function Documentation	398
	main	398
24.182.	samples/USB-DIO-32/daisample.c File Reference	398
24.182.	Function Documentation	398
	main	398
24.183.	samples/USB-DIO-32/read_and_write_sample.c File Reference	398
24.183.	Function Documentation	398
	main	398
24.184.	samples/USB-DIO-96/read_and_write_sample.c File Reference	399
24.184.	Macro Definition Documentation	400
	BITS_PER_BYTE	400
	DEVICES_REQUIRED	400
	MAX_DIO_BYTES	400
	MASK_BYTES	400
	MAX_NAME_SIZE	400
24.184.	Enumeration Type Documentation	400
	EXIT_CODE	400
24.184.	Function Documentation	400
	find_dio_96	400
	main	400
24.185.	samples/USB-DIO-32/sample3.c File Reference	400
24.185.	Function Documentation	401
	main	401
24.186.	samples/USB-DIO-48/read_and_write.c File Reference	401
24.186.	Detailed Description	401
24.186.	Function Documentation	402
	find_dio	402
	CHECK_RESULT	402
	main	402
24.187.	samples/USB-DIO-96/dio96_read_write.c File Reference	402
24.187.	Macro Definition Documentation	402
	BITS_PER_BYTE	402
	DEVICES_REQUIRED	402
	MAX_DIO_BYTES	402
	MASK_BYTES	402
	MAX_NAME_SIZE	402
	PORT_C	402
	PORT_B	402
	PORT_A	402
	MAKE_MASK	402
24.187.	Function Documentation	403
	find_dio_96	403
	show_byte	403
	main	403
24.188.	samples/USB-DIO-96/mytest.c File Reference	403
24.188.	Macro Definition Documentation	404

BITS_PER_BYTE	404
DEVICES_REQUIRED	404
MAX_DIO_BYTES	404
MASK_BYTES	404
MAX_NAME_SIZE	404
PORT_C	404
PORT_B	404
PORT_A	404
MAKE_MASK	404
24.188.2 Function Documentation	404
find_dio_96	404
show_byte	405
main	405
24.189.3 samples/USB-DIO-96/tmp.c File Reference	405
24.189.1 Macro Definition Documentation	405
BITS_PER_BYTE	405
DEVICES_REQUIRED	405
MAX_DIO_BYTES	405
MASK_BYTES	405
MAX_NAME_SIZE	405
PORT_C	405
PORT_B	406
PORT_A	406
MAKE_MASK	406
24.189.2 Function Documentation	406
find_dio_96	406
show_byte	406
main	406
24.190.3 samples/USB-DIO-96/write_sample.c File Reference	406
24.190.1 Macro Definition Documentation	407
BITS_PER_BYTE	407
DEVICES_REQUIRED	407
MAX_DIO_BYTES	407
MASK_BYTES	407
MAX_NAME_SIZE	407
24.190.2 Enumeration Type Documentation	407
EXIT_CODE	407
24.190.3 Function Documentation	408
find_dio_96	408
fnd	408
main	408
24.191.3 samples/USB-IDIO-16_8/idio_sample.c File Reference	408
24.191.1 Detailed Description	408
24.191.2 Macro Definition Documentation	409
RATE_LIMIT	409
24.191.3 Function Documentation	409
main	409
24.192.3 samples/USB-IDIO-16_8/idio_sample2.c File Reference	409
24.192.1 Detailed Description	409

24.192.2	Macro Definition Documentation	409
	RATE_LIMIT	409
24.192.3	Function Documentation	410
	main	410
24.193	samples/USB-IDIO-16_8/perftest.c File Reference	410
24.193.1	Function Documentation	410
	main	410
24.194	samples/USB-IIRO-16_8/iiro_sample.c File Reference	410
24.194.1	Detailed Description	411
24.194.2	Macro Definition Documentation	411
	RATE_LIMIT	411
24.194.3	Function Documentation	411
	find_idio	411
	main	411
24.195	media/jdamon/Development/Documents/Projects/aiusb_patrick_mcbride_issue/AIOUSB/README- E.md File Reference	411
24.196	b/wrappers/README.md File Reference	411
24.197	samples/USB-AI16-16/android/read_channels_test/README.md File Reference	411
24.198	samples/USB-AI16-16/android/README.md File Reference	411
24.199	samples/USB-AI16-16/java/extcal/README.md File Reference	411
24.200	samples/USB-AI16-16/java/read_channels_test/native-utils/README.md File Reference	411
24.201	samples/USB-AI16-16/java/read_channels_test/README.md File Reference	411
24.202	samples/USB-AI16-16/java/README.md File Reference	411
Index		412

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 [prerequisites](#) 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
- [Raspberry 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
```

Raspberry 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 source.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 -l 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 acquisition 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 aioub 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* [accessio DOT com](http://accessio.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 "-g" 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 library

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 using 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_acquire_test.c](#) Sample that 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 a 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 ADCCConfig 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 acquisition, 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-IIRO** 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,

```


[illegible]

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 **AIOWS** Sample - sample.cpp, receiver.cpp

AIOUSB Java Sample - Sample.java

5.7.2 Basic AIOWSB 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 – `AIOWUSB_Init()`, `AIOWUSB_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_StreamFrame()`, `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 -laiusb -lusb-1.0 -o sample g++ receiver.cpp -laiusb -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 (`aiusb.jar`) must be installed somewhere on your system. To compile the program, either use the supplied Makefile or use the command:

```
javac -cp ../../java/aiusb.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 Java 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/aiusb.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 -laiusb -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-A016-16A sample program version 1.13, 26 November 2009
AIOUSB library version 1.84, 22 December 2009
This program demonstrates controlling a USB-A016-16A 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-A016-16A
  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-A016-16A 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 (aiusb.jar) must be installed somewhere on your system. To compile the program, either use the supplied Makefile or use the command:

```
javac -cp ../java/aiusb.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 libaiouusb* . To do this you should have run "make" in the directories \$AIOUSB_ROOT/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 libaiouusb* . To do this you should have run "make" in the directories \$AIOUSB_ROOT/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 be 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
<div>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 <code>fxload -V</code> or <code>/sbin/fxload -V</code> 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.</div> <div>2. A recent version of perl is required, with the switch module installed.</div>		

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 -laiusb -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 usually 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 believe 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 memcopy, just cast directly

Parameters

<i>biter</i>	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 'aiousblnit = 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 [iirro_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 DIO_ReadAllToDIOBuf")

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

<i>DeviceIndex</i>	
<i>buf</i>	

Returns

Global [DIO_ReadIntoDIOBuf](#) (unsigned long DeviceIndex, [DIOBuf](#) *buf) ACCES_DEPRECATED("Please use DIO_ReadAllToDIOBuf")

You should use the function DIO_ReadAllToDIOBuf instead

Parameters

<i>DeviceIndex</i>	
<i>buf</i>	

Returns

Chapter 18

Namespace Index

18.1 Namespace List

Here is a list of all namespaces with brief descriptions:

AIOUSB	61
----------------------------------	----

Chapter 19

Hierarchical Index

19.1 Class Hierarchy

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

ad_gain_pairs	63
ADConfigBlock	63
ADRange	64
AI16_DataPoint	64
AI16_DataSet	67
aio_channel_range	70
aio_counts_converter	71
aio_either_val	71
aio_ret_value	72
AIOArgument	72
AIOArguments	73
AIOBuf	73
aiobuf_iterator	73
AIOChannelMask	74
AIOChannelRangeTmp	74
AIOCmd	75
AIOCommandLineOptions	75
AIOContinuousBuf	77
AIODeviceInfo	79
AIODeviceQuery	79
aioerror	80
AIOFifo	80
AIOGainRange	81
AIOProductGroup	81
AIOProductRange	81
aiusb_libusb_args	81
AIOUSBDevice	82
aiusboption	85
AIOWDGConfig	85
AnalogIORange	106
AI16_InputRange	69
AO16_OutputRange	116
DA12_OutputRange	132
BulkAcquireWorkerParams	117
channel_range	117
cJSON	118
cJSON_Hooks	118
config_options	119
configuration	119
Counter	120
CStringArray	126
DeviceInfo	133
DeviceProperties	134
DeviceSubsystem	135
AnalogInputSubsystem	86
AnalogOutputSubsystem	110
AO16_AnalogOutputSubsystem	111
DA12_AnalogOutputSubsystem	127
CounterSubsystem	123
DigitalIOSubsystem	136
DIOStreamSubsystem	142

- DIOBuf 141
- exception
 - Error 145
- invalid_argument
 - IllegalArgumentException 146
- lookup 147
- mux_settings 147
- new_aio_fifo 147
- options 148
- opts 149
- OutputVoltagePoint 151
- ProductIDName 152
- rangelookup 152
- runtime_error
 - OperationFailedException 148
- TestCaseSetup 153
- USBDevice 178
- USBDeviceBase 179
 - USB_AI16_Family 157
 - USB_AIO16_Family 159
 - USB_AO16_Family 162
 - USB_CTR_15_Family 165
 - USB_DA12_8A_Family 167
 - USB_DA12_8E_Family 169
 - USB_DIO_16_Family 171
 - USB_DIO_32_Family 174
 - USB_DIO_Family 176
- USBDeviceManager 184
- ushort_array 190
- vector
 - AI16_DataPointArray 66
 - BoolArray 117
 - CounterList 123
 - DoubleArray 145
 - IntArray 146
 - OutputVoltagePointArray 151
 - StringArray 152
 - UCharArray 157
 - USBDeviceArray 179
 - UShortArray 190

Chapter 20

Data Structure Index

20.1 Data Structures

Here are the data structures with brief descriptions:

ad_gain_pairs	63
ADCConfigBlock	63
ADRange	64
AI16_DataPoint	
Class AI16_DataPoint represents a single data point captured from a USB_AI16_Family device	64
AI16_DataPointArray	66
AI16_DataSet	
Class AI16_DataSet represents a data set captured from a USB_AI16_Family device	67
AI16_InputRange	69
aio_channel_range	70
aio_counts_converter	71
aio_either_val	71
aio_ret_value	72
AIOArgument	72
AIOArguments	73
AIOBuf	73
aiobuf_iterator	73
AIOChannelMask	74
AIOChannelRangeTmp	74
AIOCmd	75
AIOCommandLineOptions	75
AIOContinuousBuf	
AIOContinuousBuf provides a buffer that is used with the AIOUSB highspeed data acquisition API	77
AIODeviceInfo	79
AIODeviceQuery	79
aioerror	80
AIOFifo	
AIOFifo is a base class that is also instantiable for creating simple fifos for performing fast data acquisition	80
AIOGainRange	81
AIOProductGroup	
A smart product group that marks a range of ACCES I/O Products	81
AIOProductRange	
A simplified range of Products based off of device ids	81
aiousb_libusb_args	81
AIOUSBDevice	82
aiousboption	85
AIOWDGConfig	85
AnalogInputSubsystem	
Class AnalogInputSubsystem represents the analog input subsystem of a device	86
AnalogIORange	
Class AnalogIORange helps manage analog I/O range settings and provides voltage-count conversion utilities	106
AnalogOutputSubsystem	
Class AnalogOutputSubsystem is the superclass of the analog output subsystem of a device	110
AO16_AnalogOutputSubsystem	
Class AO16_AnalogOutputSubsystem represents the analog output subsystem of a device	111
AO16_OutputRange	116
BoolArray	117
BulkAcquireWorkerParams	117
channel_range	117
cJSON	118

cJSON_Hooks	118
config_options	119
configuration	119
Counter	
Class Counter represents a single counter/timer	120
CounterList	123
CounterSubsystem	
Class CounterSubsystem represents the counter/timer subsystem of a device	123
CStringArray	126
DA12_AnalogOutputSubsystem	
Class DA12_AnalogOutputSubsystem represents the analog output subsystem of a device	127
DA12_OutputRange	132
DeviceInfo	133
DeviceProperties	
Allows us to keep track of streaming (bulk) acquires without making the user keep track of the memory management	134
DeviceSubsystem	
Class DeviceSubsystem is the abstract super class for all device subsystems	135
DigitalIOSubsystem	
Class DigitalIOSubsystem represents the digital I/O subsystem of a device	136
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	141
DIOStreamSubsystem	
Class DIOStreamSubsystem represents the digital I/O streaming subsystem of a device	142
DoubleArray	145
Error	145
IllegalArgumentException	
Class IllegalArgumentException is thrown whenever an invalid argument is passed to a method	146
IntArray	146
lookup	147
mux_settings	147
new_aio_fifo	147
OperationFailedException	
Class OperationFailedException is thrown whenever an operation attempted on a device fails	148
options	148
opts	149
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	151
OutputVoltagePointArray	151
ProductIDName	152
rangelookup	152
StringArray	152
TestCaseSetup	153
UCharArray	157
USB_AI16_Family	157
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, 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	159
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_AO12_4A, USB_AO12_4	162
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	165
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	167
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	169

[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 171

[USB_DIO_32_Family](#)
Class [USB_DIO_32_Family](#) represents a USB-DIO-32-family device, which encompasses the following product IDs: USB_DIO_32 174

[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 176

[USBDevice](#) 178

[USBDeviceArray](#) 179

[USBDeviceBase](#)
Class [USBDeviceBase](#) is the abstract super class of all USB device families 179

[USBDeviceManager](#)
Class [USBDeviceManager](#) manages all the USB devices on the bus 184

[ushort_array](#) 190

[UShortArray](#) 190

Chapter 21

File Index

21.1 File List

Here is a list of all files with brief descriptions:

deprecated/classlib/AI16_DataPoint.cpp	191
deprecated/classlib/AI16_DataPoint.hpp	191
deprecated/classlib/AI16_DataSet.cpp	191
deprecated/classlib/AI16_DataSet.hpp	191
deprecated/classlib/AI16_InputRange.cpp	192
deprecated/classlib/AI16_InputRange.hpp	192
deprecated/classlib/AnalogInputSubsystem.cpp	
Class AnalogInputSubsystem implementation	192
deprecated/classlib/AnalogInputSubsystem.hpp	193
deprecated/classlib/AnalogIORange.cpp	193
deprecated/classlib/AnalogIORange.hpp	193
deprecated/classlib/AnalogOutputSubsystem.cpp	193
deprecated/classlib/AnalogOutputSubsystem.hpp	194
deprecated/classlib/AO16_AnalogOutputSubsystem.cpp	194
deprecated/classlib/AO16_AnalogOutputSubsystem.hpp	194
deprecated/classlib/AO16_OutputRange.cpp	194
deprecated/classlib/AO16_OutputRange.hpp	194
deprecated/classlib/Counter.cpp	
Class Counter implementation	195
deprecated/classlib/Counter.hpp	195
deprecated/classlib/CounterSubsystem.cpp	
Class CounterSubsystem implementation	196
deprecated/classlib/CounterSubsystem.hpp	196
deprecated/classlib/CppCommon.h	196
deprecated/classlib/DA12_AnalogOutputSubsystem.cpp	
Class DA12_AnalogOutputSubsystem implementation	196
deprecated/classlib/DA12_AnalogOutputSubsystem.hpp	197
deprecated/classlib/DA12_OutputRange.cpp	197
deprecated/classlib/DA12_OutputRange.hpp	197
deprecated/classlib/DeviceSubsystem.cpp	198
deprecated/classlib/DeviceSubsystem.hpp	198
deprecated/classlib/DigitalIOSubsystem.cpp	
Class DigitalIOSubsystem implementation	198
deprecated/classlib/DigitalIOSubsystem.hpp	198
deprecated/classlib/DIOStreamSubsystem.cpp	
Class DIOStreamSubsystem implementation	199
deprecated/classlib/DIOStreamSubsystem.hpp	199
deprecated/classlib/OutputVoltagePoint.hpp	199
deprecated/classlib/USB_AI16_Family.cpp	200
deprecated/classlib/USB_AI16_Family.hpp	200
deprecated/classlib/USB_AIO16_Family.cpp	201
deprecated/classlib/USB_AIO16_Family.hpp	201
deprecated/classlib/USB_AO16_Family.cpp	201
deprecated/classlib/USB_AO16_Family.hpp	201
deprecated/classlib/USB_CTR_15_Family.cpp	202
deprecated/classlib/USB_CTR_15_Family.hpp	202
deprecated/classlib/USB_DA12_8A_Family.cpp	202
deprecated/classlib/USB_DA12_8A_Family.hpp	202
deprecated/classlib/USB_DA12_8E_Family.cpp	203
deprecated/classlib/USB_DA12_8E_Family.hpp	203
deprecated/classlib/USB_DIO_16_Family.cpp	203
deprecated/classlib/USB_DIO_16_Family.hpp	203

deprecated/classlib/USB_DIO_32_Family.cpp	204
deprecated/classlib/USB_DIO_32_Family.hpp	204
deprecated/classlib/USB_DIO_Family.cpp	204
deprecated/classlib/USB_DIO_Family.hpp	204
deprecated/classlib/USBDeviceBase.cpp	205
deprecated/classlib/USBDeviceBase.hpp	205
deprecated/classlib/USBDeviceManager.cpp	
Class USBDeviceManager implementation	206
deprecated/classlib/USBDeviceManager.hpp	
Class USBDeviceManager, OperationFailedException, IllegalArgumentException declarations	206
lib/ADCCfgBlock.c	207
lib/ADCCfgBlock.h	211
lib/AIOBuf.c	214
lib/AIOBuf.h	215
lib/AIOChannelMask.c	217
lib/AIOChannelMask.h	219
lib/AIOChannelRange.c	222
lib/AIOChannelRange.h	223
lib/AIOCmd.c	224
lib/AIOCmd.h	
General structure for processing AIOUSB commands	224
lib/AIOCommandLine.c	225
lib/AIOCommandLine.h	228
lib/AIOConfiguration.c	232
lib/AIOConfiguration.h	233
lib/AIOContinuousBuffer.c	
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	234
lib/AIOContinuousBuffer.h	246
lib/AIOCountsConverter.c	
General header files for the AIOUSB library	255
lib/AIOCountsConverter.h	256
lib/AIODeviceInfo.c	258
lib/AIODeviceInfo.h	259
lib/AIODeviceQuery.c	
A simple structure for querying a USB card . This provides a simpler interface for more complicated queries going forward	260
lib/AIODeviceQuery.h	262
lib/AIODeviceTable.c	265
lib/AIODeviceTable.h	269
lib/AIOEither.c	272
lib/AIOEither.h	
General structure for AIOUSB Fifo	273
lib/AIOFifo.c	
General structure for AIOUSB Fifo	275
lib/AIOFifo.h	277
lib/AIOList.c	280
lib/AIOList.h	281
lib/AIOProductTypes.c	282
lib/AIOProductTypes.h	284
lib/AIOTuple.c	285
lib/AIOTuple.h	286
lib/AIOTypes.h	286
lib/aiousb.h	
General header files for the AIOUSB library	301
lib/AIOUSB_ADC.c	
Configuration functions for ADC elements	302
lib/AIOUSB_ADC.h	318
lib/AIOUSB_Core.c	
General header files for the AIOUSB library	327
lib/AIOUSB_Core.h	331
lib/AIOUSB_CTR.c	
Counter functionality	335
lib/AIOUSB_CTR.h	337
lib/AIOUSB_CustomEEPROM.c	
General header files for EEPROM functionality	338
lib/AIOUSB_CustomEEPROM.h	339

lib/AIOUSB_DAC.c	
Core code to handle DACs on AIOUSB devices	340
lib/AIOUSB_DAC.h	342
lib/AIOUSB_DIO.c	
Core code to interface with Digital cards	343
lib/AIOUSB_DIO.h	346
lib/AIOUSB_Log.c	347
lib/AIOUSB_Log.h	348
lib/AIOUSB_Properties.c	
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	350
lib/AIOUSB_Properties.h	352
lib/AIOUSB_USB.c	353
lib/AIOUSB_USB.h	354
lib/AIOUSB_WDG.c	354
lib/AIOUSB_WDG.h	355
lib/AIOUSBDevice.c	357
lib/AIOUSBDevice.h	358
lib/cJSON.c	359
lib/cJSON.h	361
lib/CStringArray.c	364
lib/CStringArray.h	364
lib/DIOBuf.c	
A smart buffer for handling Bit values and performing Bit arithmetic. This alleviates the need to perform bitwise operations on unsigned chars or other primitive data types in programming languages	365
lib/DIOBuf.h	368
lib/USBDevice.c	376
lib/USBDevice.h	378
lib/mocks/mock_aiocontbuf_get_data.c	371
lib/mocks/mock_aiocontbuf_get_data_arduino.c	371
lib/mocks/mock_capture_usb.c	
This file will allow capturing of all USB traffic, in and out	372
lib/mocks/mock_dio.c	374
lib/mocks/mock_usb_xfers.c	375
lib/wrappers/scilab/foo.c	379
lib/wrappers/scilab/foo.h	380
samples/TestLib/aiocommon.c	380
samples/TestLib/aiocommon.h	381
samples/TestLib/TestCaseSetup.cpp	382
samples/TestLib/TestCaseSetup.h	382
samples/USB-AI16-16/bulk_acquire_sample.c	384
samples/USB-AI16-16/burst_test.c	384
samples/USB-AI16-16/continuous_mode.c	385
samples/USB-AI16-16/continuous_mode_callback.c	385
samples/USB-AI16-16/continuous_mode_from_json_config.c	386
samples/USB-AI16-16/daitest.cpp	387
samples/USB-AI16-16/dio_sample.c	387
samples/USB-AI16-16/diotest.c	388
samples/USB-AI16-16/diotest2.cpp	388
samples/USB-AI16-16/read_channels_test.c	393
samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp	393
samples/USB-AI16-16/reverse_cal_table.cpp	389
samples/USB-AI16-16/sample.cpp	394
samples/USB-AI16-16/simp_test.cpp	395
samples/USB-AI16-16/simple_continuous_with_json.c	
Sample that demonstrates data ac	395
samples/USB-AI16-16/start_stop_continuous.c	396
samples/USB-AI16-16/test_fastscan.cpp	396
samples/USB-AI16-16/HOLD/dirktest.c	388
samples/USB-AI16-16/HOLD/julian_test.c	389
samples/USB-AI16-16/HOLD/reverse_cal_table.cpp	389
samples/USB-AI16-16/HOLD/sample_dio.c	390
samples/USB-AI16-16/HOLD/slow_receiver_test.cpp	390
samples/USB-AI16-16/HOLD/test.c	391
samples/USB-AI16-16/java/extcal/src/main/java/com/accesio/extcal.cpp	391
samples/USB-AI16-16/jni/read_channels_test.c	392
samples/USB-AO16-16/sample.cpp	394
samples/USB-ARB1/stream_test.c	397
samples/USB-DA12-8A/sample.cpp	394
samples/USB-DA12-8A/SampleClass.cpp	397
samples/USB-DIO-16/receiver.cpp	397
samples/USB-DIO-16/sample.cpp	394

samples/USB-DIO-16/[standalone_receiver.c](#) 398

samples/USB-DIO-32/[daisample.c](#) 398

samples/USB-DIO-32/[read_and_write_sample.c](#) 398

samples/USB-DIO-32/[sample3.c](#) 400

samples/USB-DIO-48/[read_and_write.c](#)

 AIOUSB sample program that writes and reads from a USB-DIO-48 401

samples/USB-DIO-96/[dio96_read_write.c](#) 402

samples/USB-DIO-96/[mytest.c](#) 403

samples/USB-DIO-96/[read_and_write_sample.c](#) 399

samples/USB-DIO-96/[tmp.c](#) 405

samples/USB-DIO-96/[write_sample.c](#) 406

samples/USB-IDIO-16_8/[idio_sample.c](#) 408

samples/USB-IDIO-16_8/[idio_sample2.c](#) 409

samples/USB-IDIO-16_8/[perftest.c](#) 410

samples/USB-IIRO-16_8/[iirro_sample.c](#) 410

Chapter 22

Namespace Documentation

22.1 AIOWS 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](#), [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](#).
- 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_AO12_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/[ADConfigBlock.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

<i>out</i>	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

<i>range</i>	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. The 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 an 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 <AIOTUSB_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/[AIOTUSB_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 with 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](#) [bDACDIOSStream](#)
- [AIOUSB_BOOL](#) [bDACSlowWaveStream](#)
- [AIOUSB_BOOL](#) [bDACDIOClock](#)
- unsigned [DACsUsed](#)
- [AIOUSB_BOOL](#) [bADCStream](#)
- unsigned [ADCCChannels](#)
- unsigned [ADCMUXChannels](#)
- unsigned char [RangeShift](#)
- unsigned [ADCCChannelsPerGroup](#)
number of A/D channels in each config.
- [AIOUSB_BOOL](#) [bDIOSStream](#)
- 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 bDIORed
- 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 [setTriggerMode\(int triggerMode \)](#)).

- 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
- Al16_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

<i>out</i>	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

<i>force</i>	<i>True</i> forces this class to interrogate the device anew; <i>false</i> 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

<i>range</i>	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

<i>autoConfig</i>	<i>True</i> enables automatically sending modified configuration, <i>false</i> 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

<i>discard</i>	<i>false</i> indicates that no samples will be discarded; <i>true</i> 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

<i>calMode</i>	the calibration mode. May be one of: AnalogInputSubsystem::CAL_MODE_NORMAL AnalogInputSubsystem::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

<i>triggerMode</i>	a bitwise OR of these flags: AnalogInputSubsystem::TRIG_MODE_CTR0_EXT AnalogInputSubsystem::TRIG_MODE_FALLING_EDGE AnalogInputSubsystem::TRIG_MODE_SCAN AnalogInputSubsystem::TRIG_MODE_EXTERNAL AnalogInputSubsystem::TRIG_MODE_TIMER
--------------------	--

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

IllegalArgumentException	
--	--

int getRange (int channel) const

Gets the current range for *channel*.

Parameters

<i>channel</i>	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

<i>startChannel</i>	the first channel for which to obtain the current range.
<i>numChannels</i>	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

<i>channel</i>	the channel for which to set the range.
<i>range</i>	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

<i>startChannel</i>	the first channel for which to set the range.
<i>range</i>	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

<i>channel</i>	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

<i>startChannel</i>	the first channel for which to obtain the current differential mode.
<i>numChannels</i>	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://accessio.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

<i>channel</i>	the channel for which to set differential or single-ended mode.
<i>differentialMode</i>	<i>false</i> selects single-ended mode; <i>true</i> 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

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

<i>overSample</i>	number of over-samples (0-255).
-------------------	---------------------------------

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

<i>IllegalArgumentException</i>	
---	--

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

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

AnalogInputSubsystem & setCalibrationTable (const UShortArray & *calTable*)

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

Parameters

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

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

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

<i>OperationFailedException</i>	
---	--

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

Sets the streaming block size.

Parameters

<i>blockSize</i>	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

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

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\(int startChannel, int num-Channels, int numSamples \)*](#)).

Parameters

<i>clockHz</i>	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

<i>autoCal</i>	<i>true</i> uses the internal voltage references to automatically calibrate the A/D; <i>false</i> generates a default (uncalibrated) table.
<i>returnCalTable</i>	<i>true</i> causes <i>calibrate()</i> to return the generated calibration table; <i>false</i> returns an empty table.
<i>saveFileName</i>	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

<i>OperationFailedException</i>	
---	--

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

<i>points</i>	array of voltage-count pairs to calibrate the A/D with.
<i>returnCalTable</i>	<i>true</i> causes calibrate() to return the generated calibration table; <i>false</i> returns an empty table.
<i>saveFileName</i>	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

<i>startChannel</i>	the first channel to read.
<i>numChannels</i>	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

<i>channel</i>	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 [set-Range\(int channel, int range \)](#)). 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

<i>startChannel</i>	the first channel to read.
---------------------	----------------------------

<i>numChannels</i>	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

<i>channel</i>	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

<i>startChannel</i>	the first channel to read.
<i>numChannels</i>	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

<i>startChannel</i>	the first channel to read.
<i>numChannels</i>	the number of channels to read.
<i>numSamples</i>	the total number of <i>samples</i> 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

<i>numSamples</i>	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

<i>method</i>	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

<i>channel</i>	the channel number to use for converting counts to volts.
<i>counts</i>	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 haven't changed since the count values were obtained.

Parameters

<i>startChannel</i>	the first channel number to use for converting counts to volts.
<i>counts</i>	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

<i>channel</i>	the channel number to use for converting volts to counts.
<i>volts</i>	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 haven't changed since the voltage values were obtained.

Parameters

<i>startChannel</i>	the first channel number to use for converting volts to counts.
<i>volts</i>	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

<i>minCounts</i>	minimum counts for current range.
<i>maxCounts</i>	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

<i>range</i>	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

<i>minCounts</i>	minimum counts for current range.
<i>maxCounts</i>	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

<i>minVolts</i>	minimum volts for current range.
<i>maxVolts</i>	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

<i>counts</i>	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

<i>volts</i>	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

<i>out</i>	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

<i>channel</i>	the channel to write to.
<i>counts</i>	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 0-0xffff, 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

<i>points</i>	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

<i>range</i>	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

<i>range</i>	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

--

<i>channel</i>	the channel to write to.
<i>volts</i>	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.

<i>points</i>	an array of OutputVoltagePoint points representing channel-voltage pairs.
---------------	---

Returns

This subsystem, useful for chaining together multiple operations.

IllegalArgumentException	
--	--

double countsToVolts (unsigned short counts) const [inline]

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

<i>counts</i>	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.

<i>volts</i>	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

<i>range</i>	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 <AIOTUSB_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/AIOTUSB_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 * [valuelstring](#)
- int [valueint](#)
- double [valuedouble](#)
- char * [string](#)

23.40.1 Field Documentation

struct cJSON* next

struct cJSON * prev

struct cJSON* child

int type

char* valuelstring

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

<i>mode</i>	the counter mode. May be one of: Counter::MODE_TERMINAL_COUNT Counter::MODE_ONE_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

<i>count</i>	the count value (0-65,535) to load into the counter.
--------------	--

Returns

This counter, useful for chaining together multiple operations.

Exceptions

<i>OperationFailedException</i>	
---	--

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

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

Parameters

<i>mode</i>	the counter mode (see <i>setMode(int mode)</i>).
<i>count</i>	the count value (0-65,535) to load into the counter.

Returns

This counter, useful for chaining together multiple operations.

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

unsigned short readCount ()

Reads a counter's current count value.

Returns

The current count value (0-65,535).

Exceptions

<i>OperationFailedException</i>	
---	--

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

<i>OperationFailedException</i>	
---	--

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

<i>mode</i>	the counter mode (see <i>setMode(int mode)</i>).
<i>count</i>	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 retriggeable 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 (retriggeable) (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

<i>out</i>	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

<i>counter</i>	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

<i>oldData</i>	<i>true</i> includes the "old data" indication in the returned data; <i>false</i> 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

<i>counter</i>	the counter to select as a gate (0 to n-1).
----------------	---

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

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

<i>counterBlock</i>	the counter block to use to generate the frequency.
<i>clockHz</i>	the desired output frequency (in Hertz).

Returns

The actual frequency that will be generated, limited by the device's capabilities.

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

CounterSubsystem& stopClock (int *counterBlock*) [inline]

Halts the counter started by [*startClock\(int counterBlock, double clockHz \)*](#)

Parameters

<i>counterBlock</i>	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](#) = 0xffff
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

<i>range</i>	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

<i>channel</i>	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

<i>startChannel</i>	the first channel for which to obtain the current range.
<i>numChannels</i>	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

<i>channel</i>	the channel for which to set the range.
<i>range</i>	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

<i>startChannel</i>	the first channel for which to set the range.
<i>range</i>	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

<i>range</i>	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

<i>channel</i>	the channel to write to.
<i>volts</i>	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

<i>points</i>	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

<i>channel</i>	the channel whose current range will be used to perform the conversion.
<i>counts</i>	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

<i>channel</i>	the channel whose current range will be used to perform the conversion.
<i>volts</i>	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

<i>range</i>	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

<i>out</i>	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

<i>tristate</i>	<i>true</i> causes all bits on the device to enter tristate (high-impedance) mode; <i>false</i> 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.
<i>outputs</i>	an array of boolean values, one per digital I/O <i>port</i> . Each <i>true</i> value in the array configures the entire corresponding I/O port as an output port; each <i>false</i> value configures the entire corresponding I/O port as an input port.
<i>values</i>	an array of boolean values, one per digital I/O <i>bit</i> , starting with bit 0 of the device (that is, the least significant bit on the lowest numbered port). Each <i>true</i> value in the array sets the corresponding output bit to a "1"; each <i>false</i> 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

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

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

<i>tristates</i>	an array of boolean values, one per tristate group. Each <i>true</i> value in the array puts the entire corresponding I/O port into tristate (high-impedance) mode; each <i>false</i> 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.
<i>outputs</i>	an array of boolean values, one per digital I/O <i>port</i> . Each <i>true</i> value in the array configures the entire corresponding I/O port as an output port; each <i>false</i> value configures the entire corresponding I/O port as an input port.
<i>values</i>	an array of boolean values, one per digital I/O <i>bit</i> , starting with bit 0 of the device (that is, the least significant bit on the lowest numbered port). Each <i>true</i> value in the array sets the corresponding output bit to a "1"; each <i>false</i> 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

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

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

<i>tristates</i>	an array of boolean values, one per tristate group, which will receive the current tristate mode of each tristate group. Each <i>true</i> value returned in the array indicates that the entire corresponding I/O port is in tristate (high-impedance) mode; each <i>false</i> value indicates that the entire corresponding I/O port is not in tristate mode. If this parameter is <i>null</i> , then the tristate configuration is not returned.
<i>outputs</i>	an array of boolean values, one per digital I/O <i>port</i> , which will receive the current output mode of each I/O port. Each <i>true</i> value returned in the array indicates that the entire corresponding I/O port is configured as an output port; each <i>false</i> value indicates that the entire corresponding I/O port is configured as an input port. If this parameter is <i>null</i> , then the output configuration is not returned.

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

bool read (int *channel*)

Reads a single digital input channel.

Parameters

<i>channel</i>	the channel to read.
----------------	----------------------

Returns

True indicates that the bit is set ("1"); *false* indicates that the bit is clear ("0").

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

BoolArray read (int *startChannel*, int *numChannels*)

Reads multiple digital input channels.

Parameters

<i>startChannel</i>	the first channel to read.
<i>numChannels</i>	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

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

DigitalIOSubsystem & write (int *channel*, bool *value*)

Writes a single digital output channel.

Parameters

<i>channel</i>	the channel to write.
<i>value</i>	the value to write to the specified channel. <i>True</i> sets the output bit to a "1" and <i>false</i> 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

<i>startChannel</i>	the first channel to write.
<i>values</i>	an array containing the values to write to the specified channels. For each channel, <i>true</i> sets the output bit to a "1" and <i>false</i> clears the output bit to a "0".

Returns

This subsystem, useful for chaining together multiple operations.

Exceptions

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

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 hexadecimal representations as well as use it for generating digital input, 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

<i>out</i>	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

<i>blockSize</i>	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

<i>IllegalArgumentException</i>	
<i>OperationFailedException</i>	

`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

<i>directionRead</i>	<i>true</i> sets read clock; <i>false</i> sets write clock.
<i>clockHz</i>	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

<i>OperationFailedException</i>	
---	--

`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

<i>OperationFailedException</i>	
---	--

`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

<i>directionRead</i>	<i>true</i> open the stream for reading; <i>false</i> 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

<i>numSamples</i>	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

<i>values</i>	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

<i>method</i>	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

<i>message</i>	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

<i>result</i>	AIOUSB module result code.
---------------	--

OperationFailedException (const std::string & *message*) [inline]

Constructs the exception from a simple string message.

Parameters

<i>message</i>	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* adccconfig_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

<i>channel</i>	the channel number to output voltage to.
<i>volts</i>	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 <AIOTUSB_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

<i>result</i>	
<i>linnum</i>	

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 *blk_size*, unsigned int *ovr_sampl*, unsigned int *clk_speed*)

Demonstrate bulk acquire.

Parameters

<i>blk_size</i>	
<i>ovr_sampl</i>	
<i>clk_speed</i>	

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 [USB_AI16_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_AI16_Family](#) instance. You can then cast the [USBDeviceBase](#)
* reference obtained from one of those methods to a [USB_AI16_Family](#) and make use of this class' methods, 1
*

```
USBDeviceArray devices = deviceManager.getDeviceByProductID( USB\_AI12\_32A, USB\_AI12\_32E );  
if( devices.size() > 0 )  
  USB\_AI16\_Family &device = *( USB\_AI16\_Family * ) devices.at( 0 );
```


*

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

<i>productID</i>	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

<i>out</i>	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: 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.

#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](#) analogOutputSubsystem
- [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

<i>productID</i>	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

<i>out</i>	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 analogOutputSubsystem [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

<i>productID</i>	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

<i>out</i>	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

<i>productID</i>	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

<i>out</i>	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

<i>productID</i>	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

<i>out</i>	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

<i>productID</i>	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

<i>out</i>	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

<i>productID</i>	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

<i>out</i>	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

<i>productID</i>	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

<i>out</i>	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_IIR-O4_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_IIR-O4_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::getDeviceBy-ProductID\(int productID \) const](#), to obtain a reference to a [USB_DIO_Family](#) instance. You can then cast the [USB-DeviceBase](#) 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

<i>productID</i>	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

<i>out</i>	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, ADCCConfigBlock *configBlock)`
- `int(* usb_get_config)(USBDevice *usb, ADCCConfigBlock *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

<i>out</i>	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

<i>timeout</i>	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

<i>address</i>	starting address from 0x000 to 0x1FF within the EEPROM.
<i>data</i>	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

<i>address</i>	starting address from 0x000 to 0x1FF within the EEPROM.
<i>numBytes</i>	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

<i>out</i>	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

<i>productID</i>	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

<i>productID</i>	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. [ProductIDToName\(\)](#) 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

<i>productName</i>	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 [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

<i>productName</i>	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

<i>result</i>	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

<i>productID</i>	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

<i>minProductID</i>	the minimum product ID to search for.
<i>maxProductID</i>	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

<i>productIDs</i>	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

<i>serialNumber</i>	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 <aiusb.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, 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.

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_AO12_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: USB_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 <AIOTUSB_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 <aiusb.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, [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)
- [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_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)
- 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

<i>config</i>	
<i>deviceDesc</i>	
<i>size</i>	

```
AIORET_TYPE ADCConfigBlockInitForCounterScan ( ADCConfigBlock * config, AIOUSBDevice * deviceDesc )
```

Parameters

<i>config</i>	
<i>deviceDesc</i>	

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
*     - 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](#) [ADCConfigBlock](#)

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*, [ADCCalMode](#) *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 ADConfigBlock

24.74.2 Function Documentation

[AIORET_TYPE ADCConfigBlockInit](#) ([ADCConfigBlock](#) * config, [AIOUSBDevice](#) * deviceDesc, unsigned size)

Parameters

<i>config</i>	
<i>deviceDesc</i>	
<i>size</i>	

[AIORET_TYPE ADCConfigBlockInitForCounterScan](#) ([ADCConfigBlock](#) * config, [AIOUSBDevice](#) * deviceDesc)

Parameters

<i>config</i>	
<i>deviceDesc</i>	

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
*     - 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 iterator.

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 iterator.

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

<i>biter</i>	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 iterator.
- [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 memcopy, just cast directly

Parameters

<i>biter</i>	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 iterator 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

<i>number_ - channels</i>	The number of bits in our Bit Mask
---------------------------	------------------------------------

`void DeleteAIOChannelMask (AIOChannelMask * mask)`

Destructor for the [AIOChannelMask](#) object.

Parameters

<i>mask</i>	AIOChannelMask to delete
-------------	--

`AIORET_TYPE AIOChannelMaskIndices (AIOChannelMask * mask, int * pos)`

Returns an iterator 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

<i>obj</i>	The AIOChannelMask bit mask object
<i>*tmp</i>	The object we save the BitMask to
<i>index</i>	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

<i>obj</i>	
------------	--

Returns

`AIORET_TYPE AIOChannelMaskNumberSignals (AIOChannelMask * obj)`

`AIORET_TYPE AIOChannelMaskSetMaskFromStr (AIOChannelMask * obj, const char * bitfields)`

Rely on the base type to determine the sizes.

Parameters

<i>obj</i>	
<i>bitfields</i>	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

<i>bitfields</i>	
------------------	--

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

<i>obj</i>	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

<i>obj</i>	AIOChannelMask bit mask object
<i>index</i>	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 iterator 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

<i>number_channels</i>	The number of bits in our Bit Mask
------------------------	------------------------------------

void [DeleteAIOChannelMask](#) ([AIOChannelMask](#) * *mask*)

Destructor for the [AIOChannelMask](#) object.

Parameters

<i>mask</i>	AIOChannelMask to delete
-------------	--

AIOChannelMask* [NewAIOChannelMaskFromStr](#) (const char * *bitfields*)

Creates a new [AIOChannelMask](#) object from a character string of 1's and 0's.

Parameters

<i>bitfields</i>	
------------------	--

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

<i>obj</i>	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

<i>obj</i>	AIOChannelMask bit mask object
<i>index</i>	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

<i>obj</i>	The AIOChannelMask bit mask object
* <i>tmp</i>	The object we save the BitMask to
<i>index</i>	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

<i>obj</i>	
------------	--

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 iterator 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

<i>obj</i>	
<i>bitfields</i>	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](#) [AIOCommandLineOverrideADCCConfigBlock](#) ([ADCCConfigBlock](#) *config, [AIOCommandLineOptions](#) *options)
- [AIORET_TYPE](#) [AIOCommandLineOptionsOverrideADCCConfigBlock](#) ([ADCCConfigBlock](#) *config, [AIOCommandLineOptions](#) *options)
Allows the [AIOCommandLineOptions](#) options to override the settings in the [ADCCConfigBlock](#).
- [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

<i>options</i>	
<i>argc</i>	Pointer to number of arguments in argv.
<i>argv</i>	An array of strings

Returns

```
AIORET_TYPE AIOProcessCmdline ( AIOCommandLineOptions * options, int argc, char ** argv )
```

A simplified command line parsing library for.

Parameters

<i>options</i>	AIOCommandLineOptions object that holds overridden cmd line options
<i>argc</i>	Number of command line arguments
<i>argv</i>	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

<i>argc</i>	Number of command line arguments
<i>argv</i>	Array of strings to the command line arguments
<i>options</i>	

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

<i>options</i>	
----------------	--

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

<i>options</i>	AIOCommandLineOptions object
<i>indices</i>	Array of devices found
<i>num_devices</i>	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

<i>config</i>	ADCConfigBlock object that is written to the AIOUSB device
<i>options</i>	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

<i>options</i>	AIOCommandLineOptions object that holds overridden cmd line options
<i>argc</i>	Number of command line arguments
<i>argv</i>	Array of strings to the command line arguments

Returns

AIORET_TYPE AIOProcessCommandLine (AIOCommandLineOptions * options, int * argc, char ** argv)

Parameters

<i>options</i>	
<i>argc</i>	Pointer to number of arguments in argv.
<i>argv</i>	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

<i>argc</i>	Number of command line arguments
<i>argv</i>	Array of strings to the command line arguments
<i>options</i>	

AIORET_TYPE DeleteAIOCommandLineOptions (AIOCommandLineOptions * options)

A Descructor for the allocated [AIOCommandLineOptions](#).

Parameters

<i>options</i>	
----------------	--

Returns

AIORET_TYPE AIOCommandLineOptionsListDevices (**AIOCommandLineOptions** * *options*, intlist * *indices*)

Lists any devices that were matched and then lists which index was specified.

Parameters

<i>options</i>	AIOCommandLineOptions object
<i>indices</i>	Array of devices found
<i>num_devices</i>	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

<i>config</i>	ADCConfigBlock object that is written to the AIOUSB device
<i>options</i>	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](#) [AIOContinuousBufConfiguration](#)

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 AIOContinuousBufConfiguration
```

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 [AIOContinuousBufForceTerminateAcquisitionOverrun](#) (AIOContinuousBuf *buf)
- AIORET_TYPE [AIOContinuousBufForceTerminateAcquisition](#) (AIOContinuousBuf *buf)
- AIOContinuousBuf * [NewAIOContinuousBufForCounts](#) (unsigned long DeviceIndex, unsigned scantcounts, 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 scantcounts, 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 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](#)(*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, [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](#))

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://accessio.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 aioubsh header files can be located:

```
export CPATH=/usr/local/include/libusb-1.0/:/usr/local/include/aioubsh/
```

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 -laioubsh -lusb-1.0 -o sample
*
* or, to enable debug features
*
*     g++ -ggdb sample.cpp -laioubshdbg -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

<i>object</i>	
---------------	--

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 AIOContinuousBufForceTerminateAcquisitionOvrrun (AIOContinuousBuf * *buf*)

AIORET_TYPE AIOContinuousBufForceTerminateAcquisition (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

<i>deviceIndex</i>	
<i>num_channels</i>	
<i>num_oversamples</i>	
<i>base_size</i>	

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 AIOContinuousBufInitADConfigBlock (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

<i>buf</i>	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)

ADCCConfigBlock* AIOContinuousBufGetADCCConfigBlock (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

<i>buf</i>	
<i>read_buf</i>	
<i>tmpbuffer_size</i>	
<i>size</i>	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

-

<i>buf</i>	
<i>read_buf</i>	
<i>tmpbuffer_size</i>	
<i>num_scans</i>	

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

<i>buf</i>	
------------	--

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

-

<i>buf</i>	
------------	--

Returns

status code of start.

AIORET_TYPE AIOContinuousBufStopAcquisition (**AIOContinuousBuf** * *buf*)

AIORET_TYPE AIOContinuousBufSetChannelMask (**AIOContinuousBuf** * *buf*, **AIOChannelMask** * *mask*)

Sets the channel mask.

Parameters

<i>buf</i>	
<i>mask</i>	

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

<i>buf</i>	
<i>writebuf</i>	
<i>wrbufsize</i>	
<i>size</i>	
<i>flag</i>	

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

<i>buf</i>	
<i>cmd</i>	
<i>callback</i>	

Returns

>= 0 if successful, < 0 if failure

AIORET_TYPE AIOContinuousBufCallbackStart (AIOContinuousBuf * buf)

Setups the Automated runs for continuous mode runs.

Parameters

<i>buf</i>	
------------	--

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

<i>buf</i>	
<i>readbuf</i>	
<i>readbufsize</i>	
<i>size</i>	

Returns

If number is positive, it is the number of bytes that have been read.

AIORET_TYPE AIOContinuousBufLock (AIOContinuousBuf * buf)

Parameters

<i>buf</i>	
------------	--

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 scantcounts, unsigned num_channels)
- [AIOContinuousBuf * NewAIOContinuousBufForVolts](#) (unsigned long DeviceIndex, unsigned scantcounts, 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 readbuf-size, 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 data-size, 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 an 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

<i>deviceIndex</i>	
<i>num_channels</i>	
<i>num_oversamples</i>	
<i>base_size</i>	

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

<i>buf</i>	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

<i>buf</i>	
------------	--

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*)

ADCCfgBlock* AIOContinuousBufGetADCCfgBlock (**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

<i>buf</i>	
<i>mask</i>	

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

<i>buf</i>	
------------	--

Returns

```
AIORET_TYPE AIOContinuousBufUnlock ( AIOContinuousBuf * buf )

AIORET_TYPE AIOContinuousBufCallbackStart ( AIOContinuousBuf * buf )
```

Setups the Automated runs for continuous mode runs.

Parameters

<i>buf</i>	
------------	--

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

<i>buf</i>	
<i>cmd</i>	
<i>callback</i>	

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

- | | |
|-----------------------|--|
| <i>buf</i> | |
| <i>read_buf</i> | |
| <i>tmpbuffer_size</i> | |

<i>size</i>	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

- | | |
|-----------------------|--|
| <i>buf</i> | |
| <i>read_buf</i> | |
| <i>tmpbuffer_size</i> | |
| <i>num_scans</i> | |

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

<i>buf</i>	
<i>readbuf</i>	
<i>readbufsize</i>	
<i>size</i>	

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

<i>buf</i>	
<i>writebuf</i>	
<i>wrbufsize</i>	
<i>size</i>	
<i>flag</i>	

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](#) * [NewAIOGainRangeFromADCCConfigBlock](#) ([ADCCConfigBlock](#) *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

<i>cc</i>	Counts converter object
<i>tobufptr</i>	ToFifo (double)
<i>frombufptr</i>	From Fifo (unsigned short)
<i>num_counts</i>	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

<i>DeviceIndex</i>	
--------------------	--

Returns

[AIODeviceQuery](#) * object

AIORET_TYPE DeleteAIODeviceQuery ([AIODeviceQuery](#) * *devq*)

Destructor for [AIODeviceQuery](#) *.

Parameters

<i>devq</i>	AIODeviceQuery *
-------------	----------------------------------

Returns

>= 0, success, otherwise error

char* AIODeviceQueryToStr ([AIODeviceQuery](#) * *devq*)

Converts the [AIODeviceQuery](#) into a string representation.

Parameters

<i>devq</i>	AIODeviceQuery *
-------------	----------------------------------

Returns

String representing the Device query, NULL if not defined

char* AIODeviceQueryToRepr ([AIODeviceQuery](#) * *devq*)

Repr version of this product.

Parameters

<i>devq</i>	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

<i>devq</i>	AIODeviceQuery *
-------------	----------------------------------

Returns

>= 0, the product ID in question, otherwise error

AIORET_TYPE AIODeviceQueryGetIndex ([AIODeviceQuery](#) * *devq*)

Returns the Index associated with the [AIODeviceQuery](#).

Parameters

<i>devq</i>	AIODeviceQuery *
-------------	----------------------------------

Returns

>= 0 index , otherwise error

AIORET_TYPE AIODeviceQueryNameSize ([AIODeviceQuery](#) * *devq*)

Returns the strlenght of the Device name of the device in question.

Parameters

<i>devq</i>	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

<i>devq</i>	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

<i>devq</i>	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

<i>devq</i>	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

<i>DeviceIndex</i>	
--------------------	--

Returns

[AIODeviceQuery](#) * object

[AIORET_TYPE](#) [DeleteAIODeviceQuery](#) ([AIODeviceQuery](#) * *devq*)

Destructor for [AIODeviceQuery](#) *.

Parameters

<i>devq</i>	AIODeviceQuery *
-------------	----------------------------------

Returns

>= 0, success, otherwise error

char* AIODeviceQueryToStr (AIODeviceQuery * *devq*)

Converts the [AIODeviceQuery](#) into a string representation.

Parameters

<i>devq</i>	AIODeviceQuery *
-------------	----------------------------------

Returns

String representing the Device query, NULL if not defined

char* AIODeviceQueryToRepr (AIODeviceQuery * *devq*)

Repr version of this product.

Parameters

<i>devq</i>	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

<i>devq</i>	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

<i>devq</i>	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

<i>devq</i>	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

<i>devq</i>	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

<i>devq</i>	AIODeviceQuery *
-------------	----------------------------------

Returns

!= 0 the name of the card, otherwise an error

AIORET_TYPE [AIODeviceQueryGetIndex](#) (**AIODeviceQuery** * *devq*)

Returns the Index associated with the [AIODeviceQuery](#).

Parameters

<i>devq</i>	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 [aiousblnit](#) = 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

<i>DeviceIndex</i>	
<i>pPID</i>	
<i>pNameSize</i>	
<i>pName</i>	
<i>pDIOBytes</i>	
<i>pCounters</i>	

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

<i>name</i>	
-------------	--

Returns

productNameIndex[] represents an index into productIDNameTable[], sorted by product name (see notes for [ProductIDToName\(\)](#))

<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

	<i>DeviceIndex</i>	Device index we are probing
out	<i>res</i>	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

<i>DeviceIndex</i>	
--------------------	--

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

<i>numDevices</i>	
-------------------	--

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

	<i>DeviceIndex</i>	Device index we are probing
out	<i>res</i>	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

<i>DeviceIndex</i>	
<i>pPID</i>	
<i>pNameSize</i>	
<i>pName</i>	
<i>pDIOBytes</i>	
<i>pCounters</i>	

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 ProductIDToName())

<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

AIUSB_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

<i>numDevices</i>	
-------------------	--

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) aioreset_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) aioreset_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 0xffffffffffff

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](#) 0xffffffffffff

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_ao_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 with 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_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)

24.103.1 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_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)

24.104.1 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 groupecpy ( 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* groupecpy (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__CStringArray_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 <= [LAST_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_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_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](#), __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_AIOCCHANNELMASK](#), __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_AIOCONTINUOUS_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](#) = 0xFFFFFFFFEul, [diOnly](#) = 0xFFFFFFFFDul, [diNone](#) = 0xFFFFFFFFFul }
 - 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](#), [CLEAR_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](#), [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](#) }
- 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, [AD_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_OFRRQ = 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_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, __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_AIOCHANNELMASK, __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_AIOCONTINUOUS_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

USB_CTR_15
USB_IIRO4_2SM
USB_IIRO4_COM
USB_DIO16RO8
USB_DIO48DO24
USB_DIO24DO12
USB_DO24
PICO_DIO16RO8
USBP_I18IDO4A
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_OFRR
AUR_CTR_COS_BULK_ABORT
AUR_CTR_PUR_MFRR
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/aiusb.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](#) ([ADConfigBlock](#) *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 num-Channels, const unsigned short counts[], double volts[])
- [PRIVATE AIORET_TYPE AIOUSB_ArrayVoltsToCounts](#) (unsigned long DeviceIndex, int startChannel, int num-Channels, 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 *Config-BufSize)
Determine information 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 *Config-BufSize)
- 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 bSingle-Ended)
- 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 End-Channel)
- 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 *Config-BufSize, const char *CalFileName)
Determine information about the device found at a specific DeviceIndex.
- unsigned long [ADC_BulkAcquire](#) (unsigned long DeviceIndex, unsigned long BufSize, void *pBuf)
Determine information 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)
- [AIORRESULT ADC_ClearFastITConfig](#) (unsigned long DeviceIndex)

Frees memory associated with the FastConfig Config blocks.

- unsigned long [ADC_CreateADBuf](#) ([AIOUSBDevice](#) *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.

- [AIORRESULT AIOUSB_SetRangeSingle](#) ([ADConfigBlock](#) *config, unsigned long channel, unsigned char gain-Code)
- [AIORRET_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)
- [AIORRET_TYPE AIOUSB_SetAllGainCodeAndDiffMode](#) ([ADConfigBlock](#) *config, unsigned gainCode, [AIOUSB_BOOL](#) differentialMode)
- [AIORRET_TYPE AIOUSB_GetGainCode](#) (const [ADConfigBlock](#) *config, unsigned channel)
- [AIORRET_TYPE AIOUSB_SetGainCode](#) ([ADConfigBlock](#) *config, unsigned channel, unsigned gainCode)
- [AIORRET_TYPE AIOUSB_IsDifferentialMode](#) (const [ADConfigBlock](#) *config, unsigned channel)
- [AIORRET_TYPE AIOUSB_SetDifferentialMode](#) ([ADConfigBlock](#) *config, unsigned channel, [AIOUSB_BOOL](#) differentialMode)
- [AIORRET_TYPE AIOUSB_GetCalMode](#) (const [ADConfigBlock](#) *config)
- [AIORRET_TYPE AIOUSB_SetCalMode](#) ([ADConfigBlock](#) *config, unsigned calMode)
- unsigned [AIOUSB_GetTriggerMode](#) (const [ADConfigBlock](#) *config)
- [AIORRET_TYPE AIOUSB_SetTriggerMode](#) ([ADConfigBlock](#) *config, unsigned triggerMode)
- unsigned [AIOUSB_GetStartChannel](#) (const [ADConfigBlock](#) *config)
- unsigned [AIOUSB_GetEndChannel](#) (const [ADConfigBlock](#) *config)
- [AIORRET_TYPE AIOUSB_SetScanRange](#) ([ADConfigBlock](#) *config, unsigned startChannel, unsigned end-Channel)
- [AIORRET_TYPE AIOUSB_GetOversample](#) ([ADConfigBlock](#) *config)
- [AIORRET_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 (ADConfigBlock * *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

<i>DeviceIndex</i>	
<i>forceRead</i>	

unsigned long WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

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

<i>DeviceIndex</i>	
<i>groundCounts</i>	
<i>referenceCounts</i>	

PRIVATE AIORET_TYPE AIOUSB_GetScan (unsigned long *DeviceIndex*, unsigned short *counts*[])

Performs a scan and averages the voltage values.

Parameters

<i>DeviceIndex</i>	
<i>counts</i>	

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

<i>DeviceIndex</i>	
<i>startChannel</i>	
<i>numChannels</i>	
<i>counts</i>	
<i>volts</i>	

Returns

PRIVATE AIORET_TYPE AIOUSB_ArrayVoltsToCounts (unsigned long *DeviceIndex*, int *startChannel*, int *numChannels*, const double *volts*[], unsigned short *counts*[])

Parameters

<i>DeviceIndex</i>	
<i>startChannel</i>	
<i>numChannels</i>	
<i>volts</i>	
<i>counts</i>	

Returns

unsigned short AIOUSB_VoltsToCounts (unsigned long *DeviceIndex*, unsigned *channel*, double *volts*)

Parameters

<i>DeviceIndex</i>	
<i>channel</i>	
<i>volts</i>	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex*, unsigned long *ChannelIndex*, double * *singlevoltage*)

Read one voltage input's current value.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
<i>ChannelIndex</i>	number indicating which channel's data you wish to get
<i>singlevoltage</i>	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

<i>DeviceIndex</i>	
<i>pBuf</i>	

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

<i>DeviceIndex</i>	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
<i>pBuf</i>	Pointer to an array of WORDs. Each element in the array will receive the value read from the corresponding A/D input channel. The array must be at least as large as the number 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 WORDs. Only elements 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 remain 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*)

Determine information about the device found at a specific DeviceIndex.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
<i>ConfigBuf</i>	a pointer to the first of an array of bytes for configuration data
<i>ConfigBufSize</i>	a pointer to a variable holding the number of configuration bytes to read. Will be set to the number 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

<i>DeviceIndex</i>	
<i>pConfigBuf</i>	
<i>ConfigBufSize</i>	

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

<i>DeviceIndex</i>	
<i>config</i>	

Returns

validate settings

unsigned long ADC_RangeAll (unsigned long *DeviceIndex*, unsigned char * *pGainCodes*, unsigned long *bSingleEnded*)

Parameters

<i>DeviceIndex</i>	
<i>pGainCodes</i>	
<i>bSingleEnded</i>	

Returns

unsigned long ADC_Range1 (unsigned long *DeviceIndex*, unsigned long *ADChannel*, unsigned char *GainCode*, unsigned long *bSingleEnded*)

Parameters

<i>DeviceIndex</i>	
<i>ADChannel</i>	
<i>GainCode</i>	
<i>bSingleEnded</i>	

Returns

unsigned long ADC_ADMode (unsigned long *DeviceIndex*, unsigned char *TriggerMode*, unsigned char *CalMode*)

Parameters

<i>DeviceIndex</i>	
<i>TriggerMode</i>	
<i>CalMode</i>	

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex*, unsigned char *Oversample*)

Parameters

<i>DeviceIndex</i>	
<i>Oversample</i>	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

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

<i>ProductID</i>	produc we are looking up
<i>num_channels</i>	Number of channels we will be sampling on
<i>num_oversamples</i>	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

<i>DeviceIndex</i>	
<i>StartChannel</i>	
<i>EndChannel</i>	

Returns

unsigned long ADC_SetCal (unsigned long *DeviceIndex*, const char * *CalFileName*)

Parameters

<i>DeviceIndex</i>	
<i>CalFileName</i>	

Returns

AIOUSB_BOOL ADC_CanCalibrate (unsigned long *productID*)

unsigned long ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

Returns

unsigned long ADC_Initialize (unsigned long *DeviceIndex*, unsigned char * *pConfigBuf*, unsigned long * *ConfigBufSize*, const char * *CalFileName*)

Determine information about the device found at a specific DeviceIndex.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
<i>pConfigBuf</i>	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
<i>ConfigBufSize</i>	a pointer to a variable holding the number of configuration bytes to write.
<i>CalFileName</i>	the file name of a calibration file, or a command 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 information about the device found at a specific DeviceIndex.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
<i>BufSize</i>	the size, in bytes, of the buffer to receive the data
<i>pBuf</i>	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return immediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or moved. 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

<i>DeviceIndex</i>	
--------------------	--

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

<i>DeviceIndex</i>	
<i>BytesLeft</i>	

Returns

unsigned long ADC_CreateFastITConfig (unsigned long *DeviceIndex*, int *size*)

Creates FastIT Config Blocks.

Parameters

<i>DeviceIndex</i>	
<i>size</i>	

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

<i>DeviceIndex</i>	
--------------------	--

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
*     - 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

<i>DeviceIndex</i>	
<i>pData</i>	buffer we will write data into

Returns

unsigned long ADC_GetITScanV (unsigned long *DeviceIndex*, double * *pBuf*)

Parameters

<i>DeviceIndex</i>	
<i>pBuf</i>	

Returns

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex*, AIOUSB_BOOL *discard*)

Parameters

<i>DeviceIndex</i>	
<i>discard</i>	

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

<i>deviceIndex</i>	
--------------------	--

Returns

void DoLoadCalTable (unsigned short *const *calTable*, unsigned long *DeviceIndex*, double *groundCounts*, double *referenceCounts*)

Loads the Cal table for Automatic internal calibration.

Parameters

<i>calTable</i>	
<i>DeviceIndex</i>	
<i>groundCounts</i>	
<i>referenceCounts</i>	

AIORESULT AIOUSB_SetRangeSingle (ADConfigBlock * *config*, unsigned long *channel*, unsigned char *gainCode*)

Parameters

<i>config</i>	
<i>channel</i>	
<i>gainCode</i>	

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

<i>DeviceIndex</i>	
<i>autoCal</i>	
<i>returnCalTable</i>	
<i>saveFileName</i>	

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

<i>config</i>	
<i>channel</i>	
<i>gainCode</i>	

AIORET_TYPE AIOUSB_IsDifferentialMode (const ADConfigBlock * *config*, unsigned *channel*)

Parameters

<i>config</i>	
<i>channel</i>	

Returns

AIORET_TYPE AIOUSB_SetDifferentialMode (ADConfigBlock * *config*, unsigned *channel*, AIOUSB_BOOL *differentialMode*)

Parameters

<i>config</i>	
<i>channel</i>	
<i>differentialMode</i>	

AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock * *config*, unsigned *calMode*)

Parameters

<i>config</i>	
<i>calMode</i>	

unsigned AIOUSB_GetTriggerMode (const ADConfigBlock * *config*)

Parameters

<i>config</i>	
---------------	--

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

<i>config</i>	
<i>startChannel</i>	
<i>endChannel</i>	

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/AIUSB_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 AIUSB_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)
Determine information 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)
- [AIUSB_BOOL ADC_CanCalibrate](#) (unsigned long ProductID)
- [AIORESULT ADC_Initialize](#) (unsigned long DeviceIndex, unsigned char *pConfigBuf, unsigned long *ConfigBufSize, const char *CalFileName)
Determine information about the device found at a specific DeviceIndex.
- [AIORESULT ADC_BulkAcquire](#) (unsigned long DeviceIndex, unsigned long BufSize, void *pBuf)
Determine information 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

<i>DeviceIndex</i>	
<i>pBuf</i>	

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

<i>DeviceIndex</i>	
<i>pGainCodes</i>	
<i>bSingleEnded</i>	

Returns

AIORET_TYPE ADC_GetChannelV (unsigned long *DeviceIndex*, unsigned long *ChannelIndex*, double * *singlevoltage*)

Read one voltage input's current value.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
<i>ChannelIndex</i>	number indicating which channel's data you wish to get
<i>singlevoltage</i>	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

<i>DeviceIndex</i>	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
<i>pBuf</i>	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

<i>DeviceIndex</i>	
<i>counts</i>	

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*)

Determine information about the device found at a specific DeviceIndex.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to query; generally either diOnly or a specific device's Device Index.
<i>ConfigBuf</i>	a pointer to the first of an array of bytes for configuration data
<i>ConfigBufSize</i>	a pointer to a variable holding the number of configuration bytes to read. Will be set to the number of configuration bytes read

Returns

AIORESULT ADC_SetConfig (unsigned long *DeviceIndex*, unsigned char * *pConfigBuf*, unsigned long * *ConfigBufSize*)

Parameters

<i>DeviceIndex</i>	
<i>pConfigBuf</i>	
<i>ConfigBufSize</i>	

Returns

validate settings

AIORESULT ADC_Range1 (unsigned long *DeviceIndex*, unsigned long *ADChannel*, unsigned char *GainCode*, unsigned long *bSingleEnded*)

Parameters

<i>DeviceIndex</i>	
<i>ADChannel</i>	
<i>GainCode</i>	
<i>bSingleEnded</i>	

Returns

AIORESULT ADC_ADMode (unsigned long *DeviceIndex*, unsigned char *TriggerMode*, unsigned char *CalMode*)

Parameters

<i>DeviceIndex</i>	
<i>TriggerMode</i>	
<i>CalMode</i>	

Returns

AIORESULT ADC_SetScanLimits (unsigned long *DeviceIndex*, unsigned long *StartChannel*, unsigned long *EndChannel*)

Parameters

<i>DeviceIndex</i>	
<i>StartChannel</i>	
<i>EndChannel</i>	

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

<i>ProductID</i>	produc we are looking up
<i>num_channels</i>	Number of channels we will be sampling on
<i>num_oversamples</i>	Number of oversamples we will be using

Returns

AIORESULT ADC_ClockRateForADCProduct (unsigned long *ProductID*)

AIORESULT ADC_SetCal (unsigned long *DeviceIndex*, const char * *CalFileName*)

Parameters

<i>DeviceIndex</i>	
<i>CalFileName</i>	

Returns

AIORESULT ADC_QueryCal (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

Returns

AIOUSB_BOOL `ADC_CanCalibrate (unsigned long ProductID)`

AIORESULT `ADC_Initialize (unsigned long DeviceIndex, unsigned char * pConfigBuf, unsigned long * ConfigBufSize, const char * CalFileName)`

Determine information about the device found at a specific DeviceIndex.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
<i>pConfigBuf</i>	A pointer an array of configuration bytes, identical to that used in ADC_SetConfig()
<i>ConfigBufSize</i>	a pointer to a variable holding the number of configuration bytes to write.
<i>CalFileName</i>	the filename of a calibration file, or a command 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 information about the device found at a specific DeviceIndex.

Parameters

<i>DeviceIndex</i>	DeviceIndex of the card you wish to control; generally either diOnly or a specific device's Device Index.
<i>BufSize</i>	the size, in bytes, of the buffer to receive the data
<i>pBuf</i>	a pointer to the buffer in which to receive data

Returns

AIOUSB_SUCCESS indicates success, failure otherwise

Note

This function will return immediately. A return value of AIOUSB_SUCCESS indicates that bulk data is being acquired in the background, and the buffer should not be deallocated or moved. 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

<i>DeviceIndex</i>	
<i>BytesLeft</i>	

Returns

AIORESULT `ADC_InitFastITScanV (unsigned long DeviceIndex)`

AIORESULT `ADC_CreateFastITConfig (unsigned long DeviceIndex, int size)`

Creates FastIT Config Blocks.

Parameters

<i>DeviceIndex</i>	
<i>size</i>	

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

<i>DeviceIndex</i>	
<i>pData</i>	buffer we will write data into

Returns

AIORESULT ADC_GetITScanV (unsigned long *DeviceIndex*, double * *pBuf*)

Parameters

<i>DeviceIndex</i>	
<i>pBuf</i>	

Returns

unsigned ADC_GetOversample (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

Returns

AIORESULT ADC_SetOversample (unsigned long *DeviceIndex*, unsigned char *Oversample*)

Parameters

<i>DeviceIndex</i>	
<i>Oversample</i>	

Returns

AIORESULT WriteConfigBlock (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

AIORESULT ReadConfigBlock (unsigned long *DeviceIndex*, AIOUSB_BOOL *forceRead*)

Parameters

<i>DeviceIndex</i>	
<i>forceRead</i>	

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

<i>config</i>	
<i>channel</i>	
<i>gainCode</i>	

AIORET_TYPE AIOUSB_IsDifferentialMode (**const ADConfigBlock** * *config*, unsigned *channel*)

Parameters

<i>config</i>	
<i>channel</i>	

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

<i>config</i>	
<i>channel</i>	
<i>differentialMode</i>	

AIORET_TYPE AIOUSB_GetCalMode (const ADConfigBlock * *config*)

AIORET_TYPE AIOUSB_SetCalMode (ADConfigBlock * *config*, unsigned *calMode*)

Parameters

<i>config</i>	
<i>calMode</i>	

AIORET_TYPE AIOUSB_SetOversample (ADConfigBlock * *config*, unsigned *overSample*)

AIORET_TYPE AIOUSB_GetOversample (ADConfigBlock * *config*)

unsigned AIOUSB_GetTriggerMode (const ADConfigBlock * *config*)

Parameters

<i>config</i>	
---------------	--

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

<i>config</i>	
<i>startChannel</i>	
<i>endChannel</i>	

unsigned long AIOUSB_SetStreamingBlockSize (unsigned long *DeviceIndex*, unsigned long *BlockSize*)

AIOUSB_BOOL AIOUSB_IsDiscardFirstSample (unsigned long *DeviceIndex*)

Parameters

<i>DeviceIndex</i>	
--------------------	--

Returns

unsigned long AIOUSB_SetDiscardFirstSample (unsigned long *DeviceIndex*, AIOUSB_BOOL *discard*)

Parameters

<i>DeviceIndex</i>	
<i>discard</i>	

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

<i>DeviceIndex</i>	
--------------------	--

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

<i>DeviceIndex</i>	
<i>autoCal</i>	
<i>returnCalTable</i>	
<i>saveFileName</i>	

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

<i>DeviceIndex</i>	
--------------------	--

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

<i>config</i>	
<i>DeviceIndex</i>	
<i>defaults</i>	

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex*, const char * *fileName*)

Parameters

<i>DeviceIndex</i>	
<i>fileName</i>	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex*, const unsigned short *calTable*[])

Parameters

<i>DeviceIndex</i>	
<i>calTable</i>	

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 [aiousblnit](#)

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

<i>DeviceIndex</i>	
<i>channel</i>	
<i>volts</i>	

Returns

unsigned long AIOUSB_ADC_LoadCalTable (unsigned long *DeviceIndex*, const char * *fileName*)

Parameters

<i>DeviceIndex</i>	
<i>fileName</i>	

Returns

unsigned long AIOUSB_ADC_SetCalTable (unsigned long *DeviceIndex*, const unsigned short *calTable*[])

Parameters

<i>DeviceIndex</i>	
<i>calTable</i>	

Returns

unsigned long AIOUSB_ClearFIFO (unsigned long *DeviceIndex*, FIFO_Method *Method*)

long AIOUSB_GetStreamingBlockSize (unsigned long *DeviceIndex*)

This function is deprecated.

Parameters

<i>DeviceIndex</i>	
--------------------	--

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

<i>config</i>	
<i>DeviceIndex</i>	
<i>defaults</i>	

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 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 *.*

- [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->divisor_a, and buf->divisor_b to create an output clock that matches the value stored in buf->hz *.

Parameters

	<i>hz</i>	
out	<i>diva</i>	Divisor A to be calculated
out	<i>divb</i>	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->divisor_a, and buf->divisor_b 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

	<i>hz</i>	
out	<i>diva</i>	Divisor A to be calculated
out	<i>divb</i>	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 = -AIUSB_ERROR_DEVICE_NOT_FOUND;
        goto g;
    } else if ( ( r = f ) != AIUSB_SUCCESS ) {
        goto g;
    } else if ( !(u = AIUSBDeviceGetUSBHandle( d )) ) {
        r = -AIUSB_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/AIUSB_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 blsRead)
- [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 *p-FrameData, 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

<i>DeviceIndex</i>	
<i>buf</i>	

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 *blsRead*)

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 *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) [ACCES_DEPRECATED](#)("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 blsRead)
- unsigned long [DIO_StreamClose](#) (unsigned long DeviceIndex)
- unsigned long [DIO_StreamSetClocks](#) (unsigned long DeviceIndex, double *ReadClockHz, double *WriteClockHz)
- unsigned long [DIO_StreamFrame](#) (unsigned long DeviceIndex, unsigned long FramePoints, unsigned short *pFrameData, 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

<i>DeviceIndex</i>	
<i>buf</i>	

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 *blsRead*)

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 *deviceList)
- **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

<i>minProductID</i>	
<i>maxProductID</i>	
<i>maxDevices</i>	
<i>deviceList</i>	[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

<i>DeviceIndex</i>	
<i>pSerialNumber</i>	

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

<i>where</i>	
<i>length</i>	
<i>is_ok_device</i>	

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 *device-List)
- **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

<i>minProductID</i>	
<i>maxProductID</i>	
<i>maxDevices</i>	
<i>deviceList</i>	[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

<i>DeviceIndex</i>	
<i>pSerialNumber</i>	

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

<i>where</i>	
<i>length</i>	
<i>is_ok_device</i>	

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

<i>obj</i>	
------------	--

AIORET_TYPE WDG_SetConfig (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Assigns the watchdog object to the device index in question.

Parameters

<i>DeviceIndex</i>	
<i>obj</i>	

Returns

AIORET_TYPE WDG_GetStatus (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Parameters

<i>DeviceIndex</i>	
<i>obj</i>	

Returns

AIORET_TYPE WDG_Pet (unsigned long *DeviceIndex*, AIOWDGConfig * *obj*)

Pets the watchdog and keeps it from resetting the device.

Parameters

<i>DeviceIndex</i>	
<i>obj</i>	

Returns

>= 0 if successful, < 0 otherwise

24.128 lib/AIOUSB_WDG.h File Reference

```
#include "aiosb.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_GetAsInteger (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 * [CStringArrayToStringWithDelimiter](#) (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* CStringArrayToStringWithDelimiter ( 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 * [CStringArrayToStringWithDelimiter](#) (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* CStringArrayToStringWithDelimiter (CStringArray * str, const char * delim)`

24.135 lib/DIOBuf.c File Reference

A smart buffer for handling Bit values and performing Bit arithmetic. 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 arithmetic. 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

<i>buf</i>	DIOBuf buffer one wishes to replace the content of
<i>ary</i>	Array of raw bytes values that will replace the original
<i>size_array</i>	The size, in bytes, of the <i>ary</i> 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

<i>buf</i>	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

<i>buf</i>	DIOBuf one wishes to convert to Hex
------------	-------------------------------------

Returns

A Hex string , prefixed with "0x", that represents the hexadecimal 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

<i>buf</i>	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

<i>buf</i>	
<i>index</i>	
<i>value</i>	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

<i>buf</i>	DIOBuf we wish to inspect
<i>index</i>	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

<i>size</i>	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

<i>ary</i>	
<i>size_array</i>	

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

<i>ary</i>	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

<i>buf</i>	DIOBuf buffer one wishes to replace the content of
<i>ary</i>	Array of raw bytes values that will replace the original
<i>size_array</i>	The size, in bytes, of the <i>ary</i> 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

<i>buf</i>	DIOBuf one wishes to convert to Hex
------------	-------------------------------------

Returns

A Hex string , prefixed with "0x", that represents the hexadecimal 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

<i>buf</i>	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

<i>buf</i>	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

<i>buf</i>	
<i>index</i>	
<i>value</i>	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

<i>buf</i>	DIOBuf we wish to inspect
<i>index</i>	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 wIndex, 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 USBDeviceFetchADCCConfigBlock (USBDevice * usb, ADCCConfigBlock * configBlock)

int USBDevicePutADCCConfigBlock (USBDevice * usb, ADCCConfigBlock * 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 [aioub_libusb_args](#)

Macros

- #define [INTERNAL_METHOD](#)(NAME, RETVAL,...) RETVAL (*NAME)(__VA_ARGS__)

Typedefs

- typedef struct [USBDevice](#) [USBDevice](#)
- typedef struct [aioub_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 InitializeUSBDevice, 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 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)

24.143.1 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 "aiusb.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_adccconfig_settings](#) ([ADCCConfigBlock](#) *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_ao_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_adccfg_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_ao_cmd_line](#) (struct [opts](#) *options, int argc, char *argv[])
Simple command line parser sets up testing features.
- void [print_ao_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_adccfg_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_ao_cmd_line ( struct opts * options, int argc, char * argv[] )
```

Simple command line parser sets up testing features.

```
void print_ao_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_adccfg_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 <AIOTypes.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 "AIOTypes.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 != [AIOTypes.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 <aiusb.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 "aiusb.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 <aiusb.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 "aiusb.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 "aiusb.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 <aiusb.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 [goDolt](#) (TestCaseSetup &t)
- int [main](#) (int argc, char **argv)

24.161.1 Function Documentation

```
void goDolt ( 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 [goDolt](#) ([TestCaseSetup](#) &t)
- int [main](#) (int argc, char **argv)

24.162.1 Function Documentation

void goDolt ([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 <aiusb.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 <aiusb.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@accessio.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@accessio.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 NewAIOContinuousBufFromJSON('{"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 NewAIOContinuousBufFromJSON('{"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 aioub header files can be located:

```
export CPATH=/usr/local/include/libusb-1.0/:/usr/local/include/aioub/
```

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 -laioub -lusb-1.0 -o sample
```

or, to enable debug features

```
g++ -ggdb sample.cpp -laioubdbg -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 <aioub.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://acesio.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 <aiusb.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 <aiusb.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

- `AIUSB_BOOL find_dio_96 (AIUSBDevice *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 "aioub.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 "aiusb.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 "aiusb.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/AI-
OUSB/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 Refer-
ence

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

- ~AI16_DataSet
 - AIOTypes.h, [299](#)
- ~AI16_InputRange
 - AIOTypes.h, [299](#)
- ~AI16_OutputRange
 - AIOTypes.h, [299](#)
- ~AI16_AnalogOutputSubsystem
 - AIOTypes.h, [299](#)
- ~AI16_AnalogOutputSubsystem
 - AIOTypes.h, [299](#)
- ~AI16_OutputRange
 - AIOTypes.h, [299](#)
- ~AnalogIORange
 - AIOTypes.h, [299](#)
- ~AnalogInputSubsystem
 - AIOTypes.h, [299](#)
- ~AnalogOutputSubsystem
 - AIOTypes.h, [299](#)
- ~CounterSubsystem
 - AIOTypes.h, [299](#)
- ~DA12_AnalogOutputSubsystem
 - AIOTypes.h, [299](#)
- ~DA12_AnalogOutputSubsystem
 - AIOTypes.h, [299](#)
- ~DA12_OutputRange
 - AIOTypes.h, [299](#)
- ~DIOStreamSubsystem
 - AIOTypes.h, [299](#)
- ~DeviceSubsystem
 - AIOTypes.h, [299](#)
- ~DigitalIOSubsystem
 - AIOTypes.h, [299](#)
- ~TestCaseSetup
 - AIOTypes.h, [299](#)
- ~USBDeviceBase
 - AIOTypes.h, [299](#)
- ~USBDeviceManager
 - AIOTypes.h, [299](#)
- ~USB_AI16_Family
 - AIOTypes.h, [299](#)
- ~USB_AIO16_Family
 - AIOTypes.h, [299](#)
- ~USB_AO16_Family
 - AIOTypes.h, [299](#)
- ~USB_DIO_Family
 - AIOTypes.h, [299](#)
- ACCES_VENDOR_ID
 - AIOTypes.h, [299](#)
- AD_CAL_MODE_BIP_GROUND
 - AIOTypes.h, [299](#)
- AD_CAL_MODE_GROUND
 - AIOTypes.h, [299](#)
- AD_CAL_MODE_HIGH_REF
 - AIOTypes.h, [299](#)
- AD_CAL_MODE_NORMAL
 - AIOTypes.h, [299](#)
- AD_CAL_MODE_REFERENCE
 - AIOTypes.h, [299](#)
- AD_CONFIG_CAL_MODE
 - AIOTypes.h, [299](#)
- AD_CONFIG_GAIN_CODE
 - AIOTypes.h, [299](#)
- AD_CONFIG_MUX_START_END
 - AIOTypes.h, [299](#)
- AD_CONFIG_OVERSAMPLE
 - AIOTypes.h, [299](#)
- AD_CONFIG_REGISTERS
 - AIOTypes.h, [299](#)
- AD_CONFIG_START_STOP_CHANNEL_EX
 - AIOTypes.h, [299](#)
- AD_CONFIG_TRIG_COUNT
 - AIOTypes.h, [299](#)
- AD_DIFFERENTIAL_MODE
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_0_10V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_0_1V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_0_2V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_0_5V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_10V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_1V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_2V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_5V
 - AIOTypes.h, [299](#)
- AD_GAIN_CODE_MASK
 - AIOTypes.h, [299](#)
- AD_MAX_CHANNELS
 - AIOTypes.h, [299](#)
- AD_MAX_CONFIG_REGISTERS
 - AIOTypes.h, [299](#)
- AD_MAX_TIMEOUT
 - AIOTypes.h, [299](#)
- AD_MIN_CONFIG_REGISTERS
 - AIOTypes.h, [299](#)
- AD_MIN_TIMEOUT
 - AIOTypes.h, [299](#)
- AD_MUX_CONFIG_REGISTERS
 - AIOTypes.h, [299](#)
- AD_NO_SET_CAL
 - AIOTypes.h, [299](#)
- AD_NUM_GAIN_CODE_REGISTERS
 - AIOTypes.h, [299](#)
- AD_NUM_GAIN_CODES
 - AIOTypes.h, [299](#)
- AD_REGISTER_CAL_MODE
 - AIOTypes.h, [299](#)
- AD_REGISTER_GAIN_CODE
 - AIOTypes.h, [299](#)
- AD_REGISTER_MUX_START_END
 - AIOTypes.h, [299](#)
- AD_REGISTER_OVERSAMPLE
 - AIOTypes.h, [299](#)
- AD_REGISTER_START_END
 - AIOTypes.h, [299](#)
- AD_REGISTER_TRIG_COUNT
 - AIOTypes.h, [299](#)
- AD_SCAN_BULKACQUIRE
 - AIOTypes.h, [299](#)
- AD_SCAN_CONTINUOUS
 - AIOTypes.h, [299](#)
- AD_SCAN_GETCHANNEL
 - AIOTypes.h, [299](#)
- AD_SCAN_GETCHANNELV
 - AIOTypes.h, [299](#)

AD_SCAN_GETSCAN
 AIOConfiguration.h, 234
 AD_SCAN_GETSCANV
 AIOConfiguration.h, 234
 AD_SET_CAL_AUTO
 AIOConfiguration.h, 234
 AD_SET_CAL_MANUAL
 AIOConfiguration.h, 234
 AD_SET_CAL_NORMAL
 AIOConfiguration.h, 234
 AD_TRIGGER_CTR0_EXT
 AIOTypes.h, 299
 AD_TRIGGER_EXTERNAL
 AIOTypes.h, 299
 AD_TRIGGER_FALLING_EDGE
 AIOTypes.h, 299
 AD_TRIGGER_SCAN
 AIOTypes.h, 299
 AD_TRIGGER_TIMER
 AIOTypes.h, 299
 AD_TRIGGER_VALID_MASK
 AIOTypes.h, 299
 ADCCONFIG_OPT
 AIOCommandLine.h, 229
 ADCCONIGBLOCK_CONFIG
 AIOConfiguration.h, 234
 ADCalMode_begin
 AIOTypes.h, 301
 ADCalMode_end
 AIOTypes.h, 301
 ADGainCode_begin
 AIOTypes.h, 299
 ADGainCode_end
 AIOTypes.h, 299
 ADRegister_begin
 AIOTypes.h, 299
 ADRegister_end
 AIOTypes.h, 299
 AI_16_MAX_COUNTS
 AIOTypes.h, 301
 AIO_CONT_BUF_TYPE_COUNTS
 AIOContinuousBuffer.h, 249
 AIO_CONT_BUF_TYPE_VOLTS
 AIOContinuousBuffer.h, 249
 AIO_CONTINUE_RUNNING
 AIOTypes.h, 301
 AIO_COUNTS_BUF
 AIOBuf.h, 216
 AIO_DEFAULT_BUF
 AIOBuf.h, 216
 AIO_ERROR_BUF
 AIOBuf.h, 216
 AIO_PER_CHANNEL
 AIOTypes.h, 293
 AIO_PER_OVERSAMPLE
 AIOTypes.h, 293
 AIO_PER_SCANS
 AIOTypes.h, 293
 AIO_TERMINATE_CALLBACK
 AIOTypes.h, 301
 AIO_VOLTS_BUF
 AIOBuf.h, 216
 AIOBuf.h
 AIO_COUNTS_BUF, 216
 AIO_DEFAULT_BUF, 216
 AIO_ERROR_BUF, 216
 AIO_VOLTS_BUF, 216
 AIOCONTINUOUS_BUF_ALLORNONE
 AIOTypes.h, 294
 AIOCONTINUOUS_BUF_NORMAL
 AIOTypes.h, 294
 AIOCONTINUOUS_BUF_OVERRIDE
 AIOTypes.h, 294
 AIOCONTINUOUSBUF_CONFIG
 AIOConfiguration.h, 234
 AIOChannelRange.c
 BEGIN, 222
 END, 222
 END_CHANNEL, 222
 GAIN, 222
 START_CHANNEL, 222
 AIOCommandCode_begin
 AIOTypes.h, 301
 AIOCommandCode_end
 AIOTypes.h, 301
 AIOCommandLine.h
 ADCCONFIG_OPT, 229
 CHANNEL_OPT, 229
 COUNT_OPT, 229
 DEBUG_OPT, 229
 FILE_OPT, 229
 INDEX_NUM, 229
 SAMPLE_OPT, 229
 SETCAL_OPT, 229
 TIMEOUT_OPT, 229
 AIOConfiguration.h
 AD_NO_SET_CAL, 234
 AD_SCAN_BULKACQUIRE, 234
 AD_SCAN_CONTINUOUS, 234
 AD_SCAN_GETCHANNEL, 234
 AD_SCAN_GETCHANNELV, 234
 AD_SCAN_GETSCAN, 234
 AD_SCAN_GETSCANV, 234
 AD_SET_CAL_AUTO, 234
 AD_SET_CAL_MANUAL, 234
 AD_SET_CAL_NORMAL, 234
 ADCCONIGBLOCK_CONFIG, 234
 AIOCONTINUOUSBUF_CONFIG, 234
 NO_CONFIG, 234
 AIOContinuousBufMode_begin
 AIOTypes.h, 294
 AIOContinuousBufMode_end
 AIOTypes.h, 294
 AIOContinuousBuffer.h
 AIO_CONT_BUF_TYPE_COUNTS, 249
 AIO_CONT_BUF_TYPE_VOLTS, 249
 AIODEBUG_LEVEL
 AIOUSB_Log.h, 349
 AIODEFAULT_LOG_LEVEL
 AIOUSB_Log.h, 349
 AIODEVEL_LEVEL
 AIOUSB_Log.h, 349
 AIOERROR_LEVEL
 AIOUSB_Log.h, 349
 AIOEither.h
 aioeither_value_double, 275
 aioeither_value_double_t, 275
 aioeither_value_int, 274
 aioeither_value_int32_t, 274
 aioeither_value_longdouble_t, 275
 aioeither_value_obj, 275
 aioeither_value_string, 275
 aioeither_value_uint16_t, 275
 aioeither_value_uint32_t, 274
 aioeither_value_uint8_t, 275
 aioeither_value_unsigned, 275
 aioeither_vlaue_int16_t, 275
 AIOFATAL_LEVEL
 AIOUSB_Log.h, 349
 AIOINFO_LEVEL
 AIOUSB_Log.h, 349
 AIOTypes.h
 ACCES_VENDOR_ID, 294
 AD_CAL_MODE_BIP_GROUND, 301

- AD_CAL_MODE_GROUND, 301
- AD_CAL_MODE_HIGH_REF, 301
- AD_CAL_MODE_NORMAL, 301
- AD_CAL_MODE_REFERENCE, 301
- AD_CONFIG_CAL_MODE, 299
- AD_CONFIG_GAIN_CODE, 299
- AD_CONFIG_MUX_START_END, 299
- AD_CONFIG_OVERSAMPLE, 299
- AD_CONFIG_REGISTERS, 301
- AD_CONFIG_START_END, 299
- AD_CONFIG_START_STOP_CHANNEL_EX, 299
- AD_CONFIG_TRIG_COUNT, 299
- AD_DIFFERENTIAL_MODE, 299
- AD_GAIN_CODE_0_10V, 299
- AD_GAIN_CODE_0_1V, 299
- AD_GAIN_CODE_0_2V, 299
- AD_GAIN_CODE_0_5V, 299
- AD_GAIN_CODE_10V, 299
- AD_GAIN_CODE_1V, 299
- AD_GAIN_CODE_2V, 299
- AD_GAIN_CODE_5V, 299
- AD_GAIN_CODE_MASK, 298
- AD_MAX_CHANNELS, 298
- AD_MAX_CONFIG_REGISTERS, 299
- AD_MAX_TIMEOUT, 299
- AD_MIN_CONFIG_REGISTERS, 299
- AD_MIN_TIMEOUT, 299
- AD_MUX_CONFIG_REGISTERS, 301
- AD_NUM_GAIN_CODE_REGISTERS, 299
- AD_NUM_GAIN_CODES, 299
- AD_REGISTER_CAL_MODE, 299
- AD_REGISTER_GAIN_CODE, 299
- AD_REGISTER_MUX_START_END, 299
- AD_REGISTER_OVERSAMPLE, 299
- AD_REGISTER_START_END, 299
- AD_REGISTER_TRIG_COUNT, 299
- AD_TRIGGER_CTR0_EXT, 299
- AD_TRIGGER_EXTERNAL, 299
- AD_TRIGGER_FALLING_EDGE, 299
- AD_TRIGGER_SCAN, 299
- AD_TRIGGER_TIMER, 299
- AD_TRIGGER_VALID_MASK, 299
- ADCalMode_begin, 301
- ADCalMode_end, 301
- ADGainCode_begin, 299
- ADGainCode_end, 299
- ADRegister_begin, 299
- ADRegister_end, 299
- AI_16_MAX_COUNTS, 301
- AIO_CONTINUE_RUNNING, 301
- AIO_PER_CHANNEL, 293
- AIO_PER_OVERSAMPLE, 293
- AIO_PER_SCANS, 293
- AIO_TERMINATE_CALLBACK, 301
- AIOCONTINUOUS_BUF_ALLORNONE, 294
- AIOCONTINUOUS_BUF_NORMAL, 294
- AIOCONTINUOUS_BUF_OVERRIDE, 294
- AIOCommandCode_begin, 301
- AIOCommandCode_end, 301
- AIOContinuousBufMode_begin, 294
- AIOContinuousBufMode_end, 294
- AIOUSB_ERROR_AIOCOMMANDLINE_HELP, 298
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_-
CHANNEL_RANGE, 298
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_I-
NDEX_NUM, 298
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_-
NUM_CHANNELS, 298
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_-
START_END_CHANNEL, 298
- AIOUSB_ERROR_BAD_TOKEN_TYPE, 297
- AIOUSB_ERROR_DEVICE_NOT_CONNECTED, 297
- AIOUSB_ERROR_DEVICE_NOT_FOUND, 297
- AIOUSB_ERROR_DIVIDE_BY_ZERO, 297
- AIOUSB_ERROR_DUP_NAME, 297
- AIOUSB_ERROR_FILE_NOT_FOUND, 297
- AIOUSB_ERROR_HANDLE_EOF, 297
- AIOUSB_ERROR_INVALID_ADCCONFIG, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_CAL_S-
SETTING, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_CHANN-
EL_SETTING, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_DEVIC-
E, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_MUX_S-
SETTING, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_OVERS-
AMPLE_SETTING, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_REGIS-
TER_SETTING, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_SETTIN-
G, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_SIZE, 298
- AIOUSB_ERROR_INVALID_ADCCONFIG_TRIGG-
ER_SETTING, 298
- AIOUSB_ERROR_INVALID_AIOARGUMENT, 298
- AIOUSB_ERROR_INVALID_AIOBUFTYPE, 298
- AIOUSB_ERROR_INVALID_AIOCHANNELMASK, 298
- AIOUSB_ERROR_INVALID_AIOCMD, 298
- AIOUSB_ERROR_INVALID_AIOCONFIGURATION, 298
- AIOUSB_ERROR_INVALID_AIOCONTINUOUS_B-
UFFER, 298
- AIOUSB_ERROR_INVALID_AIOCONTINUOUS_B-
UFFER_NUM_CHANNELS, 298
- AIOUSB_ERROR_INVALID_AIODEVICE_QUERY, 298
- AIOUSB_ERROR_INVALID_AIOEITHER, 298
- AIOUSB_ERROR_INVALID_AIOEITHER_ALLOCA-
TION, 298
- AIOUSB_ERROR_INVALID_AIOFIFO, 298
- AIOUSB_ERROR_INVALID_CALLBACK, 298
- AIOUSB_ERROR_INVALID_CALMODE, 298
- AIOUSB_ERROR_INVALID_CHANNEL_NUMBER, 298
- AIOUSB_ERROR_INVALID_CHANNELS_PER_GR-
OUP_SETTING, 298
- AIOUSB_ERROR_INVALID_CONFIG, 298
- AIOUSB_ERROR_INVALID_COUNTS, 298
- AIOUSB_ERROR_INVALID_COUNTS_CONVERT-
ER, 298
- AIOUSB_ERROR_INVALID_DATA, 297
- AIOUSB_ERROR_INVALID_DEVICE, 298
- AIOUSB_ERROR_INVALID_DEVICE_CHANNEL_-
SETTING, 298
- AIOUSB_ERROR_INVALID_DEVICE_FUNCTIONA-
L_PARAMETER, 298
- AIOUSB_ERROR_INVALID_DEVICE_MUX_CHAN-
NEL_SETTING, 298
- AIOUSB_ERROR_INVALID_DEVICE_SETTING, 298
- AIOUSB_ERROR_INVALID_DEVICE_STREAM_S-
SETTING, 298
- AIOUSB_ERROR_INVALID_DIOBUF, 298
- AIOUSB_ERROR_INVALID_GAINCODE, 298
- AIOUSB_ERROR_INVALID_INDEX, 297
- AIOUSB_ERROR_INVALID_LIBUSB_DEVICE_HA-
NDLE, 298
- AIOUSB_ERROR_INVALID_MEMORY, 297
- AIOUSB_ERROR_INVALID_MUTEX, 297

AIOUSB_ERROR_INVALID_PARAMETER, [297](#)
 AIOUSB_ERROR_INVALID_THREAD, [297](#)
 AIOUSB_ERROR_INVALID_TIMEOUT, [298](#)
 AIOUSB_ERROR_INVALID_USBDEVICE, [298](#)
 AIOUSB_ERROR_INVALID_VOLTAGES, [298](#)
 AIOUSB_ERROR_LIBUSB, [298](#)
 AIOUSB_ERROR_NOT_ENOUGH_MEMORY, [297](#)
 AIOUSB_ERROR_NOT_INIT, [297](#)
 AIOUSB_ERROR_NOT_SUPPORTED, [297](#)
 AIOUSB_ERROR_OPEN_FAILED, [297](#)
 AIOUSB_ERROR_TIMEOUT, [297](#)
 AIOUSB_ERROR_USB_INIT, [298](#)
 AIOUSB_ERROR_USBDEVICE_NOT_FOUND, [297](#)
 AIOUSB_FALSE, [294](#)
 AIOUSB_FIFO_COPY_ERROR, [298](#)
 AIOUSB_MAX_NAME_SIZE, [301](#)
 AIOUSB_SUCCESS, [297](#)
 AIOUSB_TRUE, [294](#)
 AUR_ADC_GET_CONFIG, [300](#)
 AUR_ADC_IMMEDIATE, [300](#)
 AUR_ADC_SET_CONFIG, [300](#)
 AUR_CTR_COS_BULK_ABORT, [300](#)
 AUR_CTR_COS_BULK_GATE2, [300](#)
 AUR_CTR_LOAD, [300](#)
 AUR_CTR_MODE, [300](#)
 AUR_CTR_MODELOAD, [300](#)
 AUR_CTR_PUR_EVCT, [300](#)
 AUR_CTR_PUR_FIRST, [300](#)
 AUR_CTR_PUR_MFRQ, [300](#)
 AUR_CTR_PUR_MPUL, [300](#)
 AUR_CTR_PUR_OFRQ, [300](#)
 AUR_CTR_READ, [300](#)
 AUR_CTR_READALL, [300](#)
 AUR_CTR_READLATCHED, [300](#)
 AUR_CTR_SELGATE, [300](#)
 AUR_DAC_CONTROL, [300](#)
 AUR_DAC_DATAPTR, [300](#)
 AUR_DAC_DIVISOR, [300](#)
 AUR_DAC_IMMEDIATE, [300](#)
 AUR_DAC_RANGE, [300](#)
 AUR_DIO_CONFIG, [300](#)
 AUR_DIO_CONFIG_QUERY, [300](#)
 AUR_DIO_READ, [300](#)
 AUR_DIO_SETCLOCKS, [300](#)
 AUR_DIO_SPI_READ, [300](#)
 AUR_DIO_SPI_WRITE, [300](#)
 AUR_DIO_STREAM_OPEN_INPUT, [300](#)
 AUR_DIO_STREAM_OPEN_OUTPUT, [300](#)
 AUR_DIO_WDG16_DEPREC, [300](#)
 AUR_DIO_WRITE, [300](#)
 AUR_EEPROM_READ, [300](#)
 AUR_EEPROM_WRITE, [300](#)
 AUR_FLASH_READWRITE, [300](#)
 AUR_GEN_ABORT_AND_CLEAR, [300](#)
 AUR_GEN_CLEAR_FIFO, [300](#)
 AUR_GEN_CLEAR_FIFO_NEXT, [300](#)
 AUR_GEN_CLEAR_FIFO_WAIT, [300](#)
 AUR_GEN_STREAM_STATUS, [300](#)
 AUR_LOAD_BULK_CALIBRATION_BLOCK, [300](#)
 AUR_OFFLINE_READWRITE, [300](#)
 AUR_PROBE_CALFEATURE, [300](#)
 AUR_READBACK_GLOBAL_STATE, [300](#)
 AUR_SAVE_GLOBAL_STATE, [300](#)
 AUR_SELF_TEST_1, [300](#)
 AUR_START_ACQUIRING_BLOCK, [300](#)
 AUR_WDG, [300](#)
 AUR_WDG_STATUS, [300](#)
 BITS_PER_BYTE, [301](#)
 CAL_TABLE_WORDS, [301](#)
 CLEAR_FIFO_METHOD_AUTO, [297](#)
 CLEAR_FIFO_METHOD_IMMEDIATE, [297](#)
 CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT, [297](#)
 CLEAR_FIFO_METHOD_NOW, [297](#)
 CLEAR_FIFO_METHOD_WAIT, [297](#)
 COUNTER_NUM_MODES, [301](#)
 COUNTERS_PER_BLOCK, [301](#)
 CYPRESS_DESC_PARAMS, [301](#)
 CYPRESS_GET_DESC, [300](#)
 CYPRESS_MAX_DESC_SIZE, [301](#)
 DAC_RANGE_0_10V, [296](#)
 DAC_RANGE_0_5V, [296](#)
 DAC_RANGE_10V, [296](#)
 DAC_RANGE_5V, [296](#)
 DAC_RESET, [301](#)
 DACRange_begin, [296](#)
 DACRange_end, [296](#)
 diFirst, [296](#)
 diNone, [296](#)
 diOnly, [296](#)
 EEPROM_CUSTOM_BASE_ADDRESS, [301](#)
 EEPROM_CUSTOM_MAX_ADDRESS, [301](#)
 EEPROM_CUSTOM_MIN_ADDRESS, [301](#)
 EEPROM_SERIAL_NUMBER_ADDRESS, [301](#)
 FIFO_Method_begin, [297](#)
 FIFO_Method_end, [297](#)
 GENERIC_DOSOMETHING_PLACEHOLDER, [301](#)
 INVALID_OBJECT, [293](#)
 JOINED, [293](#)
 MAX_IMM_ADCS, [301](#)
 MAX_USB_DEVICES, [294](#)
 NOT_STARTED, [293](#)
 PICO_DIO16RO8, [295](#)
 ProductIDS_begin, [294](#)
 ProductIDS_end, [296](#)
 RUNNING, [293](#)
 RUNNING_OR_WITH_DATA, [293](#)
 ResultCode_begin, [297](#)
 ResultCode_end, [298](#)
 TERMINATED, [293](#)
 TERMINATED_OVERRUN, [293](#)
 TERMINATING, [293](#)
 THREAD_STATUS_begin, [293](#)
 THREAD_STATUS_end, [293](#)
 USB_AI12_128, [295](#)
 USB_AI12_128A, [295](#)
 USB_AI12_128E, [295](#)
 USB_AI12_16, [295](#)
 USB_AI12_16A, [295](#)
 USB_AI12_16E, [295](#)
 USB_AI12_32, [295](#)
 USB_AI12_32A, [295](#)
 USB_AI12_32E, [295](#)
 USB_AI12_64, [295](#)
 USB_AI12_64A, [295](#)
 USB_AI12_64E, [295](#)
 USB_AI12_64M, [295](#)
 USB_AI12_64MA, [295](#)
 USB_AI12_64ME, [295](#)
 USB_AI12_96, [295](#)
 USB_AI12_96A, [295](#)
 USB_AI12_96E, [295](#)
 USB_AI16_128A, [295](#)
 USB_AI16_128E, [295](#)
 USB_AI16_16A, [295](#)
 USB_AI16_16E, [295](#)
 USB_AI16_32A, [295](#)
 USB_AI16_32E, [295](#)
 USB_AI16_64A, [295](#)
 USB_AI16_64E, [295](#)
 USB_AI16_64MA, [295](#)
 USB_AI16_64ME, [295](#)
 USB_AI16_96A, [295](#)

- USB_AI16_96E, [295](#)
- USB_AIO12_128, [296](#)
- USB_AIO12_128A, [296](#)
- USB_AIO12_128E, [296](#)
- USB_AIO12_16, [296](#)
- USB_AIO12_16A, [296](#)
- USB_AIO12_16E, [296](#)
- USB_AIO12_32, [296](#)
- USB_AIO12_32A, [296](#)
- USB_AIO12_32E, [296](#)
- USB_AIO12_64, [296](#)
- USB_AIO12_64A, [296](#)
- USB_AIO12_64E, [296](#)
- USB_AIO12_64M, [296](#)
- USB_AIO12_64MA, [296](#)
- USB_AIO12_64ME, [296](#)
- USB_AIO12_96, [296](#)
- USB_AIO12_96A, [296](#)
- USB_AIO12_96E, [296](#)
- USB_AIO16_128A, [296](#)
- USB_AIO16_128E, [296](#)
- USB_AIO16_16A, [296](#)
- USB_AIO16_16E, [296](#)
- USB_AIO16_32A, [296](#)
- USB_AIO16_32E, [296](#)
- USB_AIO16_64A, [296](#)
- USB_AIO16_64E, [296](#)
- USB_AIO16_64MA, [296](#)
- USB_AIO16_64ME, [296](#)
- USB_AIO16_96A, [296](#)
- USB_AIO16_96E, [296](#)
- USB_AO12_12, [295](#)
- USB_AO12_12A, [295](#)
- USB_AO12_16, [295](#)
- USB_AO12_16A, [295](#)
- USB_AO12_4, [295](#)
- USB_AO12_4A, [295](#)
- USB_AO12_8, [295](#)
- USB_AO12_8A, [295](#)
- USB_AO16_12, [295](#)
- USB_AO16_12A, [295](#)
- USB_AO16_16, [295](#)
- USB_AO16_16A, [295](#)
- USB_AO16_4, [295](#)
- USB_AO16_4A, [295](#)
- USB_AO16_8, [295](#)
- USB_AO16_8A, [295](#)
- USB_AO_ARB1, [295](#)
- USB_BULK_READ_ENDPOINT, [301](#)
- USB_BULK_WRITE_ENDPOINT, [301](#)
- USB_CTR_15, [294](#)
- USB_DA12_8A, [294](#)
- USB_DA12_8A_REV_A, [294](#)
- USB_DA12_8E, [294](#)
- USB_DI16A, [294](#)
- USB_DI16A_REV_A1, [294](#)
- USB_DI16A_REV_A2, [294](#)
- USB_DIO16RO8, [295](#)
- USB_DIO24_CTR6, [294](#)
- USB_DIO24DO12, [295](#)
- USB_DIO48DO24, [295](#)
- USB_DIO_16A, [294](#)
- USB_DIO_16H, [294](#)
- USB_DIO_32, [294](#)
- USB_DIO_32I, [294](#)
- USB_DIO_48, [294](#)
- USB_DIO_96, [294](#)
- USB_DO16A, [294](#)
- USB_DO16A_REV_A1, [294](#)
- USB_DO24, [295](#)
- USB_IDIO_16, [294](#)
- USB_IDIO_4, [294](#)
- USB_IDIO_8, [294](#)
- USB_IDO_16, [294](#)
- USB_II_16, [294](#)
- USB_II_16_OLD, [294](#)
- USB_II_8, [294](#)
- USB_II_8_OLD, [294](#)
- USB_IIRO4_2SM, [295](#)
- USB_IIRO4_COM, [295](#)
- USB_IIRO_16, [294](#)
- USB_IIRO_4, [294](#)
- USB_IIRO_8, [294](#)
- USB_READ_FROM_DEVICE, [301](#)
- USB_RO_16, [294](#)
- USB_WRITE_TO_DEVICE, [301](#)
- USBP_II8IDO4A, [295](#)
- VENDOR_REQUEST_begin, [300](#)
- VENDOR_REQUEST_end, [300](#)
- WITH_DATA, [293](#)
- AIOUSB_ERROR_AIOCOMMANDLINE_HELP
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_CHANNEL_RANGE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_INDEX_NUM
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_NUM_CHANNELS
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_AIOCOMMANDLINE_INVALID_START_END_CHANNEL
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_BAD_TOKEN_TYPE
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_DEVICE_NOT_CONNECTED
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_DEVICE_NOT_FOUND
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_DIVIDE_BY_ZERO
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_DUP_NAME
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_FILE_NOT_FOUND
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_HANDLE_EOF
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_CAL_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_CHANNEL_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_DEVICE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_MUX_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_OVERSAMPLE_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_REGISTER_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_SIZE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_ADCCONFIG_TRIGGER_SETTING
 - AIOTypes.h, [298](#)

- AIOUSB_ERROR_INVALID_AIOARGUMENT
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOBUFTYPE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOCHANNELMASK
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOCMD
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOCONFIGURATION
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOCONTINUOUS_BUFFER_NUM_CHANNELS
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIODEVICE_QUERY
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOEITHER
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOEITHER_ALLOCATION
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_AIOFIFO
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_CALLBACK
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_CALMODE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_CHANNEL_NUMBER
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_CHANNELS_PER_GROUP_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_CONFIG
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_COUNTS
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_COUNTS_CONVERTER
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_DATA
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_INVALID_DEVICE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_DEVICE_CHANNEL_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_DEVICE_FUNCTIONAL_PARAMETER
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_DEVICE_MUX_CHANNEL_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_DEVICE_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_DEVICE_STREAM_SETTING
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_DIOBUF
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_GAINCODE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_INDEX
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_INVALID_LIBUSB_DEVICE_HANDLE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_MEMORY
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_INVALID_MUTEX
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_INVALID_PARAMETER
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_INVALID_THREAD
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_INVALID_TIMEOUT
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_USBDEVICE
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_INVALID_VOLTAGES
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_LIBUSB
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_NOT_ENOUGH_MEMORY
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_NOT_INIT
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_NOT_SUPPORTED
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_OPEN_FAILED
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_TIMEOUT
 - AIOTypes.h, [297](#)
- AIOUSB_ERROR_USB_INIT
 - AIOTypes.h, [298](#)
- AIOUSB_ERROR_USBDEVICE_NOT_FOUND
 - AIOTypes.h, [297](#)
- AIOUSB_FALSE
 - AIOTypes.h, [294](#)
- AIOUSB_FIFO_COPY_ERROR
 - AIOTypes.h, [298](#)
- AIOUSB_Log.h
 - AIODEBUG_LEVEL, [349](#)
 - AIODEFAULT_LOG_LEVEL, [349](#)
 - AIODEVEL_LEVEL, [349](#)
 - AIOERROR_LEVEL, [349](#)
 - AIOFATAL_LEVEL, [349](#)
 - AIOINFO_LEVEL, [349](#)
 - AIOWARN_LEVEL, [349](#)
- AIOUSB_MAX_NAME_SIZE
 - AIOTypes.h, [301](#)
- AIOUSB_Properties.h
 - BASIC, [352](#)
 - JSON, [352](#)
 - TERSE, [352](#)
 - YAML, [352](#)
- AIOUSB_SUCCESS
 - AIOTypes.h, [297](#)
- AIOUSB_TRUE
 - AIOTypes.h, [294](#)
- AIOUSB_WDG.h
 - AIOUSB_WDG_READ_INDEX, [356](#)
 - AIOUSB_WDG_READ_VALUE, [356](#)
 - WDGVals_begin, [356](#)
 - WDGVals_end, [356](#)
- AIOUSB_WDG_READ_INDEX
 - AIOUSB_WDG.h, [356](#)
- AIOUSB_WDG_READ_VALUE
 - AIOUSB_WDG.h, [356](#)
- AIOWARN_LEVEL
 - AIOUSB_Log.h, [349](#)
- AUR_ADC_GET_CONFIG
 - AIOTypes.h, [300](#)
- AUR_ADC_IMMEDIATE
 - AIOTypes.h, [300](#)
- AUR_ADC_SET_CONFIG
 - AIOTypes.h, [300](#)
- AUR_CTR_COS_BULK_ABORT
 - AIOTypes.h, [300](#)
- AUR_CTR_COS_BULK_GATE2
 - AIOTypes.h, [300](#)
- AUR_CTR_LOAD
 - AIOTypes.h, [300](#)
- AUR_CTR_MODE
 - AIOTypes.h, [300](#)
- AUR_CTR_MODELOAD

AIOTypes.h, [300](#)
 AUR_CTR_PUR_EVT
 AIOTypes.h, [300](#)
 AUR_CTR_PUR_FIRST
 AIOTypes.h, [300](#)
 AUR_CTR_PUR_MFRQ
 AIOTypes.h, [300](#)
 AUR_CTR_PUR_MPUL
 AIOTypes.h, [300](#)
 AUR_CTR_PUR_OFQR
 AIOTypes.h, [300](#)
 AUR_CTR_READ
 AIOTypes.h, [300](#)
 AUR_CTR_READALL
 AIOTypes.h, [300](#)
 AUR_CTR_READLATCHED
 AIOTypes.h, [300](#)
 AUR_CTR_SELGATE
 AIOTypes.h, [300](#)
 AUR_DAC_CONTROL
 AIOTypes.h, [300](#)
 AUR_DAC_DATAPTR
 AIOTypes.h, [300](#)
 AUR_DAC_DIVISOR
 AIOTypes.h, [300](#)
 AUR_DAC_IMMEDIATE
 AIOTypes.h, [300](#)
 AUR_DAC_RANGE
 AIOTypes.h, [300](#)
 AUR_DIO_CONFIG
 AIOTypes.h, [300](#)
 AUR_DIO_CONFIG_QUERY
 AIOTypes.h, [300](#)
 AUR_DIO_READ
 AIOTypes.h, [300](#)
 AUR_DIO_SETCLOCKS
 AIOTypes.h, [300](#)
 AUR_DIO_SPI_READ
 AIOTypes.h, [300](#)
 AUR_DIO_SPI_WRITE
 AIOTypes.h, [300](#)
 AUR_DIO_STREAM_OPEN_INPUT
 AIOTypes.h, [300](#)
 AUR_DIO_STREAM_OPEN_OUTPUT
 AIOTypes.h, [300](#)
 AUR_DIO_WDG16_DEPREC
 AIOTypes.h, [300](#)
 AUR_DIO_WRITE
 AIOTypes.h, [300](#)
 AUR_EEPROM_READ
 AIOTypes.h, [300](#)
 AUR_EEPROM_WRITE
 AIOTypes.h, [300](#)
 AUR_FLASH_READWRITE
 AIOTypes.h, [300](#)
 AUR_GEN_ABORT_AND_CLEAR
 AIOTypes.h, [300](#)
 AUR_GEN_CLEAR_FIFO
 AIOTypes.h, [300](#)
 AUR_GEN_CLEAR_FIFO_NEXT
 AIOTypes.h, [300](#)
 AUR_GEN_CLEAR_FIFO_WAIT
 AIOTypes.h, [300](#)
 AUR_GEN_STREAM_STATUS
 AIOTypes.h, [300](#)
 AUR_LOAD_BULK_CALIBRATION_BLOCK
 AIOTypes.h, [300](#)
 AUR_OFFLINE_READWRITE
 AIOTypes.h, [300](#)
 AUR_PROBE_CALFEATURE
 AIOTypes.h, [300](#)
 AUR_READBACK_GLOBAL_STATE
 AIOTypes.h, [300](#)
 AUR_SAVE_GLOBAL_STATE
 AIOTypes.h, [300](#)
 AUR_SELF_TEST_1
 AIOTypes.h, [300](#)
 AUR_START_ACQUIRING_BLOCK
 AIOTypes.h, [300](#)
 AUR_WDG
 AIOTypes.h, [300](#)
 AUR_WDG_STATUS
 AIOTypes.h, [300](#)
 ACCES_DEPRECATED
 AIOTypes.h, [291](#)
 ADBuf
 AIOUSBDevice, [85](#)
 ADBuf_size
 AIOUSBDevice, [85](#)
 ADC_ADMODE
 AIOUSB_ADC.c, [310](#)
 AIOUSB_ADC.h, [321](#)
 ADC_Acquire_Reference_Counts
 AIOUSB_ADC.c, [305](#)
 ADC_BulkAcquire
 AIOUSB_ADC.c, [311](#)
 AIOUSB_ADC.h, [323](#)
 ADC_BulkPoll
 AIOUSB_ADC.c, [312](#)
 AIOUSB_ADC.h, [323](#)
 ADC_CanCalibrate
 AIOUSB_ADC.c, [311](#)
 AIOUSB_ADC.h, [322](#)
 ADC_ClearADBuf
 AIOUSB_ADC.c, [312](#)
 ADC_ClearFastITConfig
 AIOUSB_ADC.c, [312](#)
 ADC_ClockRateForADCProduct
 AIOUSB_ADC.c, [310](#)
 AIOUSB_ADC.h, [322](#)
 ADC_CopyConfig
 AIOUSB_ADC.c, [308](#)
 ADC_CreateADBuf
 AIOUSB_ADC.c, [312](#)
 ADC_CreateFastITConfig
 AIOUSB_ADC.c, [312](#)
 AIOUSB_ADC.h, [323](#)
 ADC_Debug_Register_Settings
 AIOUSB_ADC.c, [313](#)
 ADC_GetADConfigBlock_Registers
 AIOUSB_ADC.c, [312](#)
 AIOUSB_ADC.h, [327](#)
 ADC_GetChannelV
 AIOUSB_ADC.c, [307](#)
 AIOUSB_ADC.h, [319](#)
 ADC_GetConfig
 AIOUSB_ADC.c, [308](#)
 AIOUSB_ADC.h, [321](#)
 ADC_GetConfigRegisters
 AIOUSB_ADC.c, [305](#)
 ADC_GetConfigSize
 AIOUSB_ADC.c, [305](#)
 ADC_GetFastITScanV
 AIOUSB_ADC.c, [313](#)
 AIOUSB_ADC.h, [323](#)
 ADC_GetITScanV
 AIOUSB_ADC.c, [313](#)
 AIOUSB_ADC.h, [324](#)
 ADC_GetMaxClockRate
 AIOUSB_ADC.c, [310](#)
 AIOUSB_ADC.h, [322](#)
 ADC_GetOversample
 AIOUSB_ADC.c, [310](#)
 AIOUSB_ADC.h, [324](#)

- ADC_GetScan
 - AIOUSB_ADC.c, [307](#)
 - AIOUSB_ADC.h, [320](#)
- ADC_GetScanV
 - AIOUSB_ADC.c, [307](#)
 - AIOUSB_ADC.h, [319](#)
- ADC_InitFastITScanV
 - AIOUSB_ADC.c, [312](#)
 - AIOUSB_ADC.h, [323](#)
- ADC_Initialize
 - AIOUSB_ADC.c, [311](#)
 - AIOUSB_ADC.h, [322](#)
- ADC_QueryCal
 - AIOUSB_ADC.c, [311](#)
 - AIOUSB_ADC.h, [322](#)
- ADC_Range1
 - AIOUSB_ADC.c, [308](#)
 - AIOUSB_ADC.h, [321](#)
- ADC_RangeAll
 - AIOUSB_ADC.c, [308](#)
 - AIOUSB_ADC.h, [319](#)
- ADC_ReadADConfigBlock
 - AIOUSB_ADC.c, [305](#)
- ADC_ResetDevice
 - AIOUSB_ADC.c, [305](#)
 - AIOUSB_Core.h, [333](#)
- ADC_ResetFastITScanV
 - AIOUSB_ADC.c, [312](#)
 - AIOUSB_ADC.h, [323](#)
- ADC_SetAllGainCodeAndDiffMode
 - AIOUSB_ADC.c, [310](#)
 - AIOUSB_ADC.h, [326](#)
- ADC_SetCal
 - AIOUSB_ADC.c, [311](#)
 - AIOUSB_ADC.h, [322](#)
- ADC_SetConfig
 - AIOUSB_ADC.c, [308](#)
 - AIOUSB_ADC.h, [321](#)
- ADC_SetFastITScanVChannels
 - AIOUSB_ADC.c, [312](#)
 - AIOUSB_ADC.h, [323](#)
- ADC_SetOversample
 - AIOUSB_ADC.c, [310](#)
 - AIOUSB_ADC.h, [324](#)
- ADC_SetScanLimits
 - AIOUSB_ADC.c, [310](#)
 - AIOUSB_ADC.h, [322](#)
- ADC_VerifyAndCorrectConfigBlock
 - ADCCConfigBlock.c, [209](#)
 - ADCCConfigBlock.h, [212](#)
- ADCCChannels
 - AIOUSBDevice, [84](#)
 - DeviceProperties, [135](#)
- ADCCChannelsPerGroup
 - AIOUSBDevice, [84](#)
 - DeviceProperties, [135](#)
 - mux_settings, [147](#)
- ADCCConfigBlock, [63](#)
 - ADCCConfigBlock.h, [212](#)
 - clock_rate, [64](#)
 - debug, [64](#)
 - device, [63](#)
 - discardFirstSample, [64](#)
 - mux_settings, [64](#)
 - registers, [64](#)
 - size, [63](#)
 - testing, [64](#)
 - timeout, [64](#)
- ADCCConfigBlock.c
 - ADC_VerifyAndCorrectConfigBlock, [209](#)
 - ADCCConfigBlockCopy, [208](#)
 - ADCCConfigBlockGetAIOUSBDevice, [208](#)
 - ADCCConfigBlockGetCalMode, [209](#)
 - ADCCConfigBlockGetClockRate, [211](#)
 - ADCCConfigBlockGetDebug, [209](#)
 - ADCCConfigBlockGetEndChannel, [210](#)
 - ADCCConfigBlockGetGainCode, [209](#)
 - ADCCConfigBlockGetJSONValueOrDefault, [210](#)
 - ADCCConfigBlockGetJSONValueOrInt, [210](#)
 - ADCCConfigBlockGetOversample, [210](#)
 - ADCCConfigBlockGetSize, [209](#)
 - ADCCConfigBlockGetStartChannel, [209](#)
 - ADCCConfigBlockGetTesting, [209](#)
 - ADCCConfigBlockGetTimeout, [210](#)
 - ADCCConfigBlockGetTriggerMode, [210](#)
 - ADCCConfigBlockInit, [209](#)
 - ADCCConfigBlockInitForCounterScan, [209](#)
 - ADCCConfigBlockInitializeDefault, [208](#)
 - ADCCConfigBlockInitializeFromAIOUSBDevice, [208](#)
 - ADCCConfigBlockSetAIOUSBDevice, [208](#)
 - ADCCConfigBlockSetAllGainCodeAndDiffMode, [209](#)
 - ADCCConfigBlockSetCalMode, [209](#)
 - ADCCConfigBlockSetChannelRange, [209](#)
 - ADCCConfigBlockSetClockRate, [211](#)
 - ADCCConfigBlockSetDebug, [209](#)
 - ADCCConfigBlockSetDevice, [208](#)
 - ADCCConfigBlockSetDifferentialMode, [210](#)
 - ADCCConfigBlockSetEndChannel, [209](#)
 - ADCCConfigBlockSetGainCode, [209](#)
 - ADCCConfigBlockSetOversample, [210](#)
 - ADCCConfigBlockSetRangeSingle, [209](#)
 - ADCCConfigBlockSetReference, [210](#)
 - ADCCConfigBlockSetRegister, [209](#)
 - ADCCConfigBlockSetScanAllChannels, [210](#)
 - ADCCConfigBlockSetScanRange, [209](#)
 - ADCCConfigBlockSetSize, [208](#)
 - ADCCConfigBlockSetStartChannel, [209](#)
 - ADCCConfigBlockSetTesting, [209](#)
 - ADCCConfigBlockSetTimeout, [210](#)
 - ADCCConfigBlockSetTriggerEdge, [210](#)
 - ADCCConfigBlockSetTriggerMode, [210](#)
 - ADCCConfigBlockSetTriggerReference, [210](#)
 - ADCCConfigBlockToJSON, [210](#)
 - ADCCConfigBlockToYAML, [210](#)
 - DeleteADCCConfigBlock, [208](#)
 - get_cal_mode, [210](#)
 - get_gain_code, [210](#)
 - is_all_digits, [210](#)
 - NewADCCConfigBlockFromJSON, [211](#)
- ADCCConfigBlock.h
 - ADC_VerifyAndCorrectConfigBlock, [212](#)
 - ADCCConfigBlock, [212](#)
 - ADCCConfigBlockCopy, [214](#)
 - ADCCConfigBlockGetAIOUSBDevice, [214](#)
 - ADCCConfigBlockGetCalMode, [213](#)
 - ADCCConfigBlockGetClockRate, [212](#)
 - ADCCConfigBlockGetDebug, [214](#)
 - ADCCConfigBlockGetEndChannel, [213](#)
 - ADCCConfigBlockGetGainCode, [212](#)
 - ADCCConfigBlockGetOversample, [213](#)
 - ADCCConfigBlockGetSize, [214](#)
 - ADCCConfigBlockGetStartChannel, [213](#)
 - ADCCConfigBlockGetTesting, [214](#)
 - ADCCConfigBlockGetTimeout, [213](#)
 - ADCCConfigBlockGetTriggerMode, [213](#)
 - ADCCConfigBlockInit, [212](#)
 - ADCCConfigBlockInitForCounterScan, [212](#)
 - ADCCConfigBlockInitializeDefault, [212](#)
 - ADCCConfigBlockInitializeFromAIOUSBDevice, [214](#)
 - ADCCConfigBlockSetAIOUSBDevice, [214](#)
 - ADCCConfigBlockSetAllGainCodeAndDiffMode, [212](#)
 - ADCCConfigBlockSetCalMode, [213](#)
 - ADCCConfigBlockSetChannelRange, [213](#)
 - ADCCConfigBlockSetClockRate, [212](#)

- ADCConfigBlockSetDebug, [214](#)
- ADCConfigBlockSetDevice, [214](#)
- ADCConfigBlockSetDifferentialMode, [214](#)
- ADCConfigBlockSetEndChannel, [213](#)
- ADCConfigBlockSetGainCode, [212](#)
- ADCConfigBlockSetOversample, [213](#)
- ADCConfigBlockSetRangeSingle, [214](#)
- ADCConfigBlockSetReference, [213](#)
- ADCConfigBlockSetRegister, [212](#)
- ADCConfigBlockSetScanRange, [213](#)
- ADCConfigBlockSetSize, [214](#)
- ADCConfigBlockSetStartChannel, [213](#)
- ADCConfigBlockSetTesting, [214](#)
- ADCConfigBlockSetTimeout, [213](#)
- ADCConfigBlockSetTriggerEdge, [213](#)
- ADCConfigBlockSetTriggerMode, [213](#)
- ADCConfigBlockToJSON, [214](#)
- ADCConfigBlockToYAML, [213](#)
- ADCMuxSettings, [212](#)
- ADCConfigBlock, [212](#)
- AIIODevice, [212](#)
- DeleteADCConfigBlock, [214](#)
- is_all_digits, [214](#)
- NewADCConfigBlockFromJSON, [214](#)
- ADCConfigBlockCopy
 - ADCConfigBlock.c, [208](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockGetAIIODevice
 - ADCConfigBlock.c, [208](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockGetCalMode
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockGetClockRate
 - ADCConfigBlock.c, [211](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockGetDebug
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockGetEndChannel
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockGetGainCode
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockGetJSONValueOrDefault
 - ADCConfigBlock.c, [210](#)
- ADCConfigBlockGetJSONValueOrInt
 - ADCConfigBlock.c, [210](#)
- ADCConfigBlockGetOversample
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockGetSize
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockGetStartChannel
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockGetTesting
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockGetTimeout
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockGetTriggerMode
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockInit
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockInitForCounterScan
 - ADCConfigBlock.c, [209](#)
- ADCConfigBlock.h, [212](#)
- ADCConfigBlockInitializeDefault
 - ADCConfigBlock.c, [208](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockInitializeFromAIIODevice
 - ADCConfigBlock.c, [208](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetAIIODevice
 - ADCConfigBlock.c, [208](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetAllGainCodeAndDiffMode
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockSetCalMode
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetChannelRange
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetClockRate
 - ADCConfigBlock.c, [211](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockSetDebug
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetDevice
 - ADCConfigBlock.c, [208](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetDifferentialMode
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetEndChannel
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetGainCode
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockSetOversample
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetRangeSingle
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetReference
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetRegister
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [212](#)
- ADCConfigBlockSetScanAllChannels
 - ADCConfigBlock.c, [210](#)
- ADCConfigBlockSetScanRange
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetSize
 - ADCConfigBlock.c, [208](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetStartChannel
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetTesting
 - ADCConfigBlock.c, [209](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockSetTimeout
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetTriggerEdge
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfigBlockSetTriggerMode
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)

- ADCConfigBlockSetTriggerReference
 - ADCConfigBlock.c, [210](#)
- ADCConfigBlockToJSON
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [214](#)
- ADCConfigBlockToYAML
 - ADCConfigBlock.c, [210](#)
 - ADCConfigBlock.h, [213](#)
- ADCConfiguration
 - AIOConfiguration.h, [233](#)
- ADCMUXChannels
 - AIOUSBDevice, [84](#)
 - DeviceProperties, [135](#)
 - mux_settings, [147](#)
- ADCMuxSettings
 - ADCConfigBlock.h, [212](#)
- ADCScanType
 - AIOConfiguration.h, [234](#)
- ADCSetCalFunction
 - AIOConfiguration.h, [234](#)
- ADCalMode
 - AIOTypes.h, [301](#)
- ADConfigBlock
 - ADCConfigBlock.h, [212](#)
- ADConfigBlockToYAML
 - AIOUSB_ADC.c, [313](#)
- ADGainCode
 - AIOTypes.h, [299](#)
- ADRange, [64](#)
 - minVolts, [64](#)
 - range, [64](#)
- ADRegister
 - AIOTypes.h, [298](#)
- AI16_DataPoint, [64](#)
 - AIOUSB::AI16_DataPoint, [65](#)
 - AIOUSB::AI16_InputRange, [70](#)
- AI16_DataPointArray, [66](#)
 - AIOUSB::AI16_DataPoint, [66](#)
 - AIOUSB::AI16_DataPointArray, [66](#)
- AI16_DataSet, [67](#)
 - AIOUSB::AI16_DataPoint, [66](#)
 - AIOUSB::AI16_DataSet, [67](#)
- AI16_InputRange, [69](#)
 - AIOUSB::AI16_InputRange, [70](#)
- AIO_ASSERT
 - AIOTypes.h, [292](#)
- AIO_ASSERT_CONFIG
 - AIOTypes.h, [292](#)
- AIO_ASSERT_DIOBUF
 - AIOTypes.h, [292](#)
- AIO_ASSERT_EXIT
 - AIOTypes.h, [292](#)
- AIO_ASSERT_RET
 - AIOTypes.h, [292](#)
- AIO_ASSERT_USB
 - AIOTypes.h, [292](#)
- AIO_CHAR_ARRAY
 - AIOTuple.h, [286](#)
- AIO_DEBUG_LEVEL
 - AIOUSB_Log.h, [349](#)
- AIO_DEBUG_STR
 - AIOUSB_Log.h, [349](#)
- AIO_DEVEL_STR
 - AIOUSB_Log.h, [349](#)
- AIO_EITHER_TYPE
 - AIOEither.h, [274](#)
- AIO_ERROR
 - AIOTypes.h, [292](#)
- AIO_ERROR_STR
 - AIOUSB_Log.h, [349](#)
- AIO_ERROR_VALUE
 - AIOEither.h, [274](#)
- AIO_FATAL_STR
 - AIOUSB_Log.h, [349](#)
- AIO_FIFO_INTERFACE
 - AIOFifo, [80](#)
 - AIOFifo.h, [278](#)
 - new_aio_fifo, [148](#)
- AIO_INFO_STR
 - AIOUSB_Log.h, [349](#)
- AIO_MAKE_ERROR
 - AIOTypes.h, [292](#)
- AIO_NUMBER
 - AIOTypes.h, [293](#)
- AIO_OPTIONS
 - aiocommon.c, [381](#)
 - aiocommon.h, [381](#)
- AIO_PRODUCT_GROUP
 - AIOProductTypes.h, [284](#)
- AIO_RANGE
 - AIOProductTypes.h, [284](#)
- AIO_RESET_STR
 - AIOUSB_Log.h, [349](#)
- AIO_SCAN_TYPE
 - AIOTypes.h, [293](#)
- AIO_WARN_STR
 - AIOUSB_Log.h, [349](#)
- AIOArgument, [72](#)
 - actual_size, [72](#)
 - config, [72](#)
 - debug, [72](#)
 - size, [72](#)
 - threaded, [72](#)
- AIOArgumentInitialize
 - AIOConfiguration.c, [233](#)
 - AIOConfiguration.h, [234](#)
- AIOArguments, [73](#)
 - device_args, [73](#)
 - number_arguments, [73](#)
- AIOBuf, [73](#)
 - AIOBuf.h, [216](#)
 - defined, [73](#)
 - endpos, [73](#)
 - size, [73](#)
 - type, [73](#)
- AIOBuf.c
 - AIOBufGetIterator, [215](#)
 - AIOBufGetRaw, [215](#)
 - AIOBufGetSize, [215](#)
 - AIOBufGetTotalSize, [215](#)
 - AIOBufGetType, [215](#)
 - AIOBufGetTypeSize, [215](#)
 - AIOBufIteratorGetValue, [215](#)
 - AIOBufIteratorIsValid, [215](#)
 - AIOBufIteratorNext, [215](#)
 - AIOBufRead, [215](#)
 - AIOBufWrite, [215](#)
 - DeleteAIOBuf, [215](#)
 - NewAIOBuf, [215](#)
- AIOBuf.h
 - AIOBuf, [216](#)
 - AIOBufGetIterator, [216](#)
 - AIOBufGetRaw, [216](#)
 - AIOBufGetSize, [216](#)
 - AIOBufGetType, [216](#)
 - AIOBufIterator, [216](#)
 - AIOBufIteratorGetValue, [216](#)
 - AIOBufIteratorIsValid, [217](#)
 - AIOBufIteratorNext, [217](#)
 - AIOBufRead, [216](#)
 - AIOBufType, [216](#)
 - AIOBufWrite, [216](#)
 - DeleteAIOBuf, [216](#)
 - NewAIOBuf, [216](#)

- AIOBufGetIterator
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- AIOBufGetRaw
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- AIOBufGetSize
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- AIOBufGetTotalSize
 - AIOBuf.c, [215](#)
- AIOBufGetType
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- AIOBufGetTypeSize
 - AIOBuf.c, [215](#)
- AIOBufIterator
 - AIOBuf.h, [216](#)
- AIOBufIteratorGetValue
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- AIOBufIteratorIsValid
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [217](#)
- AIOBufIteratorNext
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [217](#)
- AIOBufRead
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- AIOBufType
 - AIOBuf.h, [216](#)
- AIOBufWrite
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- AIOBufferType
 - AIOTypes.h, [293](#)
- AIOChannelMask, [74](#)
 - active_signals, [74](#)
 - number_signals, [74](#)
 - pos, [74](#)
 - signal_index, [74](#)
 - signal_indices, [74](#)
 - signals, [74](#)
 - size, [74](#)
 - strrep, [74](#)
 - strrepsmall, [74](#)
- AIOChannelMask.c
 - AIOChannelMaskGetMask, [219](#)
 - AIOChannelMaskGetMaskAtIndex, [218](#)
 - AIOChannelMaskGetSize, [218](#)
 - AIOChannelMaskIndices, [218](#)
 - AIOChannelMaskNextIndex, [218](#)
 - AIOChannelMaskNumberChannels, [218](#)
 - AIOChannelMaskNumberSignals, [218](#)
 - AIOChannelMaskSetMaskAtIndex, [218](#)
 - AIOChannelMaskSetMaskFromInt, [218](#)
 - AIOChannelMaskSetMaskFromStr, [218](#)
 - AIOChannelMaskToString, [219](#)
 - AIOChannelMaskToStringAtIndex, [219](#)
 - DeleteAIOChannelMask, [218](#)
 - NewAIOChannelMask, [217](#)
 - NewAIOChannelMaskFromChr, [219](#)
 - NewAIOChannelMaskFromStr, [219](#)
- AIOChannelMask.h
 - AIOChannelMaskGetMask, [221](#)
 - AIOChannelMaskGetMaskAtIndex, [221](#)
 - AIOChannelMaskGetSize, [221](#)
 - AIOChannelMaskIndices, [221](#)
 - AIOChannelMaskNextIndex, [221](#)
 - AIOChannelMaskNumberChannels, [221](#)
 - AIOChannelMaskNumberSignals, [221](#)
 - AIOChannelMaskSetMaskAtIndex, [222](#)
 - AIOChannelMaskSetMaskFromInt, [222](#)
 - AIOChannelMaskSetMaskFromStr, [222](#)
 - AIOChannelMaskToString, [221](#)
 - AIOChannelMaskToStringAtIndex, [221](#)
 - aio_channel_obj, [220](#)
 - BIT_LENGTH, [220](#)
 - DeleteAIOChannelMask, [220](#)
 - NewAIOChannelMask, [220](#)
 - NewAIOChannelMaskFromChr, [221](#)
 - NewAIOChannelMaskFromStr, [220](#)
- AIOChannelMaskGetMask
 - AIOChannelMask.c, [219](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskGetMaskAtIndex
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskGetSize
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskIndices
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskNextIndex
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskNumberChannels
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskNumberSignals
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskSetMaskAtIndex
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [222](#)
- AIOChannelMaskSetMaskFromInt
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [222](#)
- AIOChannelMaskSetMaskFromStr
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [222](#)
- AIOChannelMaskToString
 - AIOChannelMask.c, [219](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelMaskToStringAtIndex
 - AIOChannelMask.c, [219](#)
 - AIOChannelMask.h, [221](#)
- AIOChannelRange
 - AIOChannelRange.h, [223](#)
- AIOChannelRange.c
 - AIOChannelRangeGetEnd, [223](#)
 - AIOChannelRangeGetGain, [223](#)
 - AIOChannelRangeGetStart, [223](#)
 - AIOChannelRangeToStr, [223](#)
 - aio_channel_range_error, [223](#)
 - DeleteAIOChannelRange, [223](#)
 - lookup_voltage_range, [223](#)
 - NewAIOChannelRangeFromStr, [223](#)
 - STATE, [222](#)
- AIOChannelRange.h
 - AIOChannelRange, [223](#)
 - AIOChannelRangeGetEnd, [224](#)
 - AIOChannelRangeGetGain, [224](#)
 - AIOChannelRangeGetStart, [224](#)
 - AIOChannelRangeToStr, [224](#)
 - aio_channel_range_error, [224](#)
 - DeleteAIOChannelRange, [224](#)
 - NewAIOChannelRangeFromStr, [224](#)
- AIOChannelRangeGetEnd
 - AIOChannelRange.c, [223](#)
 - AIOChannelRange.h, [224](#)
- AIOChannelRangeGetGain

- AIOChannelRange.c, [223](#)
 - AIOChannelRange.h, [224](#)
- AIOChannelRangeGetStart
 - AIOChannelRange.c, [223](#)
 - AIOChannelRange.h, [224](#)
- AIOChannelRangeTmp, [74](#)
 - AIOCommandLine.h, [229](#)
 - end_channel, [74](#)
 - gaincode, [74](#)
 - start_channel, [74](#)
- AIOChannelRangeToStr
 - AIOChannelRange.c, [223](#)
 - AIOChannelRange.h, [224](#)
- AIOCmd, [75](#)
 - AIOCmd.h, [225](#)
 - channel, [75](#)
 - num_channels, [75](#)
 - num_samples, [75](#)
 - num_scans, [75](#)
 - stop_scan, [75](#)
 - stop_scan_arg, [75](#)
- AIOCmd.c
 - DeleteAIOCmd, [224](#)
 - NewAIOCmd, [224](#)
 - NewAIOCmdFromJSON, [224](#)
- AIOCmd.h
 - AIOCmd, [225](#)
 - DeleteAIOCmd, [225](#)
 - NewAIOCmd, [225](#)
 - NewAIOCmdFromJSON, [225](#)
- AIOCommandCode
 - AIOTypes.h, [301](#)
- AIOCommandLine.c
 - AIOCommandLineListDevices, [227](#)
 - AIOCommandLineOptionsListDevices, [227](#)
 - AIOCommandLineOptionsOverrideADCCConfigBlock, [227](#)
 - AIOCommandLineOverrideADCCConfigBlock, [227](#)
 - AIOGetChannelRange, [227](#)
 - AIOPrintUsage, [226](#)
 - AIOProcessCmdline, [226](#)
 - AIOProcessCommandLine, [226](#)
 - DeleteAIOCommandLineOptions, [227](#)
 - NewAIOCommandLineOptionsFromDefaultOptions, [227](#)
 - NewDefaultAIOCommandLineOptions, [227](#)
 - opterr, [228](#)
 - optind, [228](#)
- AIOCommandLine.h
 - AIOChannelRangeTmp, [229](#)
 - AIOCommandLineListDevices, [232](#)
 - AIOCommandLineOptions, [229](#)
 - AIOCommandLineOptionsListDevices, [230](#)
 - AIOCommandLineOptionsOverrideADCCConfigBlock, [232](#)
 - AIOCommandLineOverrideADCCConfigBlock, [232](#)
 - AIOGetChannelRange, [230](#)
 - AIOPrintUsage, [230](#)
 - AIOProcessCmdline, [230](#)
 - AIOProcessCommandLine, [230](#)
 - CNTS, [229](#)
 - DUMP, [229](#)
 - DeleteAIOCommandLineOptions, [230](#)
 - DeviceEnum, [229](#)
 - JCONF, [229](#)
 - NewAIOCommandLineOptionsFromDefaultOptions, [230](#)
 - NewDefaultAIOCommandLineOptions, [230](#)
 - REPEAT, [229](#)
- AIOCommandLineListDevices
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [232](#)
- AIOCommandLineOptions
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [232](#)
- AIOCommandLineOptionsListDevices
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [232](#)
- AIOCommandLineOptionsOverrideADCCConfigBlock
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [232](#)
- AIOCommandLineOverrideADCCConfigBlock
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [232](#)
- AIOConfiguration
 - AIOConfiguration.h, [233](#)
- AIOConfiguration.c
 - AIOArgumentInitialize, [233](#)
 - AIOConfigurationInitialize, [233](#)
 - AIOConfigurationSetDebug, [233](#)
 - AIOConfigurationSetTimeout, [233](#)
 - DeleteAIOConfiguration, [233](#)
 - NewAIOConfiguration, [233](#)
- AIOConfiguration.h
 - ADCCConfiguration, [233](#)
 - ADCScanType, [234](#)
 - ADCSetCalFunction, [234](#)
 - AIOArgumentInitialize, [234](#)
 - AIOConfiguration, [233](#)
 - AIOConfigurationInitialize, [234](#)
 - AIOConfigurationSetDebug, [234](#)
 - AIOConfigurationSetTimeout, [234](#)
 - AIOContinuousBufConfiguration, [233](#)
 - ConfigurationType, [233](#)
 - NewAIOConfiguration, [234](#)
- AIOConfigurationInitialize
 - AIOConfiguration.c, [233](#)
 - AIOConfiguration.h, [234](#)
- AIOConfigurationSetDebug
 - AIOConfiguration.c, [233](#)
 - AIOConfiguration.h, [234](#)
- AIOConfigurationSetTimeout
 - AIOConfiguration.c, [233](#)
 - AIOConfiguration.h, [234](#)
- AIOCommandLineOptions, [75](#)
 - adccconfig_json, [77](#)
 - aiobuf_json, [76](#)
 - AIOCommandLine.h, [229](#)
 - block_size, [76](#)
 - buffer_size, [76](#)
 - calibration, [76](#)
 - clock_rate, [76](#)
 - counts, [76](#)
 - debug_level, [76](#)
 - default_aiobuf_json, [76](#)
 - default_clock_rate, [76](#)
 - default_end_channel, [76](#)
 - default_num_channels, [76](#)
 - default_num_oversamples, [76](#)
 - default_num_scans, [76](#)
 - default_start_channel, [76](#)
 - end_channel, [76](#)
 - gain_code, [76](#)
 - index, [76](#)
 - num_channels, [76](#)
 - num_oversamples, [76](#)
 - num_scans, [76](#)
 - number_ranges, [76](#)
 - outfile, [76](#)
 - pass_through, [76](#)
 - physical, [76](#)
 - ranges, [77](#)
 - rate_limit, [76](#)
 - repeat_number, [76](#)
 - reset, [76](#)
 - slow_acquire, [76](#)
 - start_channel, [76](#)
 - verbose, [76](#)
 - with_timing, [76](#)

- AIOConfiguration.c, [233](#)
- AIOConfiguration.h, [234](#)
- AIOContinuousBufConfiguration
 - AIOConfiguration.h, [233](#)
- AIOContinuousBuf, [77](#)
 - AIOContinuousBuffer.h, [249](#)
 - base_size, [78](#)
 - block_size, [78](#)
 - buffer, [78](#)
 - bytes_processed, [78](#)
 - callback, [78](#)
 - counter_control, [78](#)
 - countsbuf, [78](#)
 - debug, [78](#)
 - DeviceIndex, [78](#)
 - exitcode, [78](#)
 - fifo, [78](#)
 - hz, [78](#)
 - lock, [78](#)
 - mask, [78](#)
 - num_channels, [78](#)
 - num_oversamples, [78](#)
 - num_scans, [78](#)
 - PopN, [78](#)
 - PushN, [78](#)
 - scans_read, [78](#)
 - size, [78](#)
 - start_scanning, [78](#)
 - status, [78](#)
 - tattr, [78](#)
 - testing, [78](#)
 - timeout, [78](#)
 - type, [78](#)
 - unit_size, [78](#)
 - work, [78](#)
 - worker, [78](#)
- AIOContinuousBuf_BufSizeForCounts
 - AIOContinuousBuffer.c, [240](#)
- AIOContinuousBuf_GetDeviceIndex
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_GetOverSample
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_InitConfiguration
 - AIOContinuousBuffer.c, [239](#)
- AIOContinuousBuf_NumberChannels
 - AIOContinuousBuffer.c, [243](#)
- AIOContinuousBuf_NumberSignals
 - AIOContinuousBuffer.c, [243](#)
- AIOContinuousBuf_SaveConfig
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SendPreConfig
 - AIOContinuousBuffer.c, [239](#)
- AIOContinuousBuf_SetAllGainCodeAndDiffMode
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SetCallback
 - AIOContinuousBuffer.c, [240](#)
- AIOContinuousBuf_SetChannelRange
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SetChannelRangeGain
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SetDeviceIndex
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SetDiscardFirstSample
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SetOversample
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SetStartAndEndChannel
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_SetTesting
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuf_CallbackStart
 - AIOContinuousBuffer.c, [244](#)
- AIOContinuousBuffer.h, [252](#)
- AIOContinuousBuf_CallbackStartCallbackWithAcquisition-Function
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBufCleanup
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [255](#)
- AIOContinuousBuf_CountScansAvailable
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBuf_CountsAvailable
 - AIOContinuousBuffer.c, [240](#)
- AIOContinuousBuf_End
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBuf_ForceTerminateAcquisition
 - AIOContinuousBuffer.c, [239](#)
- AIOContinuousBuf_ForceTerminateAcquisitionOverrun
 - AIOContinuousBuffer.c, [239](#)
- AIOContinuousBuf_GetADCCConfigBlock
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBuf_GetBaseSize
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBuf_GetBufferSize
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBuf_GetCallback
 - AIOContinuousBuffer.c, [242](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBuf_GetClock
 - AIOContinuousBuffer.c, [242](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBuf_GetDataAvailable
 - AIOContinuousBuffer.c, [240](#)
- AIOContinuousBuf_GetDebug
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBuf_GetDeviceIndex
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBuf_GetExitCode
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBuf_GetNumberChannels
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBuf_GetNumberOfScansToRead
 - AIOContinuousBuffer.c, [242](#)
- AIOContinuousBuf_GetNumberSamplesPerScan
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBuf_GetNumberScans
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBuf_GetOversample
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBuf_GetReadPosition
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBuf_GetRemainingSize
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBuf_GetRunStatus
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBuf_GetScansRead
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)

- AIOContinuousBufGetSize
 - AIOContinuousBuffer.c, [240](#)
- AIOContinuousBufGetSizeNumElements
 - AIOContinuousBuffer.c, [240](#)
- AIOContinuousBufGetStatus
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBufGetStreamingBlockSize
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufGetTesting
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufGetTimeout
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufGetTotalSamplesExpected
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufGetUnitSize
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufGetWritePosition
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBufInitADCConfigBlock
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufInitConfiguration
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [249](#)
- AIOContinuousBufInitiateCallbackAcquisition
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBufLoadCounters
 - AIOContinuousBuffer.c, [243](#)
- AIOContinuousBufLock
 - AIOContinuousBuffer.c, [244](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufMode
 - AIOTypes.h, [293](#)
- AIOContinuousBufNumberChannels
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufNumberSamplesAvailable
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufNumberSignals
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufNumberWriteSamplesRemaining
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufPending
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBufPopN
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufPreSetup
 - AIOContinuousBuffer.c, [243](#)
- AIOContinuousBufPushN
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufRead
 - AIOContinuousBuffer.c, [244](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufReadCompleteScanCounts
 - AIOContinuousBuffer.c, [242](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufReadIntegerNumberOfScans
 - AIOContinuousBuffer.c, [242](#)
- AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufReadIntegerScanCounts
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBufReadNSamples
 - AIOContinuousBuffer.c, [243](#)
- AIOContinuousBufReadSingle
 - AIOContinuousBuffer.c, [242](#)
- AIOContinuousBufReset
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufResetDevice
 - AIOContinuousBuffer.c, [244](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufSaveConfig
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufSendPreConfig
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetAllGainCodeAndDiffMode
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetBaseSize
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetCallback
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetChannelMask
 - AIOContinuousBuffer.c, [242](#)
 - AIOContinuousBuffer.h, [251](#)
- AIOContinuousBufSetChannelRange
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufSetClock
 - AIOContinuousBuffer.c, [242](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufSetCountsBuffer
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufSetDebug
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufSetDefaultModeForCounterScan
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBufSetDeviceIndex
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufSetDiscardFirstSample
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetNumberChannels
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetNumberOfScansToRead
 - AIOContinuousBuffer.c, [242](#)
- AIOContinuousBufSetNumberScans
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufSetOverSample
 - AIOContinuousBuffer.c, [245](#)
- AIOContinuousBufSetOversample
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetStartAndEndChannel
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetStreamingBlockSize
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetTesting

- AIOContinuousBuffer.c, [245](#)
- AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetTimeout
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufSetUnitSize
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [250](#)
- AIOContinuousBufSetVoltsBuffer
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufSimpleSetupConfig
 - AIOContinuousBuffer.c, [245](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufStart
 - AIOContinuousBuffer.c, [242](#)
- AIOContinuousBufStopAcquisition
 - AIOContinuousBuffer.c, [242](#)
 - AIOContinuousBuffer.h, [253](#)
- AIOContinuousBufToJSON
 - AIOContinuousBuffer.c, [246](#)
 - AIOContinuousBuffer.h, [255](#)
- AIOContinuousBufUnlock
 - AIOContinuousBuffer.c, [244](#)
 - AIOContinuousBuffer.h, [252](#)
- AIOContinuousBufWrite
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [254](#)
- AIOContinuousBufWriteCounts
 - AIOContinuousBuffer.c, [243](#)
 - AIOContinuousBuffer.h, [255](#)
- AIOContinuousBuffer.c
 - AIOContinuousBuf_BufSizeForCounts, [240](#)
 - AIOContinuousBuf_GetDeviceIndex, [245](#)
 - AIOContinuousBuf_GetOverSample, [245](#)
 - AIOContinuousBuf_InitConfiguration, [239](#)
 - AIOContinuousBuf_NumberChannels, [243](#)
 - AIOContinuousBuf_NumberSignals, [243](#)
 - AIOContinuousBuf_SaveConfig, [245](#)
 - AIOContinuousBuf_SendPreConfig, [239](#)
 - AIOContinuousBuf_SetAllGainCodeAndDiffMode, [245](#)
 - AIOContinuousBuf_SetCallback, [240](#)
 - AIOContinuousBuf_SetChannelRange, [245](#)
 - AIOContinuousBuf_SetChannelRangeGain, [245](#)
 - AIOContinuousBuf_SetDeviceIndex, [245](#)
 - AIOContinuousBuf_SetDiscardFirstSample, [245](#)
 - AIOContinuousBuf_SetOversample, [245](#)
 - AIOContinuousBuf_SetStartAndEndChannel, [245](#)
 - AIOContinuousBuf_SetTesting, [245](#)
 - AIOContinuousBuf_CallbackStart, [244](#)
 - AIOContinuousBuf_CallbackStartCallbackWith-
AcquisitionFunction, [243](#)
 - AIOContinuousBufCleanup, [243](#)
 - AIOContinuousBufCountScansAvailable, [240](#)
 - AIOContinuousBufCountsAvailable, [240](#)
 - AIOContinuousBufEnd, [245](#)
 - AIOContinuousBufForceTerminateAcquisition, [239](#)
 - AIOContinuousBufForceTerminateAcquisitionOverrun, [239](#)
 - AIOContinuousBufGetADCConfigBlock, [240](#)
 - AIOContinuousBufGetBaseSize, [239](#)
 - AIOContinuousBufGetBufferSize, [239](#)
 - AIOContinuousBufGetCallback, [242](#)
 - AIOContinuousBufGetClock, [242](#)
 - AIOContinuousBufGetDataAvailable, [240](#)
 - AIOContinuousBufGetDebug, [245](#)
 - AIOContinuousBufGetDeviceIndex, [245](#)
 - AIOContinuousBufGetExitCode, [240](#)
 - AIOContinuousBufGetNumberChannels, [239](#)
 - AIOContinuousBufGetNumberOfScansToRead, [242](#)
 - AIOContinuousBufGetNumberSamplesPerScan, [243](#)
 - AIOContinuousBufGetNumberScans, [240](#)
 - AIOContinuousBufGetOversample, [245](#)
 - AIOContinuousBufGetReadPosition, [240](#)
 - AIOContinuousBufGetRemainingSize, [240](#)
 - AIOContinuousBufGetRunStatus, [240](#)
 - AIOContinuousBufGetScansRead, [240](#)
 - AIOContinuousBufGetSize, [240](#)
 - AIOContinuousBufGetSizeNumElements, [240](#)
 - AIOContinuousBufGetStatus, [240](#)
 - AIOContinuousBufGetStreamingBlockSize, [240](#)
 - AIOContinuousBufGetTesting, [245](#)
 - AIOContinuousBufGetTimeout, [245](#)
 - AIOContinuousBufGetTotalSamplesExpected, [243](#)
 - AIOContinuousBufGetUnitSize, [240](#)
 - AIOContinuousBufGetWritePosition, [240](#)
 - AIOContinuousBufInitADCConfigBlock, [239](#)
 - AIOContinuousBufInitConfiguration, [239](#)
 - AIOContinuousBufInitiateCallbackAcquisition, [243](#)
 - AIOContinuousBufLoadCounters, [243](#)
 - AIOContinuousBufLock, [244](#)
 - AIOContinuousBufNumberChannels, [243](#)
 - AIOContinuousBufNumberSamplesAvailable, [243](#)
 - AIOContinuousBufNumberSignals, [243](#)
 - AIOContinuousBufNumberWriteSamplesRemaining, [243](#)
 - AIOContinuousBufPending, [240](#)
 - AIOContinuousBufPopN, [239](#)
 - AIOContinuousBufPreSetup, [243](#)
 - AIOContinuousBufPushN, [239](#)
 - AIOContinuousBufRead, [244](#)
 - AIOContinuousBufReadCompleteScanCounts, [242](#)
 - AIOContinuousBufReadIntegerNumberOfScans, [242](#)
 - AIOContinuousBufReadIntegerScanCounts, [240](#)
 - AIOContinuousBufReadNSamples, [243](#)
 - AIOContinuousBufReadSingle, [242](#)
 - AIOContinuousBufReset, [240](#)
 - AIOContinuousBufResetDevice, [244](#)
 - AIOContinuousBufSaveConfig, [245](#)
 - AIOContinuousBufSendPreConfig, [240](#)
 - AIOContinuousBufSetAllGainCodeAndDiffMode, [245](#)
 - AIOContinuousBufSetBaseSize, [239](#)
 - AIOContinuousBufSetCallback, [240](#)
 - AIOContinuousBufSetChannelMask, [242](#)
 - AIOContinuousBufSetChannelRange, [245](#)
 - AIOContinuousBufSetClock, [242](#)
 - AIOContinuousBufSetCountsBuffer, [240](#)
 - AIOContinuousBufSetDebug, [245](#)
 - AIOContinuousBufSetDefaultModeForCounterScan, [245](#)
 - AIOContinuousBufSetDeviceIndex, [245](#)
 - AIOContinuousBufSetDiscardFirstSample, [245](#)
 - AIOContinuousBufSetNumberChannels, [239](#)
 - AIOContinuousBufSetNumberOfScansToRead, [242](#)
 - AIOContinuousBufSetNumberScans, [240](#)
 - AIOContinuousBufSetOverSample, [245](#)
 - AIOContinuousBufSetOversample, [245](#)
 - AIOContinuousBufSetStartAndEndChannel, [245](#)
 - AIOContinuousBufSetStreamingBlockSize, [240](#)
 - AIOContinuousBufSetTesting, [245](#)
 - AIOContinuousBufSetTimeout, [245](#)
 - AIOContinuousBufSetUnitSize, [240](#)
 - AIOContinuousBufSetVoltsBuffer, [240](#)
 - AIOContinuousBufSimpleSetupConfig, [245](#)
 - AIOContinuousBufStart, [242](#)
 - AIOContinuousBufStopAcquisition, [242](#)
 - AIOContinuousBufToJSON, [246](#)
 - AIOContinuousBufUnlock, [244](#)
 - AIOContinuousBufWrite, [243](#)
 - AIOContinuousBufWriteCounts, [243](#)
 - BaseSizeRange, [246](#)
 - continue_running, [243](#)
 - ConvertCountsToVoltsFunction, [238](#)

- DeleteAIOContinuousBuf, [240](#)
- GetJSONValueOrDefault, [245](#)
- NewAIOContinuousBuf, [239](#)
- NewAIOContinuousBufForCounts, [239](#)
- NewAIOContinuousBufForVolts, [239](#)
- NewAIOContinuousBufFromJSON, [246](#)
- number_to_read, [243](#)
- RangeValueLookup, [238](#)
- RawCountsWorkFunction, [238](#)
- ResetCounters, [243](#)
- SetConfig, [243](#)
- StartStreaming, [243](#)
- TrueFalse, [246](#)
- AIOContinuousBuffer.h
 - AIOContinuousBuf, [249](#)
 - AIOContinuousBufCallbackStart, [252](#)
 - AIOContinuousBufCallbackStartCallbackWithAcquisitionFunction, [253](#)
 - AIOContinuousBufCleanup, [255](#)
 - AIOContinuousBufCountScansAvailable, [254](#)
 - AIOContinuousBufEnd, [254](#)
 - AIOContinuousBufGetADCCConfigBlock, [250](#)
 - AIOContinuousBufGetBaseSize, [250](#)
 - AIOContinuousBufGetBufferSize, [250](#)
 - AIOContinuousBufGetCallback, [250](#)
 - AIOContinuousBufGetClock, [254](#)
 - AIOContinuousBufGetDebug, [252](#)
 - AIOContinuousBufGetDeviceIndex, [250](#)
 - AIOContinuousBufGetExitCode, [253](#)
 - AIOContinuousBufGetNumberChannels, [250](#)
 - AIOContinuousBufGetNumberSamplesPerScan, [252](#)
 - AIOContinuousBufGetNumberScans, [252](#)
 - AIOContinuousBufGetOversample, [250](#)
 - AIOContinuousBufGetReadPosition, [253](#)
 - AIOContinuousBufGetRemainingSize, [253](#)
 - AIOContinuousBufGetRunStatus, [253](#)
 - AIOContinuousBufGetScansRead, [253](#)
 - AIOContinuousBufGetStatus, [253](#)
 - AIOContinuousBufGetStreamingBlockSize, [250](#)
 - AIOContinuousBufGetTesting, [250](#)
 - AIOContinuousBufGetTimeout, [252](#)
 - AIOContinuousBufGetTotalSamplesExpected, [252](#)
 - AIOContinuousBufGetUnitSize, [250](#)
 - AIOContinuousBufGetWritePosition, [253](#)
 - AIOContinuousBufInitADCCConfigBlock, [250](#)
 - AIOContinuousBufInitConfiguration, [249](#)
 - AIOContinuousBufInitiateCallbackAcquisition, [253](#)
 - AIOContinuousBufLock, [252](#)
 - AIOContinuousBufNumberChannels, [250](#)
 - AIOContinuousBufNumberSamplesAvailable, [252](#)
 - AIOContinuousBufNumberSignals, [252](#)
 - AIOContinuousBufNumberWriteSamplesRemaining, [252](#)
 - AIOContinuousBufPending, [253](#)
 - AIOContinuousBufPopN, [252](#)
 - AIOContinuousBufPushN, [252](#)
 - AIOContinuousBufRead, [254](#)
 - AIOContinuousBufReadCompleteScanCounts, [254](#)
 - AIOContinuousBufReadIntegerNumberOfScans, [254](#)
 - AIOContinuousBufReadIntegerScanCounts, [253](#)
 - AIOContinuousBufReset, [252](#)
 - AIOContinuousBufResetDevice, [252](#)
 - AIOContinuousBufSaveConfig, [252](#)
 - AIOContinuousBufSendPreConfig, [250](#)
 - AIOContinuousBufSetAllGainCodeAndDiffMode, [250](#)
 - AIOContinuousBufSetBaseSize, [250](#)
 - AIOContinuousBufSetCallback, [250](#)
 - AIOContinuousBufSetChannelMask, [251](#)
 - AIOContinuousBufSetChannelRange, [252](#)
 - AIOContinuousBufSetClock, [254](#)
 - AIOContinuousBufSetCountsBuffer, [254](#)
 - AIOContinuousBufSetDebug, [252](#)
 - AIOContinuousBufSetDeviceIndex, [252](#)
 - AIOContinuousBufSetDiscardFirstSample, [250](#)
 - AIOContinuousBufSetNumberChannels, [250](#)
 - AIOContinuousBufSetNumberScans, [252](#)
 - AIOContinuousBufSetOversample, [250](#)
 - AIOContinuousBufSetStartAndEndChannel, [250](#)
 - AIOContinuousBufSetStreamingBlockSize, [250](#)
 - AIOContinuousBufSetTesting, [250](#)
 - AIOContinuousBufSetTimeout, [252](#)
 - AIOContinuousBufSetUnitSize, [250](#)
 - AIOContinuousBufSetVoltsBuffer, [254](#)
 - AIOContinuousBufSimpleSetupConfig, [254](#)
 - AIOContinuousBufStopAcquisition, [253](#)
 - AIOContinuousBufToJSON, [255](#)
 - AIOContinuousBufUnlock, [252](#)
 - AIOContinuousBufWrite, [254](#)
 - AIOContinuousBufWriteCounts, [255](#)
 - AIOUSB_WorkFn, [249](#)
 - DeleteAIOContinuousBuf, [249](#)
 - NewAIOContinuousBuf, [249](#)
 - NewAIOContinuousBufForCounts, [249](#)
 - NewAIOContinuousBufForVolts, [249](#)
 - NewAIOContinuousBufFromJSON, [255](#)
 - ROOTCLOCK, [249](#)
- AIOCountsConverter
 - AIOCountsConverter.h, [257](#)
- AIOCountsConverter.c
 - AIOCountsConverterConvert, [256](#)
 - AIOCountsConverterConvertAllAvailableScans, [256](#)
 - AIOCountsConverterConvertFifo, [256](#)
 - AIOCountsConverterConvertNScans, [256](#)
 - AIOCountsConverterReset, [256](#)
 - Convert, [256](#)
 - default_out, [256](#)
 - DeleteAIOCountsConverter, [256](#)
 - DeleteAIOGainRange, [256](#)
 - enhanced_out, [256](#)
 - NewAIOCountsConverter, [256](#)
 - NewAIOCountsConverterWithBuffer, [256](#)
 - NewAIOCountsConverterWithScanLimiter, [256](#)
 - NewAIOGainRangeFromADCCConfigBlock, [256](#)
- AIOCountsConverter.h
 - AIOCountsConverter, [257](#)
 - AIOCountsConverterConvert, [258](#)
 - AIOCountsConverterConvertAllAvailableScans, [258](#)
 - AIOCountsConverterConvertFifo, [258](#)
 - AIOCountsConverterConvertNScans, [258](#)
 - AIOCountsConverterReset, [258](#)
 - DeleteAIOCountsConverter, [258](#)
 - DeleteAIOGainRange, [258](#)
 - NewAIOCountsConverter, [257](#)
 - NewAIOCountsConverterFromAIOContinuousBuf, [257](#)
 - NewAIOCountsConverterWithBuffer, [257](#)
 - NewAIOCountsConverterWithScanLimiter, [257](#)
 - NewAIOGainRangeFromADCCConfigBlock, [258](#)
- AIOCountsConverterConvert
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- AIOCountsConverterConvertAllAvailableScans
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- AIOCountsConverterConvertFifo
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- AIOCountsConverterConvertNScans
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- AIOCountsConverterReset
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- AIODeviceInfo, [79](#)

- AIODDeviceInfo.h, [259](#)
 - Counters, [79](#)
 - DIOBytes, [79](#)
 - Name, [79](#)
 - NameSize, [79](#)
 - PID, [79](#)
- AIODDeviceInfo.c
 - AIODDeviceInfoGet, [258](#)
 - AIODDeviceInfoGetCounters, [258](#)
 - AIODDeviceInfoGetDIOBytes, [258](#)
 - AIODDeviceInfoGetName, [258](#)
 - DeleteAIODDeviceInfo, [258](#)
 - NewAIODDeviceInfo, [258](#)
- AIODDeviceInfo.h
 - AIODDeviceInfo, [259](#)
 - AIODDeviceInfoGet, [259](#)
 - AIODDeviceInfoGetCounters, [259](#)
 - AIODDeviceInfoGetDIOBytes, [259](#)
 - AIODDeviceInfoGetName, [259](#)
 - DeleteAIODDeviceInfo, [259](#)
 - NewAIODDeviceInfo, [259](#)
- AIODDeviceInfoGet
 - AIODDeviceInfo.c, [258](#)
 - AIODDeviceInfo.h, [259](#)
- AIODDeviceInfoGetCounters
 - AIODDeviceInfo.c, [258](#)
 - AIODDeviceInfo.h, [259](#)
- AIODDeviceInfoGetDIOBytes
 - AIODDeviceInfo.c, [258](#)
 - AIODDeviceInfo.h, [259](#)
- AIODDeviceInfoGetName
 - AIODDeviceInfo.c, [258](#)
 - AIODDeviceInfo.h, [259](#)
- AIODDeviceQuery, [79](#)
 - AIODDeviceQuery.h, [263](#)
 - index, [80](#)
 - name, [79](#)
 - nameSize, [79](#)
 - numCounters, [80](#)
 - numDIOBytes, [79](#)
 - productID, [79](#)
- AIODDeviceQuery.c
 - AIODDeviceQueryGetIndex, [261](#)
 - AIODDeviceQueryGetName, [262](#)
 - AIODDeviceQueryGetNumCounters, [262](#)
 - AIODDeviceQueryGetNumDIOBytes, [262](#)
 - AIODDeviceQueryGetProductID, [261](#)
 - AIODDeviceQueryNameSize, [261](#)
 - AIODDeviceQueryToRepr, [261](#)
 - AIODDeviceQueryToStr, [261](#)
 - DeleteAIODDeviceQuery, [261](#)
 - NewAIODDeviceQuery, [260](#)
- AIODDeviceQuery.h
 - AIODDeviceQuery, [263](#)
 - AIODDeviceQueryGetIndex, [265](#)
 - AIODDeviceQueryGetName, [264](#)
 - AIODDeviceQueryGetNumCounters, [265](#)
 - AIODDeviceQueryGetNumDIOBytes, [264](#)
 - AIODDeviceQueryGetProductID, [264](#)
 - AIODDeviceQueryNameSize, [264](#)
 - AIODDeviceQueryToRepr, [264](#)
 - AIODDeviceQueryToStr, [264](#)
 - DeleteAIODDeviceQuery, [263](#)
 - NewAIODDeviceQuery, [263](#)
- AIODDeviceQueryGetIndex
 - AIODDeviceQuery.c, [261](#)
 - AIODDeviceQuery.h, [265](#)
- AIODDeviceQueryGetName
 - AIODDeviceQuery.c, [262](#)
 - AIODDeviceQuery.h, [264](#)
- AIODDeviceQueryGetNumCounters
 - AIODDeviceQuery.c, [262](#)
- AIODDeviceQuery.h, [265](#)
 - AIODDeviceQueryGetNumDIOBytes, [262](#)
 - AIODDeviceQueryGetProductID, [261](#)
 - AIODDeviceQueryNameSize, [261](#)
 - AIODDeviceQueryToRepr, [261](#)
 - AIODDeviceQueryToStr, [261](#)
- AIODDeviceQueryToRepr
 - AIODDeviceQuery.c, [261](#)
 - AIODDeviceQuery.h, [264](#)
- AIODDeviceQueryToStr
 - AIODDeviceQuery.c, [261](#)
 - AIODDeviceQuery.h, [264](#)
- AIODDeviceTable.c
 - AIODDeviceTableAddDeviceToDeviceTable, [268](#)
 - AIODDeviceTableAddDeviceToDeviceTableWithUSB-Device, [268](#)
 - AIODDeviceTableClearDevices, [268](#)
 - AIODDeviceTableGetAIOUSBDeviceAtIndex, [268](#)
 - AIODDeviceTableGetDeviceAtIndex, [267](#)
 - AIODDeviceTableGetUSBDeviceAtIndex, [267](#)
 - AIODDeviceTableInit, [266](#)
 - AIODDeviceTablePopulateTable, [268](#)
 - AIODDeviceTablePopulateTableTest, [268](#)
 - AIODDeviceTableSetDeviceID, [268](#)
 - AIOUSB_Cleanup, [266](#)
 - AIOUSB_EnsureOpen, [267](#)
 - AIOUSB_Exit, [268](#)
 - AIOUSB_GetAllDevices, [268](#)
 - AIOUSB_Init, [268](#)
 - AIOUSB_InitTest, [266](#)
 - AIOUSB_IsInit, [266](#)
 - AIOUSB_Reset, [268](#)
 - AIOUSB_SetInit, [266](#)
 - AIOUSBGetError, [268](#)
 - aiousbInit, [268](#)
 - ClearAIODDeviceTable, [268](#)
 - ClearDevices, [268](#)
 - CloseAllDevices, [268](#)
 - deviceTable, [268](#)
 - GetDevices, [267](#)
 - GetSafeDeviceName, [267](#)
 - NUM_PROD_NAMES, [266](#)
 - ProductIDToName, [266](#)
 - ProductNameToID, [266](#)
 - QueryDeviceInfo, [266](#)
- AIODDeviceTable.h
 - AIODDeviceTableAddDeviceToDeviceTable, [269](#)
 - AIODDeviceTableAddDeviceToDeviceTableWithUSB-Device, [269](#)
 - AIODDeviceTableClearDevices, [270](#)
 - AIODDeviceTableGetAIOUSBDeviceAtIndex, [270](#)
 - AIODDeviceTableGetDeviceAtIndex, [270](#)
 - AIODDeviceTableGetUSBDeviceAtIndex, [270](#)
 - AIODDeviceTableInit, [271](#)
 - AIODDeviceTablePopulateTable, [270](#)
 - AIODDeviceTablePopulateTableTest, [270](#)
 - AIOUSB_EnsureOpen, [271](#)
 - AIOUSB_Exit, [271](#)
 - AIOUSB_GetAllDevices, [272](#)
 - AIOUSB_Init, [271](#)
 - AIOUSB_IsInit, [271](#)
 - AIOUSB_Reset, [271](#)
 - AIOUSBGetError, [272](#)
 - ClearAIODDeviceTable, [271](#)
 - ClearDevices, [270](#)
 - CloseAllDevices, [272](#)
 - deviceTable, [272](#)
 - GetDevices, [270](#)

- GetSafeDeviceName, [270](#)
- ProductIDToName, [271](#)
- ProductNameToID, [271](#)
- QueryDeviceInfo, [270](#)
- AIODeviceTableAddDeviceToDeviceTable
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [269](#)
- AIODeviceTableAddDeviceToDeviceTableWithUSBDevice
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [269](#)
- AIODeviceTableClearDevices
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [270](#)
- AIODeviceTableGetAIOUSBDeviceAtIndex
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [270](#)
- AIODeviceTableGetDeviceAtIndex
 - AIODeviceTable.c, [267](#)
 - AIODeviceTable.h, [270](#)
- AIODeviceTableGetUSBDeviceAtIndex
 - AIODeviceTable.c, [267](#)
 - AIODeviceTable.h, [270](#)
- AIODeviceTableInit
 - AIODeviceTable.c, [266](#)
 - AIODeviceTable.h, [271](#)
- AIODeviceTablePopulateTable
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [270](#)
- AIODeviceTablePopulateTableTest
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [270](#)
- AIODeviceTableSetDeviceID
 - AIODeviceTable.c, [268](#)
- AIODisplayType
 - AIOUSB_Properties.h, [352](#)
- AIOEither
 - AIOEither.h, [274](#)
- AIOEither.c
 - AIOEitherClear, [273](#)
 - AIOEitherGetLeft, [273](#)
 - AIOEitherGetRight, [273](#)
 - AIOEitherHasError, [273](#)
 - AIOEitherSetLeft, [273](#)
 - AIOEitherSetRight, [273](#)
 - AIOEitherToAIONumber, [273](#)
 - AIOEitherToAIORetType, [273](#)
 - AIOEitherToDouble, [273](#)
 - AIOEitherToInt, [273](#)
 - AIOEitherToShort, [273](#)
 - AIOEitherToString, [273](#)
 - AIOEitherToUnsigned, [273](#)
 - LOOKUP, [272](#)
- AIOEither.h
 - AIO_EITHER_TYPE, [274](#)
 - AIO_ERROR_VALUE, [274](#)
 - AIOEither, [274](#)
 - AIOEitherClear, [275](#)
 - AIOEitherGetLeft, [275](#)
 - AIOEitherGetRight, [275](#)
 - AIOEitherHasError, [275](#)
 - AIOEitherSetLeft, [275](#)
 - AIOEitherSetRight, [275](#)
 - AIOEitherToAIONumber, [275](#)
 - AIOEitherToAIORetType, [275](#)
 - AIOEitherToDouble, [275](#)
 - AIOEitherToInt, [275](#)
 - AIOEitherToShort, [275](#)
 - AIOEitherToString, [275](#)
 - AIOEitherToUnsigned, [275](#)
- AIOEitherClear
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherGetLeft
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherGetRight
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherHasError
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherSetLeft
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherSetRight
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherToAIONumber
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherToAIORetType
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherToDouble
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherToInt
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherToShort
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherToString
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOEitherToUnsigned
 - AIOEither.c, [273](#)
 - AIOEither.h, [275](#)
- AIOError
 - AIOUSB_Core.h, [333](#)
- AIOFifo, [80](#)
 - AIOFifo.h, [279](#)
 - LOCKING_MECHANISM, [80](#)
- AIOFifo.c
 - AIOFifoAllOrNoneInitialize, [277](#)
 - AIOFifoGetRefSize, [277](#)
 - AIOFifoGetSize, [276](#)
 - AIOFifoGetSizeNumElements, [276](#)
 - AIOFifoInitialize, [277](#)
 - AIOFifoRead, [277](#)
 - AIOFifoReadAllOrNone, [277](#)
 - AIOFifoReadPosition, [277](#)
 - AIOFifoReadSize, [277](#)
 - AIOFifoReadSizeNumElements, [277](#)
 - AIOFifoReset, [277](#)
 - AIOFifoResize, [277](#)
 - AIOFifoWrite, [277](#)
 - AIOFifoWriteAllOrNone, [277](#)
 - AIOFifoWritePosition, [277](#)
 - AIOFifoWriteSizeRemaining, [276](#)
 - AIOFifoWriteSizeRemainingNumElements, [276](#)
 - DeleteAIOFifo, [277](#)
 - DeleteAIOFifoTYPE, [277](#)
 - delta, [276](#)
 - increment, [277](#)
 - LOOKUP, [276](#)
 - NewAIOFifo, [277](#)
 - NewAIOFifoAllOrNone, [277](#)
 - NewAIOFifoTYPE, [277](#)
 - Pop, [277](#)
 - PopN, [277](#)
 - Push, [277](#)
 - PushN, [277](#)
 - rdelta, [276](#)

- AIOFifo.h
 - AIOFifo, [279](#)
 - AIOFifoGetRefSize, [280](#)
 - AIOFifoGetSize, [280](#)
 - AIOFifoGetSizeNumElements, [280](#)
 - AIOFifoRead, [279](#)
 - AIOFifoReadAllOrNone, [279](#)
 - AIOFifoReadPosition, [280](#)
 - AIOFifoReadSize, [280](#)
 - AIOFifoReadSizeNumElements, [280](#)
 - AIOFifoReset, [279](#)
 - AIOFifoResize, [280](#)
 - AIOFifoTYPE, [279](#)
 - AIOFifoWrite, [279](#)
 - AIOFifoWriteAllOrNone, [279](#)
 - AIOFifoWritePosition, [280](#)
 - AIOFifoWriteSizeRemaining, [280](#)
 - AIOFifoWriteSizeRemainingNumElements, [280](#)
 - DeleteAIOFifo, [279](#)
 - GRAB_RESOURCE, [278](#)
 - INPUT_TYPE, [279](#)
 - LOCKING_MECHANISM, [278](#)
 - NewAIOFifo, [279](#)
 - NewAIOFifoTYPE, [280](#)
 - PopN, [280](#)
 - Push, [280](#)
 - PushN, [280](#)
 - RELEASE_RESOURCE, [278](#)
 - TYPE, [279](#)
- AIOFifoAllOrNoneInitialize
 - AIOFifo.c, [277](#)
- AIOFifoGetRefSize
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
- AIOFifoGetSize
 - AIOFifo.c, [276](#)
 - AIOFifo.h, [280](#)
- AIOFifoGetSizeNumElements
 - AIOFifo.c, [276](#)
 - AIOFifo.h, [280](#)
- AIOFifoInitialize
 - AIOFifo.c, [277](#)
- AIOFifoRead
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [279](#)
- AIOFifoReadAllOrNone
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [279](#)
- AIOFifoReadPosition
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
- AIOFifoReadSize
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
- AIOFifoReadSizeNumElements
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
- AIOFifoReset
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [279](#)
- AIOFifoResize
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
- AIOFifoTYPE
 - AIOFifo.h, [279](#)
- AIOFifoWrite
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [279](#)
- AIOFifoWriteAllOrNone
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [279](#)
- AIOFifoWritePosition
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
- AIOFifoWriteSizeRemaining
 - AIOFifo.c, [276](#)
 - AIOFifo.h, [280](#)
- AIOFifoWriteSizeRemainingNumElements
 - AIOFifo.c, [276](#)
 - AIOFifo.h, [280](#)
- AIOGainRange, [81](#)
 - max, [81](#)
 - min, [81](#)
- AIOGetChannelRange
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [230](#)
- AIOList.c
 - Deleteint, [280](#)
 - Deleteintlist, [281](#)
 - intToString, [280](#)
 - intlistFirst, [281](#)
 - intlistInsert, [281](#)
 - intlistSize, [281](#)
 - intlistToString, [281](#)
 - Newintlist, [281](#)
 - newone, [281](#)
- AIOList.h
 - Deleteintlist, [282](#)
 - foreach_CStringArray_p, [282](#)
 - foreach_int, [282](#)
 - intlistFirst, [282](#)
 - intlistInsert, [282](#)
 - intlistSize, [282](#)
 - intlistToString, [282](#)
 - Newintlist, [282](#)
 - TAIL_Q_LIST, [281](#), [282](#)
- AIOOption
 - AIOUSB_Core.h, [333](#)
- AIOPrintUsage
 - AIOCommandLine.c, [226](#)
 - AIOCommandLine.h, [230](#)
- AIOProcessCmdline
 - AIOCommandLine.c, [226](#)
 - AIOCommandLine.h, [230](#)
- AIOProcessCommandLine
 - AIOCommandLine.c, [226](#)
 - AIOCommandLine.h, [230](#)
- AIOProductGroup, [81](#)
 - AIOProductTypes.h, [285](#)
- AIOProductGroupContains
 - AIOProductTypes.c, [283](#)
 - AIOProductTypes.h, [285](#)
- AIOProductRange, [81](#)
 - AIOProductTypes.h, [285](#)
- AIOProductRangeEnd
 - AIOProductTypes.c, [283](#)
 - AIOProductTypes.h, [285](#)
- AIOProductRangeStart
 - AIOProductTypes.c, [283](#)
 - AIOProductTypes.h, [285](#)
- AIOProductTypes.c
 - AIOProductGroupContains, [283](#)
 - AIOProductRangeEnd, [283](#)
 - AIOProductRangeStart, [283](#)
 - DeleteAIOProductGroup, [283](#)
 - DeleteAIOProductRange, [283](#)
 - groupcpy, [283](#)
 - NewAIOProductGroup, [283](#)
 - NewAIOProductRange, [283](#)
- AIOProductTypes.h
 - AIO_RANGE, [284](#)
 - AIOProductGroup, [285](#)
 - AIOProductGroupContains, [285](#)
 - AIOProductRange, [285](#)

- AIOProductRangeEnd, [285](#)
 - AIOProductRangeStart, [285](#)
 - DeleteAIOProductGroup, [285](#)
 - DeleteAIOProductRange, [285](#)
 - groupcpy, [285](#)
 - NUMARGS, [284](#)
 - NewAIOProductGroup, [285](#)
 - NewAIOProductRange, [285](#)
- AIORESULT
 - AIOTypes.h, [293](#)
- AIORET_TYPE
 - AIOTypes.h, [293](#)
- AIOTUPLE2
 - AIOTuple.h, [286](#)
- AIOTUPLE2_PTR
 - AIOTuple.h, [286](#)
- AIOTUPLE2_TO_STR
 - AIOTuple.h, [286](#)
- AIOTUPLE2_TYPE
 - AIOTuple.h, [286](#)
- AIOTuple.h
 - AIO_CHAR_ARRAY, [286](#)
 - AIOTUPLE2, [286](#)
 - AIOTUPLE2_PTR, [286](#)
 - AIOTUPLE2_TO_STR, [286](#)
 - AIOTUPLE2_TYPE, [286](#)
- AIOTypes.h
 - ACCES_DEPRECATED, [291](#)
 - ADCalMode, [301](#)
 - ADGainCode, [299](#)
 - ADRegister, [298](#)
 - AIO_ASSERT, [292](#)
 - AIO_ASSERT_EXIT, [292](#)
 - AIO_ASSERT_RET, [292](#)
 - AIO_ASSERT_USB, [292](#)
 - AIO_ERROR, [292](#)
 - AIO_MAKE_ERROR, [292](#)
 - AIO_NUMBER, [293](#)
 - AIO_SCAN_TYPE, [293](#)
 - AIOBufferType, [293](#)
 - AIOCommandCode, [301](#)
 - AIOContinuousBufMode, [293](#)
 - AIORESULT, [293](#)
 - AIORET_TYPE, [293](#)
 - AIOUSB_BOOL, [293](#)
 - AIOUSB_BOOL_VAL, [294](#)
 - AUR_CBUF_EXIT, [291](#)
 - AUR_CBUF_SETUP, [291](#)
 - COUNTS, [293](#)
 - CREATE_ENUM, [291](#)
 - DACRange, [296](#)
 - EXPORTED_FUNCTION, [291](#)
 - EnumStringLookup, [293](#)
 - FIFO_Method, [296](#)
 - FIRST_ENUM, [291](#)
 - foreach_array, [291](#)
 - G_BREAKPOINT, [293](#)
 - G_STMT_END, [292](#)
 - G_STMT_START, [292](#)
 - GCC_VERSION, [291](#)
 - HAS_PTHREAD, [291](#)
 - LAMBDA, [291](#)
 - LAST_ENUM, [291](#)
 - MAX, [291](#)
 - MAX_VALUE, [291](#)
 - MIN, [291](#)
 - MIN_VALUE, [291](#)
 - NUMBER_CHANNELS, [291](#)
 - ProductIDS, [294](#)
 - ROOTCLOCK, [293](#)
 - ResultCode, [297](#)
 - THREAD_STATUS, [293](#)
 - Ushort_Array, [293](#)
 - VALID_ENUM, [291](#)
 - VALID_PRODUCT, [291](#)
 - VENDOR_REQUEST, [299](#)
- AIOUSB, [61](#)
 - operator<<, [62](#)
- AIOUSB::AI16_DataPoint
 - AI16_DataPoint, [65](#)
 - AI16_DataPointArray, [66](#)
 - AI16_DataSet, [66](#)
 - AnalogInputSubsystem, [66](#)
 - channel, [66](#)
 - counts, [66](#)
 - differentialMode, [66](#)
 - getChannel, [65](#)
 - getCounts, [65](#)
 - getRange, [65](#)
 - getRangeText, [65](#)
 - getVolts, [66](#)
 - isDifferentialMode, [65](#)
 - range, [66](#)
 - std::vector< AI16_DataPoint >, [66](#)
 - toString, [66](#)
- AIOUSB::AI16_DataPointArray
 - AI16_DataPointArray, [66](#)
- AIOUSB::AI16_DataSet
 - ~AI16_DataSet, [67](#)
 - AI16_DataSet, [67](#)
 - AnalogInputSubsystem, [69](#)
 - calMode, [69](#)
 - discardFirstSample, [69](#)
 - getCalMode, [68](#)
 - getOverSample, [68](#)
 - getPoints, [68](#)
 - getSubsystem, [68](#)
 - getTimeStamp, [68](#)
 - getTriggerMode, [68](#)
 - isDiscardFirstSample, [68](#)
 - overSample, [69](#)
 - points, [69](#)
 - print, [69](#)
 - subsystem, [69](#)
 - timeStamp, [69](#)
 - triggerMode, [69](#)
- AIOUSB::AI16_InputRange
 - ~AI16_InputRange, [70](#)
 - AI16_DataPoint, [70](#)
 - AI16_InputRange, [70](#)
 - AnalogInputSubsystem, [70](#)
 - setRange, [70](#)
- AIOUSB::AO16_AnalogOutputSubsystem
 - AO16_AnalogOutputSubsystem, [113](#)
 - countsToVolts, [115](#)
 - getRange, [113](#)
 - getRangeText, [113](#)
 - outputRange, [116](#)
 - RANGE_10V, [115](#)
 - RANGE_5V, [115](#)
 - setRange, [113](#)
 - voltsToCounts, [115](#)
 - writeVolts, [113](#)
- AIOUSB::AO16_OutputRange
 - ~AO16_OutputRange, [116](#)
 - AO16_AnalogOutputSubsystem, [117](#)
 - AO16_OutputRange, [116](#)
 - setRange, [116](#)
- AIOUSB::AnalogIORange
 - ~AnalogIORange, [107](#)
 - AnalogIORange, [107](#)
 - countsToVolts, [109](#)
 - getRange, [107](#)
 - maxCounts, [109](#)

- maxVolts, [109](#)
- minCounts, [109](#)
- minVolts, [109](#)
- range, [109](#)
- rangeCounts, [109](#)
- rangeVolts, [109](#)
- setCountRange, [107](#)
- setRange, [107](#)
- setVoltRange, [109](#)
- voltsToCounts, [109](#)
- AIOWUSB::AnalogInputSubsystem
 - ~AnalogInputSubsystem, [89](#)
 - AnalogInputSubsystem, [89](#)
 - autoCalFeature, [106](#)
 - autoConfig, [106](#)
 - calMode, [106](#)
 - calibrate, [98](#)
 - channelsPerGroup, [106](#)
 - clearFIFO, [102](#)
 - configBlockSize, [106](#)
 - countsToVolts, [102](#)
 - differentialMode, [106](#)
 - endChannel, [106](#)
 - getCalMode, [92](#)
 - getChannelsPerGroup, [90](#)
 - getClock, [98](#)
 - getNumChannels, [90](#)
 - getNumMUXChannels, [90](#)
 - getOverSample, [96](#)
 - getRange, [93](#)
 - getRangeText, [90](#)
 - getStreamingBlockSize, [97](#)
 - getTriggerMode, [93](#)
 - inputRange, [106](#)
 - isAutoCalPresent, [90](#)
 - isAutoConfig, [91](#)
 - isDifferentialMode, [94](#)
 - isDiscardFirstSample, [92](#)
 - MAX_CHANNELS, [106](#)
 - MAX_COUNTS, [105](#)
 - MIN_COUNTS, [105](#)
 - numChannels, [106](#)
 - numMUXChannels, [106](#)
 - overSample, [106](#)
 - print, [89](#)
 - RANGE_0_10V, [104](#)
 - RANGE_0_1V, [104](#)
 - RANGE_0_2V, [104](#)
 - RANGE_0_5V, [104](#)
 - RANGE_10V, [104](#)
 - RANGE_1V, [105](#)
 - RANGE_2V, [104](#)
 - RANGE_5V, [104](#)
 - RANGE_TEXT, [105](#)
 - read, [99](#)
 - readBulkBuffer, [106](#)
 - readBulkNext, [102](#)
 - readBulkSamplesAvailable, [101](#)
 - readBulkSamplesRequested, [106](#)
 - readBulkSamplesRetrieved, [106](#)
 - readBulkStart, [101](#)
 - readConfig, [91](#)
 - readCounts, [99](#)
 - readVolts, [101](#)
 - setAutoConfig, [91](#)
 - setCalMode, [92](#)
 - setCalibrationTable, [97](#)
 - setClock, [98](#)
 - setDifferentialMode, [95](#)
 - setDiscardFirstSample, [92](#)
 - setOverSample, [96](#)
 - setRange, [94](#)
 - setRangeAndDiffMode, [96](#)
 - setScanRange, [89](#)
 - setStreamingBlockSize, [97](#)
 - setTriggerMode, [93](#)
 - startChannel, [106](#)
 - triggerMode, [106](#)
 - USB_AI16_Family, [103](#)
 - voltsToCounts, [103](#)
 - writeConfig, [91](#)
- AIOWUSB::AnalogOutputSubsystem
 - ~AnalogOutputSubsystem, [110](#)
 - AnalogOutputSubsystem, [110](#)
 - getNumChannels, [110](#)
 - maxCounts, [111](#)
 - minCounts, [111](#)
 - numChannels, [111](#)
 - print, [110](#)
 - writeCounts, [111](#)
- AIOWUSB::BoolArray
 - BoolArray, [117](#)
- AIOWUSB::Counter
 - Counter, [121](#)
 - counterIndex, [123](#)
 - CounterSubsystem, [122](#)
 - getDeviceIndex, [121](#)
 - MODE_ONE_SHOT, [122](#)
 - parent, [123](#)
 - readCount, [122](#)
 - readCountAndSetModeAndCount, [122](#)
 - readCountAndStatus, [122](#)
 - setCount, [121](#)
 - setMode, [121](#)
 - setModeAndCount, [121](#)
- AIOWUSB::CounterSubsystem
 - ~CounterSubsystem, [124](#)
 - Counter, [126](#)
 - CounterSubsystem, [124](#)
 - counters, [126](#)
 - getCounter, [124](#)
 - getNumCounterBlocks, [124](#)
 - getNumCounters, [124](#)
 - numCounterBlocks, [126](#)
 - numCounters, [126](#)
 - print, [124](#)
 - readCounts, [125](#)
 - selectGate, [125](#)
 - startClock, [125](#)
 - stopClock, [126](#)
 - USB_AI16_Family, [126](#)
 - USB_CTR_15_Family, [126](#)
 - USB_DIO_32_Family, [126](#)
- AIOWUSB::DA12_AnalogOutputSubsystem
 - countsToVolts, [131](#)
 - DA12_AnalogOutputSubsystem, [128](#)
 - getRange, [128](#)
 - getRangeText, [128](#)
 - outputRange, [132](#)
 - RANGE_10V, [132](#)
 - RANGE_5V, [132](#)
 - setRange, [129](#)
 - voltsToCounts, [131](#)
 - writeVolts, [129, 131](#)
- AIOWUSB::DA12_OutputRange
 - ~DA12_OutputRange, [133](#)
 - DA12_AnalogOutputSubsystem, [133](#)
 - DA12_OutputRange, [133](#)
 - setRange, [133](#)
- AIOWUSB::DIOStreamSubsystem
 - ~DIOStreamSubsystem, [142](#)
 - clearFIFO, [145](#)
 - clockHz, [145](#)
 - close, [144](#)

- DIOStreamSubsystem, [142](#)
- getClock, [143](#)
- getStreamingBlockSize, [143](#)
- open, [144](#)
- print, [142](#)
- read, [144](#)
- setClock, [143](#)
- setStreamingBlockSize, [143](#)
- stopClock, [144](#)
- write, [144](#)
- AIOWUSB::DeviceSubsystem
 - ~DeviceSubsystem, [136](#)
 - DeviceSubsystem, [136](#)
 - getDeviceIndex, [136](#)
 - getParent, [136](#)
 - parent, [136](#)
 - print, [136](#)
- AIOWUSB::DigitalIOSubsystem
 - ~DigitalIOSubsystem, [137](#)
 - bitsToBytes, [137](#)
 - bytesToBits, [137](#)
 - configure, [138](#)
 - DigitalIOSubsystem, [137](#)
 - getConfiguration, [139](#)
 - getNumChannels, [137](#)
 - getNumPorts, [137](#)
 - getNumTristateChannels, [138](#)
 - getNumTristateGroups, [138](#)
 - numChannels, [140](#)
 - numPorts, [140](#)
 - numTristateChannels, [140](#)
 - numTristateGroups, [140](#)
 - print, [137](#)
 - read, [139](#)
 - USB_AI16_Family, [140](#)
 - USB_AO16_Family, [140](#)
 - USB_DIO_Family, [140](#)
 - write, [140](#)
 - writeValues, [140](#)
- AIOWUSB::DoubleArray
 - DoubleArray, [145](#)
- AIOWUSB::IllegalArgumentException
 - IllegalArgumentException, [146](#)
- AIOWUSB::IntArray
 - IntArray, [146](#)
- AIOWUSB::OperationFailedException
 - OperationFailedException, [148](#)
- AIOWUSB::OutputVoltagePoint
 - channel, [151](#)
 - OutputVoltagePoint, [151](#)
 - volts, [151](#)
- AIOWUSB::OutputVoltagePointArray
 - OutputVoltagePointArray, [152](#)
- AIOWUSB::StringArray
 - StringArray, [152](#)
- AIOWUSB::UCharArray
 - UCharArray, [157](#)
- AIOWUSB::USB_AI16_Family
 - adc, [159](#)
 - analogInputSubsystem, [159](#)
 - counterSubsystem, [159](#)
 - ctr, [159](#)
 - digitalIOSubsystem, [159](#)
 - dio, [159](#)
 - getSupportedProductIDs, [158](#)
 - getSupportedProductNames, [158](#)
 - initialize, [158](#)
 - isSupportedProductID, [158](#)
 - print, [158](#)
 - supportedProductIDs, [159](#)
 - USBDeviceManager, [159](#)
- AIOWUSB::USB_AIO16_Family
 - adc, [161](#)
 - analogInputSubsystem, [162](#)
 - analogOutputSubsystem, [162](#)
 - counterSubsystem, [162](#)
 - ctr, [162](#)
 - dac, [161](#)
 - digitalIOSubsystem, [162](#)
 - dio, [162](#)
 - getSupportedProductNames, [161](#)
 - initialize, [161](#)
 - print, [161](#)
 - supportedProductIDs, [162](#)
- AIOWUSB::USB_AO16_Family
 - analogOutputSubsystem, [164](#)
 - dac, [164](#)
 - digitalIOSubsystem, [164](#)
 - dio, [164](#)
 - getSupportedProductIDs, [163](#)
 - getSupportedProductNames, [163](#)
 - initialize, [163](#)
 - isSupportedProductID, [164](#)
 - print, [164](#)
 - supportedProductIDs, [164](#)
 - USBDeviceManager, [164](#)
- AIOWUSB::USB_CTR_15_Family
 - counterSubsystem, [167](#)
 - ctr, [166](#)
 - initialize, [166](#)
 - print, [166](#)
- AIOWUSB::USB_DA12_8A_Family
 - dac, [169](#)
 - initialize, [168](#)
 - print, [168](#)
- AIOWUSB::USB_DA12_8E_Family
 - dac, [171](#)
 - initialize, [170](#)
 - print, [171](#)
- AIOWUSB::USB_DIO_16_Family
 - dio, [173](#)
 - diostream, [173](#)
 - initialize, [172](#)
 - print, [173](#)
- AIOWUSB::USB_DIO_32_Family
 - counterSubsystem, [176](#)
 - ctr, [175](#)
 - dio, [175](#)
 - initialize, [175](#)
 - print, [175](#)
- AIOWUSB::USB_DIO_Family
 - digitalIOSubsystem, [178](#)
 - dio, [178](#)
 - getSupportedProductNames, [177](#)
 - initialize, [177](#)
 - print, [177](#)
 - supportedProductIDs, [178](#)
 - USBDeviceManager, [178](#)
- AIOWUSB::USBDeviceArray
 - USBDeviceArray, [179](#)
- AIOWUSB::USBDeviceBase
 - ~USBDeviceBase, [181](#)
 - AnalogInputSubsystem, [183](#)
 - clearFIFO, [181](#)
 - customEEPROMRead, [183](#)
 - customEEPROMWrite, [182](#)
 - DIOStreamSubsystem, [183](#)
 - deviceIndex, [183](#)
 - getCommTimeout, [182](#)
 - getDeviceIndex, [181](#)
 - getMiscClock, [181](#)
 - getName, [181](#)
 - getProductID, [181](#)
 - getSerialNumber, [181](#)

- getStreamingBlockSize, [181](#)
- name, [183](#)
- print, [181](#)
- productId, [183](#)
- reset, [182](#)
- serialNumber, [183](#)
- setCommTimeout, [182](#)
- setMiscClock, [181](#)
- setStreamingBlockSize, [181](#)
- USBDeviceBase, [181](#)
- USBDeviceManager, [183](#)
- AIOUSB::USBDeviceManager
 - ~USBDeviceManager, [185](#)
 - close, [188](#)
 - deviceList, [189](#)
 - emptyDeviceList, [185](#)
 - getAIOUSBVersion, [185](#)
 - getAIOUSBVersionDate, [185](#)
 - getDeviceByProductId, [188](#), [189](#)
 - getDeviceBySerialNumber, [189](#)
 - getResultCodeAsString, [187](#)
 - isOpen, [187](#)
 - listDevices, [187](#)
 - open, [187](#)
 - openStatus, [189](#)
 - print, [185](#)
 - printDevices, [185](#)
 - productIdToName, [185](#), [186](#)
 - productNameToID, [186](#), [187](#)
 - scanForDevices, [188](#)
 - USBDeviceManager, [185](#)
- AIOUSB::UShortArray
 - UShortArray, [190](#)
- AIOUSB_ADC.c
 - ADC_ADMODE, [310](#)
 - ADC_BulkAcquire, [311](#)
 - ADC_BulkPoll, [312](#)
 - ADC_CanCalibrate, [311](#)
 - ADC_ClearADBuf, [312](#)
 - ADC_ClearFastITConfig, [312](#)
 - ADC_CopyConfig, [308](#)
 - ADC_CreateADBuf, [312](#)
 - ADC_CreateFastITConfig, [312](#)
 - ADC_GetChannelV, [307](#)
 - ADC_GetConfig, [308](#)
 - ADC_GetConfigRegisters, [305](#)
 - ADC_GetConfigSize, [305](#)
 - ADC_GetFastITScanV, [313](#)
 - ADC_GetITScanV, [313](#)
 - ADC_GetMaxClockRate, [310](#)
 - ADC_GetOversample, [310](#)
 - ADC_GetScan, [307](#)
 - ADC_GetScanV, [307](#)
 - ADC_InitFastITScanV, [312](#)
 - ADC_Initialize, [311](#)
 - ADC_QueryCal, [311](#)
 - ADC_Range1, [308](#)
 - ADC_RangeAll, [308](#)
 - ADC_ReadADConfigBlock, [305](#)
 - ADC_ResetDevice, [305](#)
 - ADC_ResetFastITScanV, [312](#)
 - ADC_SetCal, [311](#)
 - ADC_SetConfig, [308](#)
 - ADC_SetOversample, [310](#)
 - ADC_SetScanLimits, [310](#)
 - ADConfigBlockToYAML, [313](#)
 - AIOUSB_GetCalMode, [316](#)
 - AIOUSB_GetEndChannel, [316](#)
 - AIOUSB_GetGainCode, [314](#)
 - AIOUSB_GetOversample, [316](#)
 - AIOUSB_GetRegister, [314](#)
 - AIOUSB_GetScan, [305](#)
 - AIOUSB_GetStartChannel, [316](#)
 - AIOUSB_GetTriggerMode, [316](#)
 - AIOUSB_IsDifferentialMode, [316](#)
 - AIOUSB_SetCalMode, [316](#)
 - AIOUSB_SetDifferentialMode, [316](#)
 - AIOUSB_SetGainCode, [315](#)
 - AIOUSB_SetOversample, [316](#)
 - AIOUSB_SetRangeSingle, [314](#)
 - AIOUSB_SetRegister, [314](#)
 - AIOUSB_SetScanRange, [316](#)
 - AIOUSB_SetTriggerMode, [316](#)
 - AIOUSB_VoltsToCounts, [306](#)
 - adRanges, [317](#)
 - adc_get_bulk_data, [305](#)
 - adcblock_valid_channel_settings, [308](#)
 - adcblock_valid_size, [308](#)
 - adcblock_valid_trigger_settings, [308](#)
 - ConfigureAndBulkAcquire, [314](#)
 - CreateSmartBuffer, [312](#)
 - dRef, [317](#)
 - DoLoadCalTable, [314](#)
 - GetHiRef, [314](#)
 - ReadConfigBlock, [305](#)
 - valid_config_block, [308](#)
 - WriteConfigBlock, [305](#)
- AIOUSB_ADC.h
 - ADC_ADMODE, [321](#)
 - ADC_BulkAcquire, [323](#)
 - ADC_BulkPoll, [323](#)
 - ADC_CanCalibrate, [322](#)
 - ADC_CreateFastITConfig, [323](#)
 - ADC_GetChannelV, [319](#)
 - ADC_GetConfig, [321](#)
 - ADC_GetFastITScanV, [323](#)
 - ADC_GetITScanV, [324](#)
 - ADC_GetMaxClockRate, [322](#)
 - ADC_GetOversample, [324](#)
 - ADC_GetScan, [320](#)
 - ADC_GetScanV, [319](#)
 - ADC_InitFastITScanV, [323](#)
 - ADC_Initialize, [322](#)
 - ADC_QueryCal, [322](#)
 - ADC_Range1, [321](#)
 - ADC_RangeAll, [319](#)
 - ADC_ResetFastITScanV, [323](#)
 - ADC_SetCal, [322](#)
 - ADC_SetConfig, [321](#)
 - ADC_SetOversample, [324](#)
 - ADC_SetScanLimits, [322](#)
 - AIOUSB_GetCalMode, [326](#)
 - AIOUSB_GetEndChannel, [326](#)
 - AIOUSB_GetGainCode, [324](#)
 - AIOUSB_GetOversample, [326](#)
 - AIOUSB_GetRegister, [327](#)
 - AIOUSB_GetScan, [320](#)
 - AIOUSB_GetStartChannel, [326](#)
 - AIOUSB_GetTriggerMode, [326](#)
 - AIOUSB_IsDifferentialMode, [325](#)
 - AIOUSB_SetCalMode, [326](#)
 - AIOUSB_SetDifferentialMode, [326](#)
 - AIOUSB_SetGainCode, [324](#)
 - AIOUSB_SetOversample, [326](#)
 - AIOUSB_SetRegister, [327](#)
 - AIOUSB_SetScanRange, [326](#)
 - AIOUSB_SetTriggerMode, [326](#)
 - CreateSmartBuffer, [327](#)
 - ReadConfigBlock, [324](#)
 - WriteConfigBlock, [324](#)
- AIOUSB_ADC_ExternalCal
 - AIOUSB_ADC.c, [316](#)
 - AIOUSB_ADC.h, [325](#)
- AIOUSB_ADC_InternalCal

- AIOUSB_ADC.c, [314](#)
 - AIOUSB_ADC.h, [327](#)
- AIOUSB_ADC_LoadCalTable
 - AIOUSB_Core.c, [330](#)
 - AIOUSB_Core.h, [334](#)
- AIOUSB_ADC_SetCalTable
 - AIOUSB_Core.c, [330](#)
 - AIOUSB_Core.h, [334](#)
- AIOUSB_ArrayCountsToVolts
 - AIOUSB_ADC.c, [306](#)
- AIOUSB_ArrayVoltsToCounts
 - AIOUSB_ADC.c, [306](#)
- AIOUSB_BOOL
 - AIOTypes.h, [293](#)
- AIOUSB_BOOL_VAL
 - AIOTypes.h, [294](#)
- AIOUSB_CTR.c
 - CTR_8254Load, [337](#)
 - CTR_8254Mode, [337](#)
 - CTR_8254ModeLoad, [337](#)
 - CTR_8254Read, [337](#)
 - CTR_8254ReadAll, [337](#)
 - CTR_8254ReadLatched, [337](#)
 - CTR_8254ReadModeLoad, [337](#)
 - CTR_8254ReadStatus, [337](#)
 - CTR_8254SelectGate, [337](#)
 - CTR_CalculateCountersForClock, [337](#)
 - CTR_StartOutputFreq, [337](#)
- AIOUSB_CTR.h
 - CTR_8254Load, [338](#)
 - CTR_8254Mode, [338](#)
 - CTR_8254ModeLoad, [338](#)
 - CTR_8254Read, [338](#)
 - CTR_8254ReadAll, [338](#)
 - CTR_8254ReadLatched, [338](#)
 - CTR_8254ReadModeLoad, [338](#)
 - CTR_8254ReadStatus, [338](#)
 - CTR_8254SelectGate, [338](#)
 - CTR_CalculateCountersForClock, [338](#)
 - CTR_StartOutputFreq, [338](#)
- AIOUSB_Cleanup
 - AIODeviceTable.c, [266](#)
- AIOUSB_ClearFIFO
 - AIOUSB_Core.c, [330](#)
 - AIOUSB_Core.h, [334](#)
- AIOUSB_Copy_Config_Block
 - AIOUSB_ADC.c, [314](#)
- AIOUSB_Core.c
 - AIOUSB_ClearFIFO, [330](#)
 - AIOUSB_GetCommTimeout, [330](#)
 - AIOUSB_GetConfigBlock, [329](#)
 - AIOUSB_GetDevice, [329](#)
 - AIOUSB_GetMiscClock, [330](#)
 - AIOUSB_GetStreamingBlockSize, [329](#)
 - AIOUSB_GetVersion, [330](#)
 - AIOUSB_GetVersionDate, [330](#)
 - AIOUSB_InitConfigBlock, [330](#)
 - AIOUSB_Lock, [329](#)
 - AIOUSB_ResetChip, [329](#)
 - AIOUSB_SetCommTimeout, [330](#)
 - AIOUSB_SetMiscClock, [330](#)
 - AIOUSB_SetStreamingBlockSize, [329](#)
 - AIOUSB_UnLock, [329](#)
 - AIOUSB_Validate, [329](#)
 - AIOUSB_Validate_Device, [330](#)
 - AIOUSB_Validate_Lock, [329](#)
 - aio_errno, [330](#)
 - DeviceTableAtIndex, [329](#)
 - DeviceTableAtIndex_Lock, [329](#)
 - GenericVendorRead, [330](#)
 - GenericVendorWrite, [330](#)
 - productIDNameTable, [330](#)
 - ResolveDeviceIndex, [329](#)
- AIOUSB_Core.h
 - ADC_ResetDevice, [333](#)
 - AIOError, [333](#)
 - AIOOption, [333](#)
 - AIOUSB_ClearFIFO, [334](#)
 - AIOUSB_GetCommTimeout, [334](#)
 - AIOUSB_GetConfigBlock, [333](#)
 - AIOUSB_GetDevice, [333](#)
 - AIOUSB_GetDeviceSerialNumber, [333](#)
 - AIOUSB_GetMiscClock, [333](#)
 - AIOUSB_GetStreamingBlockSize, [334](#)
 - AIOUSB_GetVersion, [334](#)
 - AIOUSB_GetVersionDate, [334](#)
 - AIOUSB_InitConfigBlock, [334](#)
 - AIOUSB_InitTest, [333](#)
 - AIOUSB_Lock, [333](#)
 - AIOUSB_ResetChip, [333](#)
 - AIOUSB_SetCommTimeout, [333](#)
 - AIOUSB_SetMiscClock, [333](#)
 - AIOUSB_UnLock, [333](#)
 - AIOUSB_Validate, [333](#)
 - AIOUSB_Validate_Device, [335](#)
 - AIOUSB_Validate_Lock, [333](#)
 - AIOUSB_VoltsToCounts, [334](#)
 - adRanges, [335](#)
 - aio_errno, [335](#)
 - aiousbInit, [335](#)
 - DeviceTableAtIndex, [333](#)
 - DeviceTableAtIndex_Lock, [333](#)
 - GenericVendorRead, [335](#)
 - GenericVendorWrite, [335](#)
 - PRIVATE, [333](#)
- AIOUSB_DAC.c
 - DACDirect, [341](#)
 - DACMultiDirect, [341](#)
 - DACOutputClose, [342](#)
 - DACOutputCloseNoEnd, [342](#)
 - DACOutputFrame, [342](#)
 - DACOutputFrameRaw, [342](#)
 - DACOutputOpen, [342](#)
 - DACOutputSetCount, [342](#)
 - DACOutputSetInterlock, [342](#)
 - DACOutputStart, [342](#)
 - DACSetBoardRange, [342](#)
- AIOUSB_DAC.h
 - DACDirect, [342](#)
 - DACMultiDirect, [342](#)
 - DACOutputClose, [343](#)
 - DACOutputCloseNoEnd, [343](#)
 - DACOutputFrame, [343](#)
 - DACOutputFrameRaw, [343](#)
 - DACOutputOpen, [343](#)
 - DACOutputSetCount, [343](#)
 - DACOutputSetInterlock, [343](#)
 - DACOutputStart, [343](#)
 - DACSetBoardRange, [343](#)
- AIOUSB_DEBUG
 - AIOUSB_Log.h, [349](#)
- AIOUSB_DEVEL
 - AIOUSB_Log.h, [349](#)
- AIOUSB_DIO.c
 - aiousb_h tons, [344](#)
 - DIO_ConfigurationQuery, [345](#)
 - DIO_Configure, [345](#)
 - DIO_ConfigureEx, [345](#)
 - DIO_ConfigureWithDIOBuf, [344](#)
 - DIO_Read1, [345](#)
 - DIO_Read8, [345](#)
 - DIO_ReadAll, [345](#)
 - DIO_ReadAllToCharStr, [345](#)
 - DIO_ReadIntoDIOBuf, [345](#)

- DIO_StreamClose, [345](#)
- DIO_StreamFrame, [345](#)
- DIO_StreamOpen, [345](#)
- DIO_StreamSetClocks, [345](#)
- DIO_Write1, [345](#)
- DIO_Write8, [345](#)
- DIO_WriteAll, [345](#)
- pow_of_minsize, [345](#)
- AIOTUSB_DIO.h
 - DIO_ConfigurationQuery, [347](#)
 - DIO_Configure, [347](#)
 - DIO_ConfigureEx, [347](#)
 - DIO_ConfigureWithDIOBuf, [347](#)
 - DIO_Read1, [347](#)
 - DIO_Read8, [347](#)
 - DIO_ReadAll, [347](#)
 - DIO_ReadAllToCharStr, [347](#)
 - DIO_ReadIntoDIOBuf, [347](#)
 - DIO_StreamClose, [347](#)
 - DIO_StreamFrame, [347](#)
 - DIO_StreamOpen, [347](#)
 - DIO_StreamSetClocks, [347](#)
 - DIO_Write1, [347](#)
 - DIO_Write8, [347](#)
 - DIO_WriteAll, [347](#)
- AIOTUSB_ERROR
 - AIOTUSB_Log.h, [349](#)
- AIOTUSB_ERROR_VALUE
 - AIOTypes.h, [292](#)
- AIOTUSB_EnsureOpen
 - AIODDeviceTable.c, [267](#)
 - AIODDeviceTable.h, [271](#)
- AIOTUSB_Exit
 - AIODDeviceTable.c, [268](#)
 - AIODDeviceTable.h, [271](#)
- AIOTUSB_FATAL
 - AIOTUSB_Log.h, [349](#)
- AIOTUSB_FindDeviceIndicesByGroup
 - AIOTUSB_Properties.c, [351](#)
 - AIOTUSB_Properties.h, [353](#)
- AIOTUSB_FindDevices
 - AIOTUSB_Properties.c, [351](#)
 - AIOTUSB_Properties.h, [353](#)
- AIOTUSB_FindDevicesByGroup
 - AIOTUSB_Properties.c, [351](#)
 - AIOTUSB_Properties.h, [353](#)
- AIOTUSB_GetAllDevices
 - AIODDeviceTable.c, [268](#)
 - AIODDeviceTable.h, [272](#)
- AIOTUSB_GetCalMode
 - AIOTUSB_ADC.c, [316](#)
 - AIOTUSB_ADC.h, [326](#)
- AIOTUSB_GetCommTimeout
 - AIOTUSB_Core.c, [330](#)
 - AIOTUSB_Core.h, [334](#)
- AIOTUSB_GetConfigBlock
 - AIOTUSB_Core.c, [329](#)
 - AIOTUSB_Core.h, [333](#)
- AIOTUSB_GetDevice
 - AIOTUSB_Core.c, [329](#)
 - AIOTUSB_Core.h, [333](#)
- AIOTUSB_GetDeviceByProductID
 - AIOTUSB_Properties.c, [351](#)
 - AIOTUSB_Properties.h, [352](#)
- AIOTUSB_GetDeviceProperties
 - AIOTUSB_Properties.c, [351](#)
 - AIOTUSB_Properties.h, [353](#)
- AIOTUSB_GetDeviceSerialNumber
 - AIOTUSB_Core.h, [333](#)
 - AIOTUSB_Properties.c, [351](#)
 - AIOTUSB_Properties.h, [353](#)
- AIOTUSB_GetEndChannel
 - AIOTUSB_ADC.c, [316](#)
 - AIOTUSB_ADC.h, [326](#)
- AIOTUSB_GetGainCode
 - AIOTUSB_ADC.c, [314](#)
 - AIOTUSB_ADC.h, [324](#)
- AIOTUSB_GetMiscClock
 - AIOTUSB_Core.c, [330](#)
 - AIOTUSB_Core.h, [333](#)
- AIOTUSB_GetOversample
 - AIOTUSB_ADC.c, [316](#)
 - AIOTUSB_ADC.h, [326](#)
- AIOTUSB_GetRegister
 - AIOTUSB_ADC.c, [314](#)
 - AIOTUSB_ADC.h, [327](#)
- AIOTUSB_GetResultCodeAsString
 - AIOTUSB_Core.h, [334](#)
 - AIOTUSB_Properties.c, [351](#)
 - AIOTUSB_Properties.h, [353](#)
- AIOTUSB_GetScan
 - AIOTUSB_ADC.c, [305](#)
 - AIOTUSB_ADC.h, [320](#)
- AIOTUSB_GetStartChannel
 - AIOTUSB_ADC.c, [316](#)
 - AIOTUSB_ADC.h, [326](#)
- AIOTUSB_GetStreamingBlockSize
 - AIOTUSB_Core.c, [329](#)
 - AIOTUSB_Core.h, [334](#)
- AIOTUSB_GetTriggerMode
 - AIOTUSB_ADC.c, [316](#)
 - AIOTUSB_ADC.h, [326](#)
- AIOTUSB_GetVersion
 - AIOTUSB_Core.c, [330](#)
 - AIOTUSB_Core.h, [334](#)
- AIOTUSB_GetVersionDate
 - AIOTUSB_Core.c, [330](#)
 - AIOTUSB_Core.h, [334](#)
- AIOTUSB_INFO
 - AIOTUSB_Log.h, [349](#)
- AIOTUSB_Init
 - AIODDeviceTable.c, [268](#)
 - AIODDeviceTable.h, [271](#)
- AIOTUSB_InitConfigBlock
 - AIOTUSB_Core.c, [330](#)
 - AIOTUSB_Core.h, [334](#)
- AIOTUSB_InitTest
 - AIODDeviceTable.c, [266](#)
 - AIOTUSB_Core.h, [333](#)
- AIOTUSB_IsDifferentialMode
 - AIOTUSB_ADC.c, [316](#)
 - AIOTUSB_ADC.h, [325](#)
- AIOTUSB_IsDiscardFirstSample
 - AIOTUSB_ADC.c, [313](#)
 - AIOTUSB_ADC.h, [326](#)
- AIOTUSB_IsInit
 - AIODDeviceTable.c, [266](#)
 - AIODDeviceTable.h, [271](#)
- AIOTUSB_LOG
 - AIOTUSB_Log.h, [349](#)
- AIOTUSB_ListDevices
 - AIOTUSB_Properties.c, [352](#)
 - AIOTUSB_Properties.h, [353](#)
- AIOTUSB_Lock
 - AIOTUSB_Core.c, [329](#)
 - AIOTUSB_Core.h, [333](#)
- AIOTUSB_Log.c
 - cont_thread, [348](#)
 - message_lock, [348](#)
 - outfile, [348](#)
- AIOTUSB_Log.h
 - AIOTUSB_DEBUG, [349](#)
 - AIOTUSB_DEVEL, [349](#)
 - AIOTUSB_ERROR, [349](#)

- AIOUSB_FATAL, [349](#)
 - AIOUSB_INFO, [349](#)
 - AIOUSB_LOG, [349](#)
 - AIOUSB_WARN, [349](#)
 - CYAN, [349](#)
 - cont_thread, [349](#)
 - GREEN, [349](#)
 - LOG_LEVEL, [349](#)
 - MAGENTA, [349](#)
 - message_lock, [350](#)
 - outfile, [350](#)
 - RED, [349](#)
- AIOUSB_Properties.c
 - AIOUSB_FindDevices, [351](#)
 - AIOUSB_FindDevicesByGroup, [351](#)
 - AIOUSB_GetDeviceProperties, [351](#)
 - AIOUSB_GetDeviceSerialNumber, [351](#)
 - AIOUSB_ListDevices, [352](#)
 - AIOUSB_ShowDevices, [351](#)
 - GetDeviceBySerialNumber, [351](#)
 - GetDeviceSerialNumber, [351](#)
 - non_usb_supported_device, [351](#)
- AIOUSB_Properties.h
 - AIODisplayType, [352](#)
 - AIOUSB_FindDevices, [353](#)
 - AIOUSB_FindDevicesByGroup, [353](#)
 - AIOUSB_GetDeviceProperties, [353](#)
 - AIOUSB_GetDeviceSerialNumber, [353](#)
 - AIOUSB_ListDevices, [353](#)
 - AIOUSB_ShowDevices, [353](#)
 - GetDeviceBySerialNumber, [353](#)
 - GetDeviceSerialNumber, [353](#)
- AIOUSB_Reset
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [271](#)
- AIOUSB_ResetChip
 - AIOUSB_Core.c, [329](#)
 - AIOUSB_Core.h, [333](#)
- AIOUSB_SetAllGainCodeAndDiffMode
 - AIOUSB_ADC.c, [314](#)
 - AIOUSB_ADC.h, [324](#)
- AIOUSB_SetCalMode
 - AIOUSB_ADC.c, [316](#)
 - AIOUSB_ADC.h, [326](#)
- AIOUSB_SetCommTimeout
 - AIOUSB_Core.c, [330](#)
 - AIOUSB_Core.h, [333](#)
- AIOUSB_SetDifferentialMode
 - AIOUSB_ADC.c, [316](#)
 - AIOUSB_ADC.h, [326](#)
- AIOUSB_SetDiscardFirstSample
 - AIOUSB_ADC.c, [313](#)
 - AIOUSB_ADC.h, [326](#)
- AIOUSB_SetGainCode
 - AIOUSB_ADC.c, [315](#)
 - AIOUSB_ADC.h, [324](#)
- AIOUSB_SetInit
 - AIODeviceTable.c, [266](#)
- AIOUSB_SetMiscClock
 - AIOUSB_Core.c, [330](#)
 - AIOUSB_Core.h, [333](#)
- AIOUSB_SetOversample
 - AIOUSB_ADC.c, [316](#)
 - AIOUSB_ADC.h, [326](#)
- AIOUSB_SetRangeSingle
 - AIOUSB_ADC.c, [314](#)
- AIOUSB_SetRegister
 - AIOUSB_ADC.c, [314](#)
 - AIOUSB_ADC.h, [327](#)
- AIOUSB_SetScanRange
 - AIOUSB_ADC.c, [316](#)
 - AIOUSB_ADC.h, [326](#)
- AIOUSB_SetStreamingBlockSize
 - AIOUSB_ADC.h, [326](#)
 - AIOUSB_Core.c, [329](#)
- AIOUSB_SetTriggerMode
 - AIOUSB_ADC.c, [316](#)
 - AIOUSB_ADC.h, [326](#)
- AIOUSB_ShowDevices
 - AIOUSB_Properties.c, [351](#)
 - AIOUSB_Properties.h, [353](#)
- AIOUSB_USB.h
 - usb_bulk_xfer, [354](#)
 - usb_close, [354](#)
 - usb_control_xfer, [354](#)
 - usb_free_devices, [354](#)
 - usb_get_devices, [354](#)
 - usb_open, [354](#)
- AIOUSB_UnLock
 - AIOUSB_Core.c, [329](#)
 - AIOUSB_Core.h, [333](#)
- AIOUSB_Validate
 - AIOUSB_Core.c, [329](#)
 - AIOUSB_Core.h, [333](#)
- AIOUSB_Validate_Device
 - AIOUSB_Core.c, [330](#)
 - AIOUSB_Core.h, [335](#)
- AIOUSB_Validate_Lock
 - AIOUSB_Core.c, [329](#)
 - AIOUSB_Core.h, [333](#)
- AIOUSB_VoltsToCounts
 - AIOUSB_ADC.c, [306](#)
 - AIOUSB_Core.h, [334](#)
- AIOUSB_WARN
 - AIOUSB_Log.h, [349](#)
- AIOUSB_WDG.c
 - DeleteWDGConfig, [354](#)
 - doSomething, [354](#)
 - NewWDGConfig, [354](#)
 - WDG_GetStatus, [355](#)
 - WDG_Pet, [355](#)
 - WDG_SetConfig, [355](#)
- AIOUSB_WDG.h
 - DeleteWDGConfig, [356](#)
 - NewWDGConfig, [356](#)
 - WDG_GetStatus, [356](#)
 - WDG_Pet, [356](#)
 - WDG_SetConfig, [356](#)
 - WDGVals, [356](#)
- AIOUSB_WorkFn
 - AIOContinuousBuffer.h, [249](#)
- AIOUSBDevice, [82](#)
 - ADBuf, [85](#)
 - ADBuf_size, [85](#)
 - ADCChannels, [84](#)
 - ADCChannelsPerGroup, [84](#)
 - ADCMUXChannels, [84](#)
 - ADCConfigBlock.h, [212](#)
 - bADCStream, [84](#)
 - bClearFIFO, [84](#)
 - bDACAborting, [84](#)
 - bDACBoardRange, [84](#)
 - bDACChannelCal, [84](#)
 - bDACClosing, [84](#)
 - bDACDIOClock, [84](#)
 - bDACDIOSStream, [84](#)
 - bDACOpen, [84](#)
 - bDACSlowWaveStream, [84](#)
 - bDACStarted, [84](#)
 - bDACStream, [84](#)
 - bDIODebounce, [84](#)
 - bDIOOpen, [84](#)
 - bDIORead, [84](#)
 - bDIOSPI, [84](#)

- bDIStream, [84](#)
- bDeviceWasHere, [85](#)
- bGateSelectable, [83](#)
- bGetName, [83](#)
- bOpen, [83](#)
- bSetCustomClocks, [84](#)
- cachedConfigBlock, [85](#)
- cachedName, [85](#)
- cachedSerialNumber, [85](#)
- commTimeout, [83](#)
- ConfigBytes, [84](#)
- Counters, [83](#)
- DACData, [84](#)
- DACsUsed, [84](#)
- DIOBytes, [83](#)
- DIOConfigBits, [83](#)
- deviceIndex, [83](#)
- discardFirstSample, [83](#)
- FastITBakConfig, [85](#)
- FastITConfig, [85](#)
- FastITConfig_size, [85](#)
- FlashSectors, [84](#)
- hDACDataMutex, [84](#)
- hDACDataSem, [84](#)
- ImmADCs, [84](#)
- ImmDACs, [84](#)
- isInit, [83](#)
- LastDIOData, [85](#)
- miscClockHz, [83](#)
- PID, [83](#)
- PendingDACData, [84](#)
- ProductID, [83](#)
- RangeShift, [84](#)
- RootClock, [83](#)
- StreamingBlockSize, [84](#)
- testing, [85](#)
- Tristates, [83](#)
- usb_device, [83](#)
- valid, [85](#)
- WDGBytes, [84](#)
- workerBusy, [85](#)
- workerResult, [85](#)
- workerStatus, [85](#)
- AIOTUSBDevice.c
 - AIOTUSBDeviceCopyADCCConfigBlock, [357](#)
 - AIOTUSBDeviceGetADCCConfigBlock, [357](#)
 - AIOTUSBDeviceGetDiscardFirstSample, [358](#)
 - AIOTUSBDeviceGetStreamingBlockSize, [357](#)
 - AIOTUSBDeviceGetTesting, [357](#)
 - AIOTUSBDeviceGetTimeout, [357](#)
 - AIOTUSBDeviceGetUSBHandle, [357](#)
 - AIOTUSBDeviceInitializeWithProductID, [357](#)
 - AIOTUSBDeviceSetADCCConfigBlock, [357](#)
 - AIOTUSBDeviceSetDiscardFirstSample, [358](#)
 - AIOTUSBDeviceSetTesting, [357](#)
 - AIOTUSBDeviceSetTimeout, [357](#)
 - AIOTUSBDeviceSetUSBHandle, [357](#)
 - AIOTUSBDeviceSize, [357](#)
 - AIOTUSBDeviceToJSON, [357](#)
 - AIOTUSBDeviceWriteADCCConfig, [357](#)
 - DeleteAIOTUSBDevice, [357](#)
 - NewAIOTUSBDevice, [357](#)
 - NewAIOTUSBDeviceFromJSON, [357](#)
- AIOTUSBDevice.h
 - AIOTUSBDeviceCopyADCCConfigBlock, [359](#)
 - AIOTUSBDeviceGetADCCConfigBlock, [359](#)
 - AIOTUSBDeviceGetDiscardFirstSample, [359](#)
 - AIOTUSBDeviceGetStreamingBlockSize, [359](#)
 - AIOTUSBDeviceGetTesting, [359](#)
 - AIOTUSBDeviceGetTimeout, [359](#)
 - AIOTUSBDeviceGetUSBHandle, [358](#)
 - AIOTUSBDeviceInitializeWithProductID, [358](#)
 - AIOTUSBDeviceSetADCCConfigBlock, [359](#)
 - AIOTUSBDeviceSetDiscardFirstSample, [359](#)
 - AIOTUSBDeviceSetTesting, [359](#)
 - AIOTUSBDeviceSetTimeout, [359](#)
 - AIOTUSBDeviceSetUSBHandle, [358](#)
 - AIOTUSBDeviceSize, [359](#)
 - AIOTUSBDeviceToJSON, [358](#)
 - AIOTUSBDeviceWriteADCCConfig, [359](#)
- AIOTUSBDeviceSetADCCConfigBlock, [359](#)
- AIOTUSBDeviceSetDiscardFirstSample, [359](#)
- AIOTUSBDeviceSetTesting, [359](#)
- AIOTUSBDeviceSetTimeout, [359](#)
- AIOTUSBDeviceSetUSBHandle, [358](#)
- AIOTUSBDeviceSize, [359](#)
- AIOTUSBDeviceToJSON, [358](#)
- AIOTUSBDeviceWriteADCCConfig, [359](#)
- DeviceDescriptor, [358](#)
- NewAIOTUSBDeviceFromJSON, [358](#)
- AIOTUSBDeviceCopyADCCConfigBlock
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceGetADCCConfigBlock
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceGetDiscardFirstSample
 - AIOTUSBDevice.c, [358](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceGetStreamingBlockSize
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceGetTesting
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceGetTimeout
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceGetUSBHandle
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [358](#)
- AIOTUSBDeviceGetUSBHandleFromDeviceIndex
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [358](#)
- AIOTUSBDeviceInitializeWithProductID
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [358](#)
- AIOTUSBDeviceSetADCCConfigBlock
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceSetDiscardFirstSample
 - AIOTUSBDevice.c, [358](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceSetTesting
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceSetTimeout
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceSetUSBHandle
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [358](#)
- AIOTUSBDeviceSize
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBDeviceToJSON
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [358](#)
- AIOTUSBDeviceWriteADCCConfig
 - AIOTUSBDevice.c, [357](#)
 - AIOTUSBDevice.h, [359](#)
- AIOTUSBGetError
 - AIOTDeviceTable.c, [268](#)
 - AIOTDeviceTable.h, [272](#)
- AIOWDGConfig, [85](#)
 - bufsize, [86](#)
 - L, [86](#)
 - timeout, [86](#)
 - wdgbuf, [86](#)
- ALERT_LEVEL
 - TestCaseSetup.h, [383](#)
- AO16_AnalogOutputSubsystem, [111](#)

- AIOUSB::AO16_AnalogOutputSubsystem, [113](#)
 - AIOUSB::AO16_OutputRange, [117](#)
- AO16_OutputRange, [116](#)
- AIOUSB::AO16_OutputRange, [116](#)
- AUR_CBUF_EXIT
 - AIOTypes.h, [291](#)
- AUR_CBUF_SETUP
 - AIOTypes.h, [291](#)
- active_signals
 - AIOChannelMask, [74](#)
- actual_size
 - AIOArgument, [72](#)
- ad_gain_pairs, [63](#)
 - gain, [63](#)
 - name, [63](#)
- adRanges
 - AIOUSB_ADC.c, [317](#)
 - AIOUSB_Core.h, [335](#)
- adc
 - AIOUSB::USB_AI16_Family, [159](#)
 - AIOUSB::USB_AIO16_Family, [161](#)
- adc_get_bulk_data
 - AIOUSB_ADC.c, [305](#)
 - mock_aiocontbuf_get_data_arduino.c, [372](#)
- adcblock_valid_channel_settings
 - AIOUSB_ADC.c, [308](#)
- adcblock_valid_size
 - AIOUSB_ADC.c, [308](#)
- adcblock_valid_trigger_settings
 - AIOUSB_ADC.c, [308](#)
- adconfig_json
 - AIOCommandLineOptions, [77](#)
 - opts, [150](#)
- add_devices_fn
 - mock_usb_xfers.c, [376](#)
- AddAllACCESUSBDevices
 - mock_usb_xfers.c, [376](#)
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- AddDevice
 - USBDevice.c, [377](#)
- aio_channel_obj
 - AIOChannelMask.h, [220](#)
- aio_channel_range, [70](#)
 - end, [70](#)
 - gain, [70](#)
 - start, [70](#)
- aio_channel_range_error
 - AIOChannelRange.c, [223](#)
 - AIOChannelRange.h, [224](#)
- aio_counts_converter, [71](#)
 - buf, [71](#)
 - channel_count, [71](#)
 - continue_conversion, [71](#)
 - Convert, [71](#)
 - ConvertFifo, [71](#)
 - converted_count, [71](#)
 - discardFirstSample, [71](#)
 - gain_ranges, [71](#)
 - num_channels, [71](#)
 - num_oversamples, [71](#)
 - num_scans, [71](#)
 - os_count, [71](#)
 - scan_count, [71](#)
 - sum, [71](#)
 - unit_size, [71](#)
- aio_either_val, [71](#)
 - number, [72](#)
 - object, [72](#)
- aio_errno
 - AIOUSB_Core.c, [330](#)
 - AIOUSB_Core.h, [335](#)
- aio_list_devices
 - aiocommon.c, [381](#)
 - aiocommon.h, [381](#)
- aio_override_adconfig_settings
 - aiocommon.c, [381](#)
 - aiocommon.h, [381](#)
- aio_override_aiobuf_settings
 - aiocommon.c, [381](#)
 - aiocommon.h, [381](#)
- aio_ret_value, [72](#)
 - errmsg, [72](#)
 - left, [72](#)
 - right, [72](#)
 - size, [72](#)
 - type, [72](#)
- aio_supply_default_command_line_settings
 - aiocommon.c, [381](#)
 - aiocommon.h, [381](#)
- aiobuf_iterator, [73](#)
 - buf, [73](#)
 - loc, [73](#)
 - next, [73](#)
- aiobuf_json
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- aiocommon.c
 - AIO_OPTIONS, [381](#)
 - aio_list_devices, [381](#)
 - aio_override_adconfig_settings, [381](#)
 - aio_override_aiobuf_settings, [381](#)
 - aio_supply_default_command_line_settings, [381](#)
 - CNTS, [380](#)
 - DUMP, [380](#)
 - get_channel_range, [380](#)
 - JCONF, [380](#)
 - print_aio_usage, [380](#)
 - process_aio_cmd_line, [380](#)
 - REPEAT, [380](#)
- aiocommon.h
 - AIO_OPTIONS, [381](#)
 - aio_list_devices, [381](#)
 - aio_override_adconfig_settings, [381](#)
 - aio_override_aiobuf_settings, [381](#)
 - aio_supply_default_command_line_settings, [381](#)
 - get_channel_range, [381](#)
 - print_aio_usage, [381](#)
 - process_aio_cmd_line, [381](#)
- aiocontbuf_get_bulk_data
 - mock_aiocontbuf_get_data.c, [371](#)
 - mock_aiocontbuf_get_data_arduino.c, [372](#)
- aioeither_value_double
 - AIOEither.h, [275](#)
- aioeither_value_double_t
 - AIOEither.h, [275](#)
- aioeither_value_int
 - AIOEither.h, [274](#)
- aioeither_value_int32_t
 - AIOEither.h, [274](#)
- aioeither_value_longdouble_t
 - AIOEither.h, [275](#)
- aioeither_value_obj
 - AIOEither.h, [275](#)
- aioeither_value_string
 - AIOEither.h, [275](#)
- aioeither_value_uint16_t
 - AIOEither.h, [275](#)
- aioeither_value_uint32_t
 - AIOEither.h, [274](#)
- aioeither_value_uint8_t
 - AIOEither.h, [275](#)
- aioeither_value_unsigned
 - AIOEither.h, [275](#)

- aioeither_vlaue_int16_t
 - AIOEither.h, [275](#)
- aioerror, [80](#)
 - error_message, [80](#)
 - retval, [80](#)
- AIOUSB, [7](#)
- aiusb_h tons
 - AIOUSB_DIO.c, [344](#)
- aiusb_libusb_args, [81](#)
 - dev, [82](#)
 - deviceDesc, [82](#)
 - handle, [82](#)
- aiusbInit
 - AIODeviceTable.c, [268](#)
 - AIOUSB_Core.h, [335](#)
- aiusboption, [85](#)
- altset
 - USBDevice, [179](#)
- AnalogIORange, [106](#)
 - AIOUSB::AnalogIORange, [107](#)
- AnalogInputSubsystem, [86](#)
 - AIOUSB::AI16_DataPoint, [66](#)
 - AIOUSB::AI16_DataSet, [69](#)
 - AIOUSB::AI16_InputRange, [70](#)
 - AIOUSB::AnalogInputSubsystem, [89](#)
 - AIOUSB::USBDeviceBase, [183](#)
- analogInputSubsystem
 - AIOUSB::USB_AI16_Family, [159](#)
 - AIOUSB::USB_AIO16_Family, [162](#)
- AnalogOutputSubsystem, [110](#)
 - AIOUSB::AnalogOutputSubsystem, [110](#)
- analogOutputSubsystem
 - AIOUSB::USB_AO16_Family, [164](#)
 - AIOUSB::USB_DA12_8A_Family, [169](#)
 - AIOUSB::USB_DA12_8E_Family, [171](#)
- analogOutputSubsytem
 - AIOUSB::USB_AIO16_Family, [162](#)
- autoCalFeature
 - AIOUSB::AnalogInputSubsystem, [106](#)
- autoConfig
 - AIOUSB::AnalogInputSubsystem, [106](#)
- BASIC
 - AIOUSB_Properties.h, [352](#)
- BEGIN
 - AIOChannelRange.c, [222](#)
- BITS_PER_BYTE
 - AIOTypes.h, [301](#)
- BACKTRACE_DEBUG
 - AIOUSB_Core.c, [329](#)
- bADCStream
 - AIOUSBDevice, [84](#)
- bClearFIFO
 - AIOUSBDevice, [84](#)
- bDACAborting
 - AIOUSBDevice, [84](#)
- bDACBoardRange
 - AIOUSBDevice, [84](#)
- bDACChannelCal
 - AIOUSBDevice, [84](#)
- bDACClosing
 - AIOUSBDevice, [84](#)
- bDACDIOClock
 - AIOUSBDevice, [84](#)
- bDACDIOStream
 - AIOUSBDevice, [84](#)
- bDACOpen
 - AIOUSBDevice, [84](#)
- bDACSlowWaveStream
 - AIOUSBDevice, [84](#)
- bDACStarted
 - AIOUSBDevice, [84](#)
- bDACStream
 - AIOUSBDevice, [84](#)
- bDIODebounce
 - AIOUSBDevice, [84](#)
- bDIOOpen
 - AIOUSBDevice, [84](#)
- bDIORead
 - AIOUSBDevice, [84](#)
- bDIOSPI
 - AIOUSBDevice, [84](#)
- bDIOStream
 - AIOUSBDevice, [84](#)
- bDeviceWasHere
 - AIOUSBDevice, [85](#)
- bGateSelectable
 - AIOUSBDevice, [83](#)
- bGetName
 - AIOUSBDevice, [83](#)
- BIT_LENGTH
 - AIOChannelMask.h, [220](#)
- BITS_PER_BYTE
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [405](#)
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [407](#)
- bOpen
 - AIOUSBDevice, [83](#)
- bSetCustomClocks
 - AIOUSBDevice, [84](#)
- base_size
 - AIOContinuousBuf, [78](#)
- BaseSizeRange
 - AIOContinuousBuffer.c, [246](#)
- bitsToBytes
 - AIOUSB::DigitalIOSubsystem, [137](#)
- block_size
 - AIOCommandLineOptions, [76](#)
 - AIOContinuousBuf, [78](#)
 - opts, [150](#)
 - TestCaseSetup, [156](#)
- BoolArray, [117](#)
 - AIOUSB::BoolArray, [117](#)
- buf
 - aio_counts_converter, [71](#)
 - aiobuf_iterator, [73](#)
- BufSize
 - BulkAcquireWorkerParams, [117](#)
- buffer
 - AIOContinuousBuf, [78](#)
 - DIOBuf, [141](#)
- buffer_size
 - AIOCommandLineOptions, [76](#)
 - config_options, [119](#)
 - opts, [150](#)
- bufsize
 - AIOWDGConfig, [86](#)
- bulk_acquire_sample.c
 - main, [384](#)
 - print_usage, [384](#)
 - process_cmd_line, [384](#)
- BulkAcquireWorkerParams, [117](#)
 - BufSize, [117](#)
 - DeviceIndex, [117](#)
 - pBuf, [117](#)
- burst_test.c
 - find_ai_board, [385](#)
- bytes_processed
 - AIOContinuousBuf, [78](#)
- bytesToBits
 - AIOUSB::DigitalIOSubsystem, [137](#)
- CAL_TABLE_WORDS

- AIOTypes.h, [301](#)
- CHANNEL_OPT
 - AIOCommandLine.h, [229](#)
- CLEAR_FIFO_METHOD_AUTO
 - AIOTypes.h, [297](#)
- CLEAR_FIFO_METHOD_IMMEDIATE
 - AIOTypes.h, [297](#)
- CLEAR_FIFO_METHOD_IMMEDIATE_AND_ABORT
 - AIOTypes.h, [297](#)
- CLEAR_FIFO_METHOD_NOW
 - AIOTypes.h, [297](#)
- CLEAR_FIFO_METHOD_WAIT
 - AIOTypes.h, [297](#)
- COUNT_OPT
 - AIOCommandLine.h, [229](#)
- COUNTER_NUM_MODES
 - AIOTypes.h, [301](#)
- COUNTERS_PER_BLOCK
 - AIOTypes.h, [301](#)
- CYPRESS_DESC_PARAMS
 - AIOTypes.h, [301](#)
- CYPRESS_GET_DESC
 - AIOTypes.h, [300](#)
- CYPRESS_MAX_DESC_SIZE
 - AIOTypes.h, [301](#)
- CAL_CHANNEL
 - TestCaseSetup, [156](#)
- CAL_MODE_GROUND
 - AIOUSB::AnalogInputSubsystem, [104](#)
- CAL_MODE_NORMAL
 - AIOUSB::AnalogInputSubsystem, [104](#)
- CAL_TABLE_WORDS
 - AIOUSB::AnalogInputSubsystem, [105](#)
- CHECK_RESULT
 - read_and_write.c, [402](#)
 - TestCaseSetup.h, [383](#)
- cJSON, [118](#)
 - child, [118](#)
 - cJSON.h, [362](#)
 - next, [118](#)
 - prev, [118](#)
 - string, [118](#)
 - type, [118](#)
 - valuedouble, [118](#)
 - valueint, [118](#)
 - valuelstring, [118](#)
- cJSON.c
 - cJSON_AddItemReferenceToArray, [360](#)
 - cJSON_AddItemReferenceToObject, [360](#)
 - cJSON_AddItemToArray, [360](#)
 - cJSON_AddItemToObject, [360](#)
 - cJSON_AsInteger, [360](#)
 - cJSON_CreateArray, [360](#)
 - cJSON_CreateBool, [360](#)
 - cJSON_CreateDoubleArray, [361](#)
 - cJSON_CreateFalse, [360](#)
 - cJSON_CreateFloatArray, [361](#)
 - cJSON_CreateIntArray, [360](#)
 - cJSON_CreateNull, [360](#)
 - cJSON_CreateNumber, [360](#)
 - cJSON_CreateObject, [360](#)
 - cJSON_CreateString, [360](#)
 - cJSON_CreateStringArray, [361](#)
 - cJSON_CreateTrue, [360](#)
 - cJSON_Delete, [360](#)
 - cJSON_DeleteItemFromArray, [360](#)
 - cJSON_DeleteItemFromObject, [360](#)
 - cJSON_DetachItemFromArray, [360](#)
 - cJSON_DetachItemFromObject, [360](#)
 - cJSON_Duplicate, [361](#)
 - cJSON_GetArrayItem, [360](#)
 - cJSON_GetArraySize, [360](#)
- cJSON_GetErrorPtr, [360](#)
- cJSON_GetObjectItem, [360](#)
- cJSON_InitHooks, [360](#)
- cJSON_Minify, [361](#)
- cJSON_Parse, [360](#)
- cJSON_ParseWithOpts, [360](#)
- cJSON_Print, [360](#)
- cJSON_PrintUnformatted, [360](#)
- cJSON_ReplaceItemInArray, [360](#)
- cJSON_ReplaceItemInObject, [360](#)
- cJSON.h
 - cJSON, [362](#)
 - cJSON_AddBoolToObject, [362](#)
 - cJSON_AddFalseToObject, [362](#)
 - cJSON_AddItemReferenceToArray, [363](#)
 - cJSON_AddItemReferenceToObject, [363](#)
 - cJSON_AddItemToArray, [363](#)
 - cJSON_AddItemToObject, [363](#)
 - cJSON_AddNullToObject, [362](#)
 - cJSON_AddNumberToObject, [362](#)
 - cJSON_AddStringToObject, [362](#)
 - cJSON_AddTrueToObject, [362](#)
 - cJSON_Array, [362](#)
 - cJSON_AsInteger, [363](#)
 - cJSON_CreateArray, [363](#)
 - cJSON_CreateBool, [363](#)
 - cJSON_CreateDoubleArray, [363](#)
 - cJSON_CreateFalse, [363](#)
 - cJSON_CreateFloatArray, [363](#)
 - cJSON_CreateIntArray, [363](#)
 - cJSON_CreateNull, [363](#)
 - cJSON_CreateNumber, [363](#)
 - cJSON_CreateObject, [363](#)
 - cJSON_CreateString, [363](#)
 - cJSON_CreateStringArray, [363](#)
 - cJSON_CreateTrue, [363](#)
 - cJSON_Delete, [362](#)
 - cJSON_DeleteItemFromArray, [363](#)
 - cJSON_DeleteItemFromObject, [363](#)
 - cJSON_DetachItemFromArray, [363](#)
 - cJSON_DetachItemFromObject, [363](#)
 - cJSON_Duplicate, [363](#)
 - cJSON_False, [362](#)
 - cJSON_GetArrayItem, [363](#)
 - cJSON_GetArraySize, [363](#)
 - cJSON_GetErrorPtr, [363](#)
 - cJSON_GetObjectItem, [363](#)
 - cJSON_Hooks, [362](#)
 - cJSON_InitHooks, [362](#)
 - cJSON_IsReference, [362](#)
 - cJSON_Minify, [363](#)
 - cJSON_NULL, [362](#)
 - cJSON_Number, [362](#)
 - cJSON_Object, [362](#)
 - cJSON_Parse, [362](#)
 - cJSON_ParseWithOpts, [363](#)
 - cJSON_Print, [362](#)
 - cJSON_PrintUnformatted, [362](#)
 - cJSON_ReplaceItemInArray, [363](#)
 - cJSON_ReplaceItemInObject, [363](#)
 - cJSON_SetIntValue, [362](#)
 - cJSON_String, [362](#)
 - cJSON_True, [362](#)
- cJSON_AddBoolToObject
 - cJSON.h, [362](#)
- cJSON_AddFalseToObject
 - cJSON.h, [362](#)
- cJSON_AddItemReferenceToArray
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_AddItemReferenceToObject
 - cJSON.c, [360](#)

- cJSON.h, [363](#)
- cJSON_AddItemToArray
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_AddItemToObject
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_AddNullToObject
 - cJSON.h, [362](#)
- cJSON_AddNumberToObject
 - cJSON.h, [362](#)
- cJSON_AddStringToObject
 - cJSON.h, [362](#)
- cJSON_AddTrueToObject
 - cJSON.h, [362](#)
- cJSON_Array
 - cJSON.h, [362](#)
- cJSON_AsInteger
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateArray
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateBool
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateDoubleArray
 - cJSON.c, [361](#)
 - cJSON.h, [363](#)
- cJSON_CreateFalse
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateFloatArray
 - cJSON.c, [361](#)
 - cJSON.h, [363](#)
- cJSON_CreateIntArray
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateNull
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateNumber
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateObject
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateString
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_CreateStringArray
 - cJSON.c, [361](#)
 - cJSON.h, [363](#)
- cJSON_CreateTrue
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_Delete
 - cJSON.c, [360](#)
 - cJSON.h, [362](#)
- cJSON_DeleteItemFromArray
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_DeleteItemFromObject
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_DetachItemFromArray
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_DetachItemFromObject
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_Duplicate
 - cJSON.c, [361](#)
 - cJSON.h, [363](#)
- cJSON_False
 - cJSON.h, [362](#)
- cJSON_GetArrayItem
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_GetArraySize
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_GetErrorPtr
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_GetObjectItem
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_Hooks, [118](#)
 - cJSON.h, [362](#)
 - free_fn, [118](#)
 - malloc_fn, [118](#)
- cJSON_InitHooks
 - cJSON.c, [360](#)
 - cJSON.h, [362](#)
- cJSON_IsReference
 - cJSON.h, [362](#)
- cJSON_Minify
 - cJSON.c, [361](#)
 - cJSON.h, [363](#)
- cJSON_NULL
 - cJSON.h, [362](#)
- cJSON_Number
 - cJSON.h, [362](#)
- cJSON_Object
 - cJSON.h, [362](#)
- cJSON_Parse
 - cJSON.c, [360](#)
 - cJSON.h, [362](#)
- cJSON_ParseWithOpts
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_Print
 - cJSON.c, [360](#)
 - cJSON.h, [362](#)
- cJSON_PrintUnformatted
 - cJSON.c, [360](#)
 - cJSON.h, [362](#)
- cJSON_ReplaceItemInArray
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_ReplaceItemInObject
 - cJSON.c, [360](#)
 - cJSON.h, [363](#)
- cJSON_SetIntValue
 - cJSON.h, [362](#)
- cJSON_String
 - cJSON.h, [362](#)
- cJSON_True
 - cJSON.h, [362](#)
- CNTS
 - AIOCommandLine.h, [229](#)
 - aiocommon.c, [380](#)
- COUNTS
 - AIOTypes.h, [293](#)
- CREATE_ENUM
 - AIOTypes.h, [291](#)
- CStringArray, [126](#)
 - CStringArray.h, [365](#)
- CStringArray.c
 - CStringArrayToString, [364](#)
 - CStringArrayToStringWithDelimiter, [364](#)
 - CopyCStringArray, [364](#)
 - DeleteCStringArray, [364](#)

- NewCStringArray, [364](#)
- NewCStringArrayFromCArgs, [364](#)
- NewCStringArrayWithStrings, [364](#)
- CStringArray.h
 - CStringArray, [365](#)
 - CStringArrayToString, [365](#)
 - CStringArrayToStringWithDelimiter, [365](#)
 - CopyCStringArray, [365](#)
 - DeleteCStringArray, [365](#)
 - NewCStringArray, [365](#)
 - NewCStringArrayFromCArgs, [365](#)
 - NewCStringArrayWithStrings, [365](#)
 - STRING_ARRAY, [365](#)
- CStringArrayToString
 - CStringArray.c, [364](#)
 - CStringArray.h, [365](#)
- CStringArrayToStringWithDelimiter
 - CStringArray.c, [364](#)
 - CStringArray.h, [365](#)
- CTR_8254Load
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254Mode
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254ModeLoad
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254Read
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254ReadAll
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254ReadLatched
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254ReadModeLoad
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254ReadStatus
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_8254SelectGate
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_CalculateCountersForClock
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CTR_StartOutputFreq
 - AIOUSB_CTR.c, [337](#)
 - AIOUSB_CTR.h, [338](#)
- CYAN
 - AIOUSB_Log.h, [349](#)
- cachedConfigBlock
 - AIOUSBDevice, [85](#)
- cachedName
 - AIOUSBDevice, [85](#)
- cachedSerialNumber
 - AIOUSBDevice, [85](#)
- cal_channel
 - opts, [150](#)
- calMode
 - AIOUSB::AI16_DataSet, [69](#)
 - AIOUSB::AnalogInputSubsystem, [106](#)
- calibrate
 - AIOUSB::AnalogInputSubsystem, [98](#)
- calibration
 - AIOCommandLineOptions, [76](#)
 - configuration, [119](#)
 - opts, [150](#)
- calibration_enabled
 - opts, [150](#)
 - TestCaseSetup, [156](#)
- calibration_file
 - configuration, [120](#)
- callback
 - AIOContinuousBuf, [78](#)
- capture_data
 - continuous_mode_callback.c, [386](#)
 - continuous_mode_from_json_config.c, [387](#)
 - dirctest.c, [389](#)
 - test.c, [391](#)
- channel
 - AIOCmd, [75](#)
 - AIOUSB::AI16_DataPoint, [66](#)
 - AIOUSB::OutputVoltagePoint, [151](#)
- channel_count
 - aio_counts_converter, [71](#)
- channel_range, [117](#)
 - end_channel, [118](#)
 - gaincode, [118](#)
 - start_channel, [118](#)
- channelsPerGroup
 - AIOUSB::AnalogInputSubsystem, [106](#)
- child
 - cJSON, [118](#)
- ClearAIODeviceTable
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [271](#)
- ClearDevices
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [270](#)
- clearFIFO
 - AIOUSB::AnalogInputSubsystem, [102](#)
 - AIOUSB::DIOStreamSubsystem, [145](#)
 - AIOUSB::USBDeviceBase, [181](#)
- clock_rate
 - ADCCfgBlock, [64](#)
 - AIOCommandLineOptions, [76](#)
 - config_options, [119](#)
 - opts, [150](#)
- clock_scale
 - opts, [150](#)
- clock_speed
 - opts, [150](#)
 - TestCaseSetup, [156](#)
- clockHz
 - AIOUSB::DIOStreamSubsystem, [145](#)
- close
 - AIOUSB::DIOStreamSubsystem, [144](#)
 - AIOUSB::USBDeviceManager, [188](#)
- CloseAllDevices
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [272](#)
 - mock_aiocontbuf_get_data_arduino.c, [372](#)
- commTimeout
 - AIOUSBDevice, [83](#)
- conf
 - USBDevice, [179](#)
- config
 - AIOArgument, [72](#)
- config_options, [119](#)
 - buffer_size, [119](#)
 - clock_rate, [119](#)
 - framePoints, [119](#)
 - number_channels, [119](#)
 - targetSerialNumber, [119](#)
 - write_clock_rate, [119](#)
- configBlock
 - TestCaseSetup, [156](#)
- configBlockSize
 - AIOUSB::AnalogInputSubsystem, [106](#)
- ConfigBytes

- AIOUSBDevice, [84](#)
- configuration, [119](#)
 - calibration, [119](#)
 - calibration_file, [120](#)
 - configure, [120](#)
 - debug, [120](#)
 - device_index, [119](#)
 - discard_first_sample, [119](#)
 - file_handle, [120](#)
 - file_name, [120](#)
 - number_scans, [119](#)
 - output_file, [120](#)
 - run, [120](#)
 - scan_type, [120](#)
 - timeout, [119](#)
 - type, [119](#)
- ConfigurationType
 - AIOConfiguration.h, [233](#)
- configure
 - AIOUSB::DigitalIOSubsystem, [138](#)
 - configuration, [120](#)
- ConfigureAndBulkAcquire
 - AIOUSB_ADC.c, [314](#)
- cont_thread
 - AIOUSB_Log.c, [348](#)
 - AIOUSB_Log.h, [349](#)
- continue_conversion
 - aio_counts_converter, [71](#)
- continue_running
 - AIOContinuousBuffer.c, [243](#)
- continuous_mode.c
 - fnd, [385](#)
- continuous_mode_callback.c
 - capture_data, [386](#)
 - fnd, [386](#)
 - fp, [386](#)
 - get_channel_range, [386](#)
 - main, [386](#)
 - process_cmd_line, [386](#)
- continuous_mode_from_json_config.c
 - capture_data, [387](#)
 - fnd, [387](#)
 - fp, [387](#)
 - get_channel_range, [387](#)
 - main, [387](#)
 - process_cmd_line, [387](#)
- Convert
 - aio_counts_converter, [71](#)
 - AIOCountsConverter.c, [256](#)
- ConvertCountsToVoltsFunction
 - AIOContinuousBuffer.c, [238](#)
- ConvertFifo
 - aio_counts_converter, [71](#)
- converted_count
 - aio_counts_converter, [71](#)
- CopyCStringArray
 - CStringArray.c, [364](#)
 - CStringArray.h, [365](#)
- CopyUSBDevice
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- Counter, [120](#)
 - AIOUSB::Counter, [121](#)
 - AIOUSB::CounterSubsystem, [126](#)
- counter_control
 - AIOContinuousBuf, [78](#)
- counterIndex
 - AIOUSB::Counter, [123](#)
- CounterList, [123](#)
- CounterSubsystem, [123](#)
 - AIOUSB::Counter, [122](#)
 - AIOUSB::CounterSubsystem, [124](#)
- counterSubsystem
 - AIOUSB::USB_AI16_Family, [159](#)
 - AIOUSB::USB_AIO16_Family, [162](#)
 - AIOUSB::USB_CTR_15_Family, [167](#)
 - AIOUSB::USB_DIO_32_Family, [176](#)
- Counters
 - AIODeviceInfo, [79](#)
 - AIOUSBDevice, [83](#)
 - DeviceProperties, [135](#)
- counters
 - AIOUSB::CounterSubsystem, [126](#)
- counts
 - AIOCommandLineOptions, [76](#)
 - AIOUSB::AI16_DataPoint, [66](#)
 - opts, [150](#)
 - TestCaseSetup, [156](#)
- countsToVolts
 - AIOUSB::AnalogInputSubsystem, [102](#)
 - AIOUSB::AnalogIORange, [109](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [131](#)
- countsbuf
 - AIOContinuousBuf, [78](#)
- CreateSmartBuffer
 - AIOUSB_ADC.c, [312](#)
 - AIOUSB_ADC.h, [327](#)
- cstuff, [11](#)
- ctr
 - AIOUSB::USB_AI16_Family, [159](#)
 - AIOUSB::USB_AIO16_Family, [162](#)
 - AIOUSB::USB_CTR_15_Family, [166](#)
 - AIOUSB::USB_DIO_32_Family, [175](#)
- CustomEEPROMRead
 - AIOUSB_CustomEEPROM.c, [339](#)
 - AIOUSB_CustomEEPROM.h, [340](#)
- customEEPROMRead
 - AIOUSB::USBDeviceBase, [183](#)
- CustomEEPROMWrite
 - AIOUSB_CustomEEPROM.c, [339](#)
 - AIOUSB_CustomEEPROM.h, [340](#)
- customEEPROMWrite
 - AIOUSB::USBDeviceBase, [182](#)
- DAC_RANGE_0_10V
 - AIOTypes.h, [296](#)
- DAC_RANGE_0_5V
 - AIOTypes.h, [296](#)
- DAC_RANGE_10V
 - AIOTypes.h, [296](#)
- DAC_RANGE_5V
 - AIOTypes.h, [296](#)
- DAC_RESET
 - AIOTypes.h, [301](#)
- DACRange_begin
 - AIOTypes.h, [296](#)
- DACRange_end
 - AIOTypes.h, [296](#)
- DEBUG_OPT
 - AIOCommandLine.h, [229](#)
- DA12_AnalogOutputSubsystem, [127](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [128](#)
 - AIOUSB::DA12_OutputRange, [133](#)
- DA12_OutputRange, [132](#)
 - AIOUSB::DA12_OutputRange, [133](#)
- DACChannels
 - DeviceProperties, [135](#)
- DACData
 - AIOUSBDevice, [84](#)
- DACDirect
 - AIOUSB_DAC.c, [341](#)
 - AIOUSB_DAC.h, [342](#)
- DACMultiDirect

- AIOUSB_DAC.c, [341](#)
 - AIOUSB_DAC.h, [342](#)
- DACOutputClose
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACOutputCloseNoEnd
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACOutputFrame
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACOutputFrameRaw
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACOutputOpen
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACOutputSetCount
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACOutputSetInterlock
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACOutputStart
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACRange
 - AIOTypes.h, [296](#)
- DACSetBoardRange
 - AIOUSB_DAC.c, [342](#)
 - AIOUSB_DAC.h, [343](#)
- DACsUsed
 - AIOUSBDevice, [84](#)
- DEBUG
 - TestCaseSetup.h, [383](#)
- DEBUG_LEVEL
 - TestCaseSetup.h, [383](#)
- DEF_CAL_CHANNEL
 - TestCaseSetup.h, [384](#)
- DEF_MAX_CHANNELS
 - TestCaseSetup.h, [383](#)
- DEF_NUM_CHANNELS
 - TestCaseSetup.h, [384](#)
- DEVICES_REQUIRED
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [405](#)
 - write_sample.c, [407](#)
- DIO_ConfigurationQuery
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_Configure
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_ConfigureEx
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_ConfigureWithDIOBuf
 - AIOUSB_DIO.c, [344](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_Read1
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_Read8
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_ReadAll
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_ReadAllToCharStr
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_ReadAllToDIOBuf
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_ReadIntoDIOBuf
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_StreamClose
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_StreamFrame
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_StreamOpen
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_StreamSetClocks
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_Write1
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_Write8
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIO_WriteAll
 - AIOUSB_DIO.c, [345](#)
 - AIOUSB_DIO.h, [347](#)
- DIOBuf, [141](#)
 - buffer, [141](#)
 - size, [141](#)
 - strbuf, [141](#)
 - strbuf_size, [141](#)
- DIOBuf.c
 - DIOBufByteSize, [367](#)
 - DIOBufGetByteAtIndex, [368](#)
 - DIOBufGetIndex, [368](#)
 - DIOBufReplaceBinString, [367](#)
 - DIOBufReplaceString, [366](#)
 - DIOBufResize, [367](#)
 - DIOBufSetByteAtIndex, [368](#)
 - DIOBufSetIndex, [367](#)
 - DIOBufSize, [367](#)
 - DIOBufToBinary, [367](#)
 - DIOBufToHex, [367](#)
 - DIOBufToInvertedBinary, [367](#)
 - DIOBufToString, [367](#)
 - DeleteDIOBuf, [367](#)
 - NewDIOBuf, [366](#)
 - NewDIOBufFromBinStr, [366](#)
 - NewDIOBufFromChar, [366](#)
- DIOBuf.h
 - DIOBufByteSize, [370](#)
 - DIOBufGetByteAtIndex, [371](#)
 - DIOBufGetIndex, [371](#)
 - DIOBufReplaceBinString, [370](#)
 - DIOBufReplaceString, [369](#)
 - DIOBufResize, [370](#)
 - DIOBufSetByteAtIndex, [371](#)
 - DIOBufSetIndex, [370](#)
 - DIOBufSize, [370](#)
 - DIOBufToBinary, [370](#)
 - DIOBufToHex, [370](#)
 - DIOBufToInvertedBinary, [370](#)
 - DIOBufToString, [370](#)
 - DIOBufferType, [369](#)
 - DeleteDIOBuf, [369](#)
 - NewDIOBuf, [369](#)
 - NewDIOBufFromBinStr, [369](#)
 - NewDIOBufFromChar, [369](#)
- DIOBufByteSize
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)

- DIOBufGetByteAtIndex
 - DIOBuf.c, [368](#)
 - DIOBuf.h, [371](#)
- DIOBufGetIndex
 - DIOBuf.c, [368](#)
 - DIOBuf.h, [371](#)
- DIOBufReplaceBinString
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufReplaceString
 - DIOBuf.c, [366](#)
 - DIOBuf.h, [369](#)
- DIOBufResize
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufSetByteAtIndex
 - DIOBuf.c, [368](#)
 - DIOBuf.h, [371](#)
- DIOBufSetIndex
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufSize
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufToBinary
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufToHex
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufToInvertedBinary
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufToString
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [370](#)
- DIOBufferType
 - DIOBuf.h, [369](#)
- DIOBytes
 - AIODeviceInfo, [79](#)
 - AIOUSBDevice, [83](#)
- DIOConfigBits
 - AIOUSBDevice, [83](#)
- DIOPorts
 - DeviceProperties, [134](#)
- DIOStreamSubsystem, [142](#)
 - AIOUSB::DIOStreamSubsystem, [142](#)
 - AIOUSB::USBDeviceBase, [183](#)
- dRef
 - AIOUSB_ADC.c, [317](#)
- DUMP
 - AIOCommandLine.h, [229](#)
 - aiocommon.c, [380](#)
- dac
 - AIOUSB::USB_AIO16_Family, [161](#)
 - AIOUSB::USB_AO16_Family, [164](#)
 - AIOUSB::USB_DA12_8A_Family, [169](#)
 - AIOUSB::USB_DA12_8E_Family, [171](#)
- daisample.c
 - main, [398](#)
- daitest.cpp
 - exit_sample, [387](#)
 - handle_signal, [387](#)
 - main, [387](#)
 - old_action, [387](#)
- dataBuf
 - TestCaseSetup, [156](#)
- debug
 - ADCCConfigBlock, [64](#)
 - AIOArgument, [72](#)
 - AIOContinuousBuf, [78](#)
 - configuration, [120](#)
 - USBDevice, [179](#)
- debug_level
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- default_aiobuf_json
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- default_clock_rate
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- default_end_channel
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- default_num_channels
 - AIOCommandLineOptions, [76](#)
 - opts, [149](#)
- default_num_oversamples
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- default_num_scans
 - AIOCommandLineOptions, [76](#)
 - opts, [149](#)
- default_out
 - AIOCountsConverter.c, [256](#)
- default_start_channel
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- defined
 - AIOBuf, [73](#)
 - mux_settings, [147](#)
- DeleteADCCConfigBlock
 - ADCCConfigBlock.c, [208](#)
 - ADCCConfigBlock.h, [214](#)
- DeleteAIOBuf
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- DeleteAIOChannelMask
 - AIOChannelMask.c, [218](#)
 - AIOChannelMask.h, [220](#)
- DeleteAIOChannelRange
 - AIOChannelRange.c, [223](#)
 - AIOChannelRange.h, [224](#)
- DeleteAIOCmd
 - AIOCcmd.c, [224](#)
 - AIOCcmd.h, [225](#)
- DeleteAIOCommandLineOptions
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [230](#)
- DeleteAIOConfiguration
 - AIOConfiguration.c, [233](#)
- DeleteAIOContinuousBuf
 - AIOContinuousBuffer.c, [240](#)
 - AIOContinuousBuffer.h, [249](#)
- DeleteAIOCountsConverter
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- DeleteAIODeviceInfo
 - AIODeviceInfo.c, [258](#)
 - AIODeviceInfo.h, [259](#)
- DeleteAIODeviceQuery
 - AIODeviceQuery.c, [261](#)
 - AIODeviceQuery.h, [263](#)
- DeleteAIOFifo
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [279](#)
- DeleteAIOFifoTYPE
 - AIOFifo.c, [277](#)
- DeleteAIOGainRange
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- DeleteAIOProductGroup
 - AIOProductTypes.c, [283](#)

- AIOProductTypes.h, [285](#)
- DeleteAIOProductRange
 - AIOProductTypes.c, [283](#)
 - AIOProductTypes.h, [285](#)
- DeleteAIOUSBDevice
 - AIOUSBDevice.c, [357](#)
- DeleteCStringArray
 - CStringArray.c, [364](#)
 - CStringArray.h, [365](#)
- DeleteDIOBuf
 - DIOBuf.c, [367](#)
 - DIOBuf.h, [369](#)
- DeleteUSBDevice
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- DeleteUSBDevices
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- DeleteWDGConfig
 - AIOUSB_WDG.c, [354](#)
 - AIOUSB_WDG.h, [356](#)
- Deleteint
 - AIOList.c, [280](#)
- Deleteintlist
 - AIOList.c, [281](#)
 - AIOList.h, [282](#)
- delta
 - AIOFifo.c, [276](#)
- deprecated/classlib/AI16_DataPoint.cpp, [191](#)
- deprecated/classlib/AI16_DataPoint.hpp, [191](#)
- deprecated/classlib/AI16_DataSet.cpp, [191](#)
- deprecated/classlib/AI16_DataSet.hpp, [191](#)
- deprecated/classlib/AI16_InputRange.cpp, [192](#)
- deprecated/classlib/AI16_InputRange.hpp, [192](#)
- deprecated/classlib/AO16_AnalogOutputSubsystem.cpp, [194](#)
- deprecated/classlib/AO16_AnalogOutputSubsystem.hpp, [194](#)
- deprecated/classlib/AO16_OutputRange.cpp, [194](#)
- deprecated/classlib/AO16_OutputRange.hpp, [194](#)
- deprecated/classlib/AnalogIORange.cpp, [193](#)
- deprecated/classlib/AnalogIORange.hpp, [193](#)
- deprecated/classlib/AnalogInputSubsystem.cpp, [192](#)
- deprecated/classlib/AnalogInputSubsystem.hpp, [193](#)
- deprecated/classlib/AnalogOutputSubsystem.cpp, [193](#)
- deprecated/classlib/AnalogOutputSubsystem.hpp, [194](#)
- deprecated/classlib/Counter.cpp, [195](#)
- deprecated/classlib/Counter.hpp, [195](#)
- deprecated/classlib/CounterSubsystem.cpp, [196](#)
- deprecated/classlib/CounterSubsystem.hpp, [196](#)
- deprecated/classlib/CppCommon.h, [196](#)
- deprecated/classlib/DA12_AnalogOutputSubsystem.cpp, [196](#)
- deprecated/classlib/DA12_AnalogOutputSubsystem.hpp, [197](#)
- deprecated/classlib/DA12_OutputRange.cpp, [197](#)
- deprecated/classlib/DA12_OutputRange.hpp, [197](#)
- deprecated/classlib/DIOStreamSubsystem.cpp, [199](#)
- deprecated/classlib/DIOStreamSubsystem.hpp, [199](#)
- deprecated/classlib/DeviceSubsystem.cpp, [198](#)
- deprecated/classlib/DeviceSubsystem.hpp, [198](#)
- deprecated/classlib/DigitalIOSubsystem.cpp, [198](#)
- deprecated/classlib/DigitalIOSubsystem.hpp, [198](#)
- deprecated/classlib/OutputVoltagePoint.hpp, [199](#)
- deprecated/classlib/README.doc, [200](#)
- deprecated/classlib/USB_AI16_Family.cpp, [200](#)
- deprecated/classlib/USB_AI16_Family.hpp, [200](#)
- deprecated/classlib/USB_AIO16_Family.cpp, [201](#)
- deprecated/classlib/USB_AIO16_Family.hpp, [201](#)
- deprecated/classlib/USB_AO16_Family.cpp, [201](#)
- deprecated/classlib/USB_AO16_Family.hpp, [201](#)
- deprecated/classlib/USB_AO16_Family.cpp, [201](#)
- deprecated/classlib/USB_AO16_Family.hpp, [201](#)
- deprecated/classlib/USB_CTR_15_Family.cpp, [202](#)
- deprecated/classlib/USB_CTR_15_Family.hpp, [202](#)
- deprecated/classlib/USB_DA12_8A_Family.cpp, [202](#)
- deprecated/classlib/USB_DA12_8A_Family.hpp, [202](#)
- deprecated/classlib/USB_DA12_8E_Family.cpp, [203](#)
- deprecated/classlib/USB_DA12_8E_Family.hpp, [203](#)
- deprecated/classlib/USB_DIO_16_Family.cpp, [203](#)
- deprecated/classlib/USB_DIO_16_Family.hpp, [203](#)
- deprecated/classlib/USB_DIO_32_Family.cpp, [204](#)
- deprecated/classlib/USB_DIO_32_Family.hpp, [204](#)
- deprecated/classlib/USB_DIO_Family.cpp, [204](#)
- deprecated/classlib/USB_DIO_Family.hpp, [204](#)
- deprecated/classlib/USBDeviceBase.cpp, [205](#)
- deprecated/classlib/USBDeviceBase.hpp, [205](#)
- deprecated/classlib/USBDeviceManager.cpp, [206](#)
- deprecated/classlib/USBDeviceManager.hpp, [206](#)
- dev
 - aiousb_libusb_args, [82](#)
- device
 - ADCCfgBlock, [63](#)
 - USBDevice, [179](#)
- device_args
 - AIOArguments, [73](#)
- device_index
 - configuration, [119](#)
- deviceDesc
 - aiousb_libusb_args, [82](#)
 - USBDevice, [179](#)
- DeviceDescriptor
 - AIOUSBDevice.h, [358](#)
- DeviceEnum
 - AIOCommandLine.h, [229](#)
- deviceFound
 - TestCaseSetup, [156](#)
- deviceHandle
 - USBDevice, [179](#)
- DeviceIndex
 - AIOContinuousBuf, [78](#)
 - BulkAcquireWorkerParams, [117](#)
 - TestCaseSetup, [156](#)
- deviceIndex
 - AIOUSB::USBDeviceBase, [183](#)
 - AIOUSBDevice, [83](#)
- DeviceInfo, [133](#)
 - index, [134](#)
 - name, [133](#)
 - nameSize, [133](#)
 - numCounters, [134](#)
 - numDIOBytes, [134](#)
 - outputMask, [133](#)
 - productID, [133](#)
 - readBuffer, [133](#)
 - serialNumber, [134](#)
 - writeBuffer, [133](#)
- deviceList
 - AIOUSB::USBDeviceManager, [189](#)
- DeviceProperties, [134](#)
 - ADCCChannels, [135](#)
 - ADCCChannelsPerGroup, [135](#)
 - ADCMUXChannels, [135](#)
 - Counters, [135](#)
 - DACChannels, [135](#)
 - DIOPorts, [134](#)
 - Name, [134](#)
 - ProductID, [134](#)
 - RootClock, [135](#)
 - SerialNumber, [134](#)
 - Tristates, [135](#)
- DeviceSubsystem, [135](#)
 - AIOUSB::DeviceSubsystem, [136](#)
- deviceTable
 - AIODeviceTable.c, [268](#)
 - AIODeviceTable.h, [272](#)

- DeviceTableAtIndex
 - AIOUSB_Core.c, [329](#)
 - AIOUSB_Core.h, [333](#)
- DeviceTableAtIndex_Lock
 - AIOUSB_Core.c, [329](#)
 - AIOUSB_Core.h, [333](#)
- diFirst
 - AIOTypes.h, [296](#)
- diNone
 - AIOTypes.h, [296](#)
- diOnly
 - AIOTypes.h, [296](#)
- differentialMode
 - AIOUSB::AI16_DataPoint, [66](#)
 - AIOUSB::AnalogInputSubsystem, [106](#)
- DigitalIOSubsystem, [136](#)
 - AIOUSB::DigitalIOSubsystem, [137](#)
- digitalIOSubsystem
 - AIOUSB::USB_AI16_Family, [159](#)
 - AIOUSB::USB_AIO16_Family, [162](#)
 - AIOUSB::USB_AO16_Family, [164](#)
 - AIOUSB::USB_DIO_16_Family, [173](#)
 - AIOUSB::USB_DIO_32_Family, [176](#)
 - AIOUSB::USB_DIO_Family, [178](#)
- dio
 - AIOUSB::USB_AI16_Family, [159](#)
 - AIOUSB::USB_AIO16_Family, [162](#)
 - AIOUSB::USB_AO16_Family, [164](#)
 - AIOUSB::USB_DIO_16_Family, [173](#)
 - AIOUSB::USB_DIO_32_Family, [175](#)
 - AIOUSB::USB_DIO_Family, [178](#)
- dio96_read_write.c
 - BITS_PER_BYTE, [402](#)
 - DEVICES_REQUIRED, [402](#)
 - find_dio_96, [403](#)
 - MAKE_MASK, [402](#)
 - MASK_BYTES, [402](#)
 - MAX_DIO_BYTES, [402](#)
 - MAX_NAME_SIZE, [402](#)
 - main, [403](#)
 - PORT_A, [402](#)
 - PORT_B, [402](#)
 - PORT_C, [402](#)
 - show_byte, [403](#)
- dio_sample.c
 - find_ai_board, [388](#)
 - main, [388](#)
- dioStreamSubsystem
 - AIOUSB::USB_DIO_16_Family, [173](#)
- diostream
 - AIOUSB::USB_DIO_16_Family, [173](#)
- diotest.c
 - main, [388](#)
- diotest2.cpp
 - main, [388](#)
- direction
 - mock_usb_xfers.c, [376](#)
- dirktest.c
 - capture_data, [389](#)
 - fnd, [389](#)
 - fp, [389](#)
 - get_channel_range, [389](#)
 - main, [389](#)
 - process_cmd_line, [389](#)
- discard_first_sample
 - configuration, [119](#)
- discardFirstSample
 - ADCCConfigBlock, [64](#)
 - aio_counts_converter, [71](#)
 - AIOUSB::AI16_DataSet, [69](#)
 - AIOUSBDevice, [83](#)
- doBulkAcquire
 - TestCaseSetup, [155](#)
- doBulkConfigBlock
 - TestCaseSetup, [154](#)
- doCSVReadVoltages
 - TestCaseSetup, [155](#)
- doCSVWithGetChannelV
 - TestCaseSetup, [155](#)
- doCleanupAfterBulk
 - TestCaseSetup, [155](#)
- doDACDirect
 - TestCaseSetup, [155](#)
- doDACDirectSetup
 - TestCaseSetup, [155](#)
- doDemonstrateReadVoltages
 - TestCaseSetup, [154](#)
- doDisplayBulkResults
 - TestCaseSetup, [156](#)
- doFastITScan
 - TestCaseSetup, [155](#)
- doFastITScanSetup
 - TestCaseSetup, [155](#)
- doGenericVendorWrite
 - TestCaseSetup, [155](#)
- doGetBuffer
 - TestCaseSetup, [155](#)
- DoLoadCalTable
 - AIOUSB_ADC.c, [314](#)
- doPreReadImmediateVoltages
 - TestCaseSetup, [155](#)
- doPreSetup
 - TestCaseSetup, [154](#)
- doScanSingleChannel
 - TestCaseSetup, [155](#)
- doSetAutoCalibration
 - TestCaseSetup, [154](#)
- doSomething
 - AIOUSB_WDG.c, [354](#)
 - TestCaseSetup, [154](#)
- doTestSetAutoCalibration
 - TestCaseSetup, [155](#)
- doVerifyGroundCalibration
 - TestCaseSetup, [154](#)
- doVerifyReferenceCalibration
 - TestCaseSetup, [154](#)
- doc/aiousb.doc, [207](#)
- doc/firmware.doc, [207](#)
- doc/index.doc, [207](#)
- doc/install.doc, [207](#)
- doc/java.doc, [207](#)
- doc/libusb.doc, [207](#)
- doc/samples.doc, [207](#)
- doc/wrappers.doc, [207](#)
- DoubleArray, [145](#)
 - AIOUSB::DoubleArray, [145](#)
- EEPROM_CUSTOM_BASE_ADDRESS
 - AIOTypes.h, [301](#)
- EEPROM_CUSTOM_MAX_ADDRESS
 - AIOTypes.h, [301](#)
- EEPROM_CUSTOM_MIN_ADDRESS
 - AIOTypes.h, [301](#)
- EEPROM_SERIAL_NUMBER_ADDRESS
 - AIOTypes.h, [301](#)
- END
 - AIOChannelRange.c, [222](#)
- END_CHANNEL
 - AIOChannelRange.c, [222](#)
- ERROR
 - TestCaseSetup.h, [383](#)
- ERROR_LEVEL
 - TestCaseSetup.h, [383](#)
- EXIT_CODE

- USB-DIO-96/read_and_write_sample.c, [400](#)
- write_sample.c, [407](#)
- EXPORTED_FUNCTION
 - AIOTypes.h, [291](#)
- emptyDeviceList
 - AIOUSB::USBDeviceManager, [185](#)
- end
 - aio_channel_range, [70](#)
- end_channel
 - AIOChannelRangeTmp, [74](#)
 - AIOCommandLineOptions, [76](#)
 - channel_range, [118](#)
 - opts, [150](#)
- endChannel
 - AIOUSB::AnalogInputSubsystem, [106](#)
- endpos
 - AIOBuf, [73](#)
- enhanced_out
 - AIOCountsConverter.c, [256](#)
- EnumStringLookup
 - AIOTypes.h, [293](#)
- envGetDouble
 - TestCaseSetup, [155](#)
- envGetInteger
 - TestCaseSetup, [155](#)
- errmsg
 - aio_ret_value, [72](#)
- Error, [145](#)
 - Error, [146](#)
 - message, [146](#)
 - what, [146](#)
- error_message
 - aioerror, [80](#)
- exit_sample
 - daitest.cpp, [387](#)
- exitcode
 - AIOContinuousBuf, [78](#)
- extcal.cpp
 - main, [392](#)
- FIFO_Method_begin
 - AIOTypes.h, [297](#)
- FIFO_Method_end
 - AIOTypes.h, [297](#)
- FILE_OPT
 - AIOCommandLine.h, [229](#)
- FATAL
 - TestCaseSetup.h, [383](#)
- FATAL_LEVEL
 - TestCaseSetup.h, [383](#)
- FIFO_Method
 - AIOTypes.h, [296](#)
- FIRST_ENUM
 - AIOTypes.h, [291](#)
- FastITBakConfig
 - AIOUSBDevice, [85](#)
- FastITConfig
 - AIOUSBDevice, [85](#)
- FastITConfig_size
 - AIOUSBDevice, [85](#)
- fifo
 - AIOContinuousBuf, [78](#)
- file_handle
 - configuration, [120](#)
- file_name
 - configuration, [120](#)
- find_ai_board
 - burst_test.c, [385](#)
 - dio_sample.c, [388](#)
 - jni/read_channels_test.c, [392](#)
 - julian_test.c, [389](#)
 - read_channels_test.c, [393](#)
 - sample_dio.c, [390](#)
 - stream_test.c, [397](#)
- find_dio
 - read_and_write.c, [402](#)
- find_dio_96
 - dio96_read_write.c, [403](#)
 - mytest.c, [404](#)
 - tmp.c, [406](#)
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [408](#)
- find_idio
 - iio_sample.c, [411](#)
- findDevice
 - TestCaseSetup, [154](#)
- Firmware/README.doc, [200](#)
- FlashSectors
 - AIOUSBDevice, [84](#)
- fn
 - continuous_mode.c, [385](#)
 - continuous_mode_callback.c, [386](#)
 - continuous_mode_from_json_config.c, [387](#)
 - dirktest.c, [389](#)
 - simple_continuous_with_json.c, [396](#)
 - start_stop_continuous.c, [396](#)
 - test.c, [391](#)
 - write_sample.c, [408](#)
- foo.c
 - foo_something, [380](#)
- foo.h
 - foo_something, [380](#)
- foo_something
 - foo.c, [380](#)
 - foo.h, [380](#)
- foreach_CStringArray_p
 - AIOList.h, [282](#)
- foreach_array
 - AIOTypes.h, [291](#)
- foreach_int
 - AIOList.h, [282](#)
- fp
 - continuous_mode_callback.c, [386](#)
 - continuous_mode_from_json_config.c, [387](#)
 - dirktest.c, [389](#)
 - simple_continuous_with_json.c, [396](#)
 - start_stop_continuous.c, [396](#)
 - test.c, [391](#)
- framePoints
 - config_options, [119](#)
- free_fn
 - cJSON_Hooks, [118](#)
- GAIN
 - AIOChannelRange.c, [222](#)
- GENERIC_DOSOMETHING_PLACEHOLDER
 - AIOTypes.h, [301](#)
- G_BREAKPOINT
 - AIOTypes.h, [293](#)
- G_STMT_END
 - AIOTypes.h, [292](#)
- G_STMT_START
 - AIOTypes.h, [292](#)
- GCC_VERSION
 - AIOTypes.h, [291](#)
- GET_ENDPOINT
 - AIOUSB_DIO.c, [344](#)
- GRAB_RESOURCE
 - AIOFifo.h, [278](#)
- GREEN
 - AIOUSB_Log.h, [349](#)
- gain
 - ad_gain_pairs, [63](#)
 - aio_channel_range, [70](#)

- gain_code
 - AIOCommandLineOptions, 76
 - opts, 150
- gain_ranges
 - aio_counts_converter, 71
- gainCodes
 - TestCaseSetup, 156
- gaincode
 - AIOChannelRangeTmp, 74
 - channel_range, 118
- GenericVendorRead
 - AIOUSB_Core.c, 330
 - AIOUSB_Core.h, 335
- GenericVendorWrite
 - AIOUSB_Core.c, 330
 - AIOUSB_Core.h, 335
- get_cal_mode
 - ADCConfigBlock.c, 210
- get_channel_range
 - aiocommon.c, 380
 - aiocommon.h, 381
 - continuous_mode_callback.c, 386
 - continuous_mode_from_json_config.c, 387
 - dirktest.c, 389
 - simple_continuous_with_json.c, 395
 - start_stop_continuous.c, 396
 - test.c, 391
- get_gain_code
 - ADCConfigBlock.c, 210
- get_options
 - read_channels_with_getchannelv_test.cpp, 393
- get_usb_device
 - USBDevice.c, 377
 - USBDevice.h, 379
- getAIOUSBVersion
 - AIOUSB::USBDeviceManager, 185
- getAIOUSBVersionDate
 - AIOUSB::USBDeviceManager, 185
- getCalMode
 - AIOUSB::AI16_DataSet, 68
 - AIOUSB::AnalogInputSubsystem, 92
- getChannel
 - AIOUSB::AI16_DataPoint, 65
- getChannelsPerGroup
 - AIOUSB::AnalogInputSubsystem, 90
- getClock
 - AIOUSB::AnalogInputSubsystem, 98
 - AIOUSB::DIOStreamSubsystem, 143
- getCommTimeout
 - AIOUSB::USBDeviceBase, 182
- getConfiguration
 - AIOUSB::DigitalIOSubsystem, 139
- getCounter
 - AIOUSB::CounterSubsystem, 124
- getCounts
 - AIOUSB::AI16_DataPoint, 65
 - TestCaseSetup, 156
- getDeviceByProductID
 - AIOUSB::USBDeviceManager, 188, 189
- GetDeviceBySerialNumber
 - AIOUSB_Properties.c, 351
 - AIOUSB_Properties.h, 353
- getDeviceBySerialNumber
 - AIOUSB::USBDeviceManager, 189
- getDeviceIndex
 - AIOUSB::Counter, 121
 - AIOUSB::DeviceSubsystem, 136
 - AIOUSB::USBDeviceBase, 181
- GetDeviceSerialNumber
 - AIOUSB_Properties.c, 351
 - AIOUSB_Properties.h, 353
- GetDevices
 - AIODeviceTable.c, 267
 - AIODeviceTable.h, 270
- getGainCodes
 - TestCaseSetup, 156
- GetHiRef
 - AIOUSB_ADC.c, 314
- GetJSONValueOrDefault
 - AIOContinuousBuffer.c, 245
- getMiscClock
 - AIOUSB::USBDeviceBase, 181
- getName
 - AIOUSB::USBDeviceBase, 181
- getNumChannels
 - AIOUSB::AnalogInputSubsystem, 90
 - AIOUSB::AnalogOutputSubsystem, 110
 - AIOUSB::DigitalIOSubsystem, 137
- getNumCounterBlocks
 - AIOUSB::CounterSubsystem, 124
- getNumCounters
 - AIOUSB::CounterSubsystem, 124
- getNumMUXChannels
 - AIOUSB::AnalogInputSubsystem, 90
- getNumPorts
 - AIOUSB::DigitalIOSubsystem, 137
- getNumTristateChannels
 - AIOUSB::DigitalIOSubsystem, 138
- getNumTristateGroups
 - AIOUSB::DigitalIOSubsystem, 138
- getOverSample
 - AIOUSB::AI16_DataSet, 68
 - AIOUSB::AnalogInputSubsystem, 96
- getParent
 - AIOUSB::DeviceSubsystem, 136
- getPoints
 - AIOUSB::AI16_DataSet, 68
- getProductID
 - AIOUSB::USBDeviceBase, 181
- getRange
 - AIOUSB::AI16_DataPoint, 65
 - AIOUSB::AnalogInputSubsystem, 93
 - AIOUSB::AnalogIORange, 107
 - AIOUSB::AO16_AnalogOutputSubsystem, 113
 - AIOUSB::DA12_AnalogOutputSubsystem, 128
- getRangeText
 - AIOUSB::AI16_DataPoint, 65
 - AIOUSB::AnalogInputSubsystem, 90
 - AIOUSB::AO16_AnalogOutputSubsystem, 113
 - AIOUSB::DA12_AnalogOutputSubsystem, 128
- getResultCodeAsString
 - AIOUSB::USBDeviceManager, 187
- GetSafeDeviceName
 - AIODeviceTable.c, 267
 - AIODeviceTable.h, 270
- getSerialNumber
 - AIOUSB::USBDeviceBase, 181
- getStreamingBlockSize
 - AIOUSB::AnalogInputSubsystem, 97
 - AIOUSB::DIOStreamSubsystem, 143
 - AIOUSB::USBDeviceBase, 181
- getSubsystem
 - AIOUSB::AI16_DataSet, 68
- getSupportedProductIDs
 - AIOUSB::USB_AI16_Family, 158
 - AIOUSB::USB_AIO16_Family, 161
 - AIOUSB::USB_AO16_Family, 163
 - AIOUSB::USB_CTR_15_Family, 166
 - AIOUSB::USB_DA12_8A_Family, 168
 - AIOUSB::USB_DA12_8E_Family, 170
 - AIOUSB::USB_DIO_16_Family, 173
 - AIOUSB::USB_DIO_32_Family, 175
 - AIOUSB::USB_DIO_Family, 177
- getSupportedProductNames

- AIOUSB::USB_AI16_Family, [158](#)
 - AIOUSB::USB_AIO16_Family, [161](#)
 - AIOUSB::USB_AO16_Family, [163](#)
 - AIOUSB::USB_CTR_15_Family, [166](#)
 - AIOUSB::USB_DA12_8A_Family, [168](#)
 - AIOUSB::USB_DA12_8E_Family, [170](#)
 - AIOUSB::USB_DIO_16_Family, [172](#)
 - AIOUSB::USB_DIO_32_Family, [175](#)
 - AIOUSB::USB_DIO_Family, [177](#)
- getTimeStamp
 - AIOUSB::AI16_DataSet, [68](#)
- getTriggerMode
 - AIOUSB::AI16_DataSet, [68](#)
 - AIOUSB::AnalogInputSubsystem, [93](#)
- getVolts
 - AIOUSB::AI16_DataPoint, [66](#)
 - TestCaseSetup, [156](#)
- goDolt
 - HOLD/reverse_cal_table.cpp, [389](#)
 - reverse_cal_table.cpp, [390](#)
- groupcpy
 - AIOProductTypes.c, [283](#)
 - AIOProductTypes.h, [285](#)
- HAS_PTHREAD
 - AIOTypes.h, [291](#)
- hDACDataMutex
 - AIOUSBDevice, [84](#)
- hDACDataSem
 - AIOUSBDevice, [84](#)
- HOLD/reverse_cal_table.cpp
 - goDolt, [389](#)
 - main, [389](#)
- handle
 - aiusb_libusb_args, [82](#)
- handle_signal
 - daitest.cpp, [387](#)
- hz
 - AIOContinuousBuf, [78](#)
- IN
 - mock_capture_usb.c, [373](#)
 - mock_dio.c, [374](#), [375](#)
 - mock_usb_xfers.c, [376](#)
- INDEX_NUM
 - AIOCommandLine.h, [229](#)
- INVALID_OBJECT
 - AIOTypes.h, [293](#)
- INFO
 - TestCaseSetup.h, [383](#)
- INFO_LEVEL
 - TestCaseSetup.h, [383](#)
- INPUT_TYPE
 - AIOFifo.h, [279](#)
- INTERNAL_METHOD
 - USBDevice.h, [379](#)
- IO_DIRECTION
 - mock_capture_usb.c, [373](#)
 - mock_dio.c, [374](#)
 - mock_usb_xfers.c, [376](#)
- id
 - ProductIDName, [152](#)
- idio_sample.c
 - main, [409](#)
 - RATE_LIMIT, [409](#)
- idio_sample2.c
 - main, [410](#)
 - RATE_LIMIT, [409](#)
- iface
 - USBDevice, [179](#)
- iio_sample.c
 - find_idio, [411](#)
 - main, [411](#)
- RATE_LIMIT, [411](#)
- IllegalArgumentException, [146](#)
 - AIOUSB::IllegalArgumentException, [146](#)
- ImmADCs
 - AIOUSBDevice, [84](#)
- ImmDACs
 - AIOUSBDevice, [84](#)
- increment
 - AIOFifo.c, [277](#)
- index
 - AIOCommandLineOptions, [76](#)
 - AIODeviceQuery, [80](#)
 - DeviceInfo, [134](#)
 - opts, [150](#)
- init_device
 - mock_capture_usb.c, [373](#)
- initialize
 - AIOUSB::USB_AI16_Family, [158](#)
 - AIOUSB::USB_AIO16_Family, [161](#)
 - AIOUSB::USB_AO16_Family, [163](#)
 - AIOUSB::USB_CTR_15_Family, [166](#)
 - AIOUSB::USB_DA12_8A_Family, [168](#)
 - AIOUSB::USB_DA12_8E_Family, [170](#)
 - AIOUSB::USB_DIO_16_Family, [172](#)
 - AIOUSB::USB_DIO_32_Family, [175](#)
 - AIOUSB::USB_DIO_Family, [177](#)
- InitializeUSBDevice
 - mock_capture_usb.c, [374](#)
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- inputRange
 - AIOUSB::AnalogInputSubsystem, [106](#)
- installation, [5](#)
- IntArray, [146](#)
 - AIOUSB::IntArray, [146](#)
- intToString
 - AIOList.c, [280](#)
- intlistFirst
 - AIOList.c, [281](#)
 - AIOList.h, [282](#)
- intlistInsert
 - AIOList.c, [281](#)
 - AIOList.h, [282](#)
- intlistSize
 - AIOList.c, [281](#)
 - AIOList.h, [282](#)
- intlistToString
 - AIOList.c, [281](#)
 - AIOList.h, [282](#)
- is_all_digits
 - ADCCConfigBlock.c, [210](#)
 - ADCCConfigBlock.h, [214](#)
- isAutoCalPresent
 - AIOUSB::AnalogInputSubsystem, [90](#)
- isAutoConfig
 - AIOUSB::AnalogInputSubsystem, [91](#)
- isDifferentialMode
 - AIOUSB::AI16_DataPoint, [65](#)
 - AIOUSB::AnalogInputSubsystem, [94](#)
- isDiscardFirstSample
 - AIOUSB::AI16_DataSet, [68](#)
 - AIOUSB::AnalogInputSubsystem, [92](#)
- isInit
 - AIOUSBDevice, [83](#)
- isOpen
 - AIOUSB::USBDeviceManager, [187](#)
- isSupportedProductID
 - AIOUSB::USB_AI16_Family, [158](#)
 - AIOUSB::USB_AIO16_Family, [161](#)
 - AIOUSB::USB_AO16_Family, [164](#)
 - AIOUSB::USB_CTR_15_Family, [166](#)
 - AIOUSB::USB_DA12_8A_Family, [168](#)

- AIOUSB::USB_DA12_8E_Family, [170](#)
 - AIOUSB::USB_DIO_16_Family, [173](#)
 - AIOUSB::USB_DIO_32_Family, [175](#)
 - AIOUSB::USB_DIO_Family, [177](#)
- JOINED
 - AIOTypes.h, [293](#)
- JSON
 - AIOUSB_Properties.h, [352](#)
- JCONF
 - AIOCommandLine.h, [229](#)
 - aiocommon.c, [380](#)
- jni/read_channels_test.c
 - find_ai_board, [392](#)
 - main, [392](#)
- julian_test.c
 - find_ai_board, [389](#)
 - main, [389](#)
- KEEP
 - mock_usb_xfers.c, [376](#)
- L
 - AIOWDGConfig, [86](#)
- LAMBDA
 - AIOTypes.h, [291](#)
- LAST_ENUM
 - AIOTypes.h, [291](#)
- LIBUSBArgs
 - USBDevice.h, [379](#)
- LOCKING_MECHANISM
 - AIOFifo, [80](#)
 - AIOFifo.h, [278](#)
 - new_aio_fifo, [148](#)
- LOG
 - TestCaseSetup.h, [383](#)
- LOG_LEVEL
 - AIOUSB_Log.h, [349](#)
- LOOKUP
 - AIOEither.c, [272](#)
 - AIOFifo.c, [276](#)
- LastDIOData
 - AIOUSBDevice, [85](#)
- left
 - aio_ret_value, [72](#)
- lib/ADCCConfigBlock.c, [207](#)
- lib/ADCCConfigBlock.h, [211](#)
- lib/AIOBuf.c, [214](#)
- lib/AIOBuf.h, [215](#)
- lib/AIOChannelMask.c, [217](#)
- lib/AIOChannelMask.h, [219](#)
- lib/AIOChannelRange.c, [222](#)
- lib/AIOChannelRange.h, [223](#)
- lib/AIOCmd.c, [224](#)
- lib/AIOCmd.h, [224](#)
- lib/AIOCommandLine.c, [225](#)
- lib/AIOCommandLine.h, [228](#)
- lib/AIOConfiguration.c, [232](#)
- lib/AIOConfiguration.h, [233](#)
- lib/AIOContinuousBuffer.c, [234](#)
- lib/AIOContinuousBuffer.h, [246](#)
- lib/AIOCountsConverter.c, [255](#)
- lib/AIOCountsConverter.h, [256](#)
- lib/AIODeviceInfo.c, [258](#)
- lib/AIODeviceInfo.h, [259](#)
- lib/AIODeviceQuery.c, [260](#)
- lib/AIODeviceQuery.h, [262](#)
- lib/AIODeviceTable.c, [265](#)
- lib/AIODeviceTable.h, [269](#)
- lib/AIOEither.c, [272](#)
- lib/AIOEither.h, [273](#)
- lib/AIOFifo.c, [275](#)
- lib/AIOFifo.h, [277](#)
- lib/AIOList.c, [280](#)
- lib/AIOList.h, [281](#)
- lib/AIOProductTypes.c, [282](#)
- lib/AIOProductTypes.h, [284](#)
- lib/AIOTuple.c, [285](#)
- lib/AIOTuple.h, [286](#)
- lib/AIOTypes.h, [286](#)
- lib/AIOUSB_ADC.c, [302](#)
- lib/AIOUSB_ADC.h, [318](#)
- lib/AIOUSB_CTR.c, [335](#)
- lib/AIOUSB_CTR.h, [337](#)
- lib/AIOUSB_Core.c, [327](#)
- lib/AIOUSB_Core.h, [331](#)
- lib/AIOUSB_CustomEEPROM.c, [338](#)
- lib/AIOUSB_CustomEEPROM.h, [339](#)
- lib/AIOUSB_DAC.c, [340](#)
- lib/AIOUSB_DAC.h, [342](#)
- lib/AIOUSB_DIO.c, [343](#)
- lib/AIOUSB_DIO.h, [346](#)
- lib/AIOUSB_Log.c, [347](#)
- lib/AIOUSB_Log.h, [348](#)
- lib/AIOUSB_Properties.c, [350](#)
- lib/AIOUSB_Properties.h, [352](#)
- lib/AIOUSB_USB.c, [353](#)
- lib/AIOUSB_USB.h, [354](#)
- lib/AIOUSB_WDG.c, [354](#)
- lib/AIOUSB_WDG.h, [355](#)
- lib/AIOUSBDevice.c, [357](#)
- lib/AIOUSBDevice.h, [358](#)
- lib/CStringArray.c, [364](#)
- lib/CStringArray.h, [364](#)
- lib/DIOBuf.c, [365](#)
- lib/DIOBuf.h, [368](#)
- lib/USBDevice.c, [376](#)
- lib/USBDevice.h, [378](#)
- lib/aiousb.h, [301](#)
- lib/cJSON.c, [359](#)
- lib/cJSON.h, [361](#)
- lib/mocks/mock_aiocontbuf_get_data.c, [371](#)
- lib/mocks/mock_aiocontbuf_get_data_arduino.c, [371](#)
- lib/mocks/mock_capture_usb.c, [372](#)
- lib/mocks/mock_dio.c, [374](#)
- lib/mocks/mock_usb_xfers.c, [375](#)
- lib/wrappers/README.doc, [200](#)
- lib/wrappers/README.md, [411](#)
- lib/wrappers/scilab/foo.c, [379](#)
- lib/wrappers/scilab/foo.h, [380](#)
- libusb, [29](#)
- listDevices
 - AIOUSB::USBDeviceManager, [187](#)
- loc
 - aiobuf_iterator, [73](#)
- lock
 - AIOContinuousBuf, [78](#)
- lookup, [147](#)
 - str, [147](#)
 - strvalue, [147](#)
 - value, [147](#)
- lookup_voltage_range
 - AIOChannelRange.c, [223](#)
- MAX_IMM_ADACS
 - AIOTypes.h, [301](#)
- MAX_USB_DEVICES
 - AIOTypes.h, [294](#)
- MAGENTA
 - AIOUSB_Log.h, [349](#)
- MAKE_MASK
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [406](#)
- MASK_BYTES

- dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [405](#)
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [407](#)
- MASK_BYTES_SIZE
 - AIOUSB_DIO.c, [344](#)
- MAX
 - AIOTypes.h, [291](#)
- MAX_CHANNELS
 - AIOUSB::AnalogInputSubsystem, [106](#)
 - TestCaseSetup, [156](#)
- MAX_COUNTS
 - AIOUSB::AnalogInputSubsystem, [105](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- MAX_DIO_BYTES
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [405](#)
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [407](#)
- MAX_NAME_SIZE
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - TestCaseSetup.h, [383](#)
 - tmp.c, [405](#)
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [407](#)
- MAX_OVERSAMPLE
 - AIOUSB::AnalogInputSubsystem, [105](#)
- MAX_PRODUCT_ID
 - AIOUSB::USBDeviceManager, [189](#)
- MAX_VALUE
 - AIOTypes.h, [291](#)
- MIN
 - AIOTypes.h, [291](#)
- MIN_COUNTS
 - AIOUSB::AnalogInputSubsystem, [105](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- MIN_PRODUCT_ID
 - AIOUSB::USBDeviceManager, [189](#)
- MIN_VALUE
 - AIOTypes.h, [291](#)
- MODE_ONE_SHOT
 - AIOUSB::Counter, [122](#)
- MODE_SQUARE_WAVE
 - AIOUSB::Counter, [123](#)
- main
 - bulk_acquire_sample.c, [384](#)
 - continuous_mode_callback.c, [386](#)
 - continuous_mode_from_json_config.c, [387](#)
 - daisample.c, [398](#)
 - daitest.cpp, [387](#)
 - dio96_read_write.c, [403](#)
 - dio_sample.c, [388](#)
 - diotest.c, [388](#)
 - diotest2.cpp, [388](#)
 - dirktest.c, [389](#)
 - extcal.cpp, [392](#)
 - HOLD/reverse_cal_table.cpp, [389](#)
 - idio_sample.c, [409](#)
 - idio_sample2.c, [410](#)
 - iirro_sample.c, [411](#)
 - jni/read_channels_test.c, [392](#)
 - julian_test.c, [389](#)
 - mytest.c, [405](#)
 - perftest.c, [410](#)
 - read_and_write.c, [402](#)
 - read_channels_test.c, [393](#)
 - read_channels_with_getchannelv_test.cpp, [393](#)
 - receiver.cpp, [398](#)
 - reverse_cal_table.cpp, [390](#)
 - sample3.c, [401](#)
 - sample_dio.c, [390](#)
 - SampleClass.cpp, [397](#)
 - simp_test.cpp, [395](#)
 - simple_continuous_with_json.c, [396](#)
 - slow_receiver_test.cpp, [390](#)
 - standalone_receiver.c, [398](#)
 - start_stop_continuous.c, [396](#)
 - stream_test.c, [397](#)
 - test.c, [391](#)
 - test_fastscan.cpp, [397](#)
 - tmp.c, [406](#)
 - USB-AI16-16/sample.cpp, [394](#)
 - USB-AO16-16/sample.cpp, [394](#)
 - USB-DA12-8A/sample.cpp, [394](#)
 - USB-DIO-16/sample.cpp, [395](#)
 - USB-DIO-32/read_and_write_sample.c, [398](#)
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [408](#)
- malloc_fn
 - cJSON_Hooks, [118](#)
- mask
 - AIOContinuousBuf, [78](#)
- max
 - AIOGainRange, [81](#)
- max_channels
 - opts, [150](#)
- maxCounts
 - AIOUSB::AnalogIORange, [109](#)
 - AIOUSB::AnalogOutputSubsystem, [111](#)
- maxVolts
 - AIOUSB::AnalogIORange, [109](#)
- maxcount
 - options, [149](#)
- maxcounts
 - TestCaseSetup, [156](#)
- maxvalue
 - rangelookup, [152](#)
- message
 - Error, [146](#)
- message_lock
 - AIOUSB_Log.c, [348](#)
 - AIOUSB_Log.h, [350](#)
- min
 - AIOGainRange, [81](#)
- minCounts
 - AIOUSB::AnalogIORange, [109](#)
 - AIOUSB::AnalogOutputSubsystem, [111](#)
- minVolts
 - ADRange, [64](#)
 - AIOUSB::AnalogIORange, [109](#)
- minvalue
 - rangelookup, [152](#)
- miscClockHz
 - AIOUSBDevice, [83](#)
- mock_capture_usb.c
 - IN, [373](#)
 - OUT, [373](#)
- mock_dio.c
 - IN, [374](#), [375](#)
 - OUT, [374](#), [375](#)
- mock_usb_xfers.c
 - IN, [376](#)
 - OUT, [376](#)
- mock_USBDeviceFetchADCCConfigBlock
 - mock_usb_xfers.c, [376](#)
- mock_USBDevicePutADCCConfigBlock
 - mock_usb_xfers.c, [376](#)
- mock_aiocontbuf_get_data.c
 - aiocontbuf_get_bulk_data, [371](#)

- mock_aiocontbuf_get_data_arduino.c
 - adc_get_bulk_data, [372](#)
 - aiocontbuf_get_bulk_data, [372](#)
 - CloseAllDevices, [372](#)
- mock_capture_usb.c
 - IO_DIRECTION, [373](#)
 - init_device, [373](#)
 - InitializeUSBDevice, [374](#)
 - mock_usb_bulk_transfer, [373](#)
 - mock_usb_control_transfer, [373](#)
 - mock_usb_get_config, [374](#)
 - mock_usb_put_config, [374](#)
 - mock_usb_request, [373](#)
 - mock_usb_reset_device, [373](#)
 - orig_usb_bulk_transfer, [374](#)
 - orig_usb_control_transfer, [374](#)
 - orig_usb_get_config, [374](#)
 - orig_usb_put_config, [374](#)
 - orig_usb_request, [374](#)
 - orig_usb_reset_device, [374](#)
 - outfile, [374](#)
- mock_dio.c
 - IO_DIRECTION, [374](#)
 - mock_usb_bulk_transfer, [375](#)
 - mock_usb_control_transfer, [375](#)
 - save_results, [375](#)
- mock_usb_bulk_transfer
 - mock_capture_usb.c, [373](#)
 - mock_dio.c, [375](#)
 - mock_usb_xfers.c, [376](#)
- mock_usb_control_transfer
 - mock_capture_usb.c, [373](#)
 - mock_dio.c, [375](#)
 - mock_usb_xfers.c, [376](#)
- mock_usb_get_config
 - mock_capture_usb.c, [374](#)
- mock_usb_put_config
 - mock_capture_usb.c, [374](#)
- mock_usb_request
 - mock_capture_usb.c, [373](#)
- mock_usb_reset_device
 - mock_capture_usb.c, [373](#)
 - mock_usb_xfers.c, [376](#)
- mock_usb_xfers.c
 - add_devices_fn, [376](#)
 - AddAllACCESUSBDevices, [376](#)
 - direction, [376](#)
 - IO_DIRECTION, [376](#)
 - KEEP, [376](#)
 - mock_USBDeviceFetchADCCConfigBlock, [376](#)
 - mock_USBDevicePutADCCConfigBlock, [376](#)
 - mock_usb_bulk_transfer, [376](#)
 - mock_usb_control_transfer, [376](#)
 - mock_usb_reset_device, [376](#)
- mux_settings, [147](#)
 - ADCCChannelsPerGroup, [147](#)
 - ADCMUXChannels, [147](#)
 - ADCCConfigBlock, [64](#)
 - defined, [147](#)
- mytest.c
 - BITS_PER_BYTE, [404](#)
 - DEVICES_REQUIRED, [404](#)
 - find_dio_96, [404](#)
 - MAKE_MASK, [404](#)
 - MASK_BYTES, [404](#)
 - MAX_DIO_BYTES, [404](#)
 - MAX_NAME_SIZE, [404](#)
 - main, [405](#)
 - PORT_A, [404](#)
 - PORT_B, [404](#)
 - PORT_C, [404](#)
 - show_byte, [404](#)
- NO_CONFIG
 - AIOConfiguration.h, [234](#)
- NO_DEVICE_FOUND
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [408](#)
- NOT_STARTED
 - AIOTypes.h, [293](#)
- NUM_CHANNELS
 - TestCaseSetup, [156](#)
- NUM_PROD_NAMES
 - AIODeviceTable.c, [266](#)
- NUMARGS
 - AIOProductTypes.h, [284](#)
- NUMBER_CHANNELS
 - AIOTypes.h, [291](#)
- Name
 - AIODeviceInfo, [79](#)
 - DeviceProperties, [134](#)
- name
 - ad_gain_pairs, [63](#)
 - AIODeviceQuery, [79](#)
 - AIOUSB::USBDeviceBase, [183](#)
 - DeviceInfo, [133](#)
 - ProductIDName, [152](#)
- NameSize
 - AIODeviceInfo, [79](#)
- nameSize
 - AIODeviceQuery, [79](#)
 - DeviceInfo, [133](#)
 - TestCaseSetup, [156](#)
- new_aio_fifo, [147](#)
 - LOCKING_MECHANISM, [148](#)
 - Pop, [148](#)
 - PopN, [148](#)
 - Push, [148](#)
 - PushN, [148](#)
- NewADCCConfigBlockFromJSON
 - ADCCConfigBlock.c, [211](#)
 - ADCCConfigBlock.h, [214](#)
- NewAIOBuf
 - AIOBuf.c, [215](#)
 - AIOBuf.h, [216](#)
- NewAIOChannelMask
 - AIOChannelMask.c, [217](#)
 - AIOChannelMask.h, [220](#)
- NewAIOChannelMaskFromChr
 - AIOChannelMask.c, [219](#)
 - AIOChannelMask.h, [221](#)
- NewAIOChannelMaskFromStr
 - AIOChannelMask.c, [219](#)
 - AIOChannelMask.h, [220](#)
- NewAIOChannelRangeFromStr
 - AIOChannelRange.c, [223](#)
 - AIOChannelRange.h, [224](#)
- NewAIOCmd
 - AIOCmd.c, [224](#)
 - AIOCmd.h, [225](#)
- NewAIOCmdFromJSON
 - AIOCmd.c, [224](#)
 - AIOCmd.h, [225](#)
- NewAIOCommandLineOptionsFromDefaultOptions
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [230](#)
- NewAIOConfiguration
 - AIOConfiguration.c, [233](#)
 - AIOConfiguration.h, [234](#)
- NewAIOContinuousBuf
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [249](#)
- NewAIOContinuousBufForCounts
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [249](#)

- NewAIOContinuousBufForVolts
 - AIOContinuousBuffer.c, [239](#)
 - AIOContinuousBuffer.h, [249](#)
- NewAIOContinuousBufFromJSON
 - AIOContinuousBuffer.c, [246](#)
 - AIOContinuousBuffer.h, [255](#)
- NewAIOCountsConverter
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [257](#)
- NewAIOCountsConverterFromAIOContinuousBuf
 - AIOCountsConverter.h, [257](#)
- NewAIOCountsConverterWithBuffer
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [257](#)
- NewAIOCountsConverterWithScanLimiter
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [257](#)
- NewAIODeviceInfo
 - AIODeviceInfo.c, [258](#)
 - AIODeviceInfo.h, [259](#)
- NewAIODeviceQuery
 - AIODeviceQuery.c, [260](#)
 - AIODeviceQuery.h, [263](#)
- NewAIOFifo
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [279](#)
- NewAIOFifoAllOrNone
 - AIOFifo.c, [277](#)
- NewAIOFifoTYPE
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
- NewAIOGainRangeFromADCCConfigBlock
 - AIOCountsConverter.c, [256](#)
 - AIOCountsConverter.h, [258](#)
- NewAIOProductGroup
 - AIOProductTypes.c, [283](#)
 - AIOProductTypes.h, [285](#)
- NewAIOProductRange
 - AIOProductTypes.c, [283](#)
 - AIOProductTypes.h, [285](#)
- NewAIOUSBDevice
 - AIOUSBDevice.c, [357](#)
- NewAIOUSBDeviceFromJSON
 - AIOUSBDevice.c, [357](#)
 - AIOUSBDevice.h, [358](#)
- NewCStringArray
 - CStringArray.c, [364](#)
 - CStringArray.h, [365](#)
- NewCStringArrayFromCArgs
 - CStringArray.c, [364](#)
 - CStringArray.h, [365](#)
- NewCStringArrayWithStrings
 - CStringArray.c, [364](#)
 - CStringArray.h, [365](#)
- NewDIOBuf
 - DIOBuf.c, [366](#)
 - DIOBuf.h, [369](#)
- NewDIOBufFromBinStr
 - DIOBuf.c, [366](#)
 - DIOBuf.h, [369](#)
- NewDIOBufFromChar
 - DIOBuf.c, [366](#)
 - DIOBuf.h, [369](#)
- NewDefaultAIOCommandLineOptions
 - AIOCommandLine.c, [227](#)
 - AIOCommandLine.h, [230](#)
- NewUSBDevice
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- NewWDGConfig
 - AIOUSB_WDG.c, [354](#)
 - AIOUSB_WDG.h, [356](#)
- Newintlist
 - AIOList.c, [281](#)
 - AIOList.h, [282](#)
- newone
 - AIOList.c, [281](#)
- next
 - aiobuf_iterator, [73](#)
 - cJSON, [118](#)
- non_usb_supported_device
 - AIOUSB_Properties.c, [351](#)
- num_channels
 - aio_counts_converter, [71](#)
 - AIOCmd, [75](#)
 - AIOCommandLineOptions, [76](#)
 - AIOContinuousBuf, [78](#)
 - opts, [149](#)
- num_oversamples
 - aio_counts_converter, [71](#)
 - AIOCommandLineOptions, [76](#)
 - AIOContinuousBuf, [78](#)
 - opts, [149](#)
- num_samples
 - AIOCmd, [75](#)
- num_scans
 - aio_counts_converter, [71](#)
 - AIOCmd, [75](#)
 - AIOCommandLineOptions, [76](#)
 - AIOContinuousBuf, [78](#)
 - opts, [149](#), [150](#)
- numChannels
 - AIOUSB::AnalogInputSubsystem, [106](#)
 - AIOUSB::AnalogOutputSubsystem, [111](#)
 - AIOUSB::DigitalIIOSubsystem, [140](#)
- numCounterBlocks
 - AIOUSB::CounterSubsystem, [126](#)
- numCounters
 - AIODeviceQuery, [80](#)
 - AIOUSB::CounterSubsystem, [126](#)
 - DeviceInfo, [134](#)
 - TestCaseSetup, [156](#)
- numDIOBytes
 - AIODeviceQuery, [79](#)
 - DeviceInfo, [134](#)
 - TestCaseSetup, [156](#)
- numMUXChannels
 - AIOUSB::AnalogInputSubsystem, [106](#)
- numPorts
 - AIOUSB::DigitalIIOSubsystem, [140](#)
- numTristateChannels
 - AIOUSB::DigitalIIOSubsystem, [140](#)
- numTristateGroups
 - AIOUSB::DigitalIIOSubsystem, [140](#)
- number
 - aio_either_val, [72](#)
- number_arguments
 - AIOArguments, [73](#)
- number_channels
 - config_options, [119](#)
- number_oversamples
 - TestCaseSetup, [156](#)
- number_ranges
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- number_scans
 - configuration, [119](#)
- number_signals
 - AIOChannelMask, [74](#)
- number_to_read
 - AIOContinuousBuffer.c, [243](#)
- OUT
 - mock_capture_usb.c, [373](#)

- mock_dio.c, [374](#), [375](#)
- mock_usb_xfers.c, [376](#)
- OPEN_PATTERN
 - AIOUSB::USBDeviceManager, [189](#)
- object
 - aio_either_val, [72](#)
- old_action
 - daitest.cpp, [387](#)
- open
 - AIOUSB::DIOStreamSubsystem, [144](#)
 - AIOUSB::USBDeviceManager, [187](#)
- openStatus
 - AIOUSB::USBDeviceManager, [189](#)
- OperationFailedException, [148](#)
 - AIOUSB::OperationFailedException, [148](#)
- operator<<
 - AIOUSB, [62](#)
- opterr
 - AIOCommandLine.c, [228](#)
- optind
 - AIOCommandLine.c, [228](#)
- options, [148](#)
 - maxcount, [149](#)
 - use_maxcount, [149](#)
- opts, [149](#)
 - adcconfig_json, [150](#)
 - aiobuf_json, [150](#)
 - block_size, [150](#)
 - buffer_size, [150](#)
 - cal_channel, [150](#)
 - calibration, [150](#)
 - calibration_enabled, [150](#)
 - clock_rate, [150](#)
 - clock_scale, [150](#)
 - clock_speed, [150](#)
 - counts, [150](#)
 - debug_level, [150](#)
 - default_aiobuf_json, [150](#)
 - default_clock_rate, [150](#)
 - default_end_channel, [150](#)
 - default_num_channels, [149](#)
 - default_num_oversamples, [150](#)
 - default_num_scans, [149](#)
 - default_start_channel, [150](#)
 - end_channel, [150](#)
 - gain_code, [150](#)
 - index, [150](#)
 - max_channels, [150](#)
 - num_channels, [149](#)
 - num_oversamples, [149](#)
 - num_scans, [149](#), [150](#)
 - number_ranges, [150](#)
 - outfile, [150](#)
 - physical, [150](#)
 - ranges, [150](#)
 - rate_limit, [150](#)
 - repeat, [150](#)
 - reset, [150](#)
 - slow_acquire, [150](#)
 - start_channel, [150](#)
 - verbose, [150](#)
 - with_timing, [150](#)
- orig_usb_bulk_transfer
 - mock_capture_usb.c, [374](#)
- orig_usb_control_transfer
 - mock_capture_usb.c, [374](#)
- orig_usb_get_config
 - mock_capture_usb.c, [374](#)
- orig_usb_put_config
 - mock_capture_usb.c, [374](#)
- orig_usb_request
 - mock_capture_usb.c, [374](#)
- orig_usb_reset_device
 - mock_capture_usb.c, [374](#)
- origconf
 - USBDevice, [179](#)
- os_count
 - aio_counts_converter, [71](#)
- outfile
 - AIOCommandLineOptions, [76](#)
 - AIOUSB_Log.c, [348](#)
 - AIOUSB_Log.h, [350](#)
 - mock_capture_usb.c, [374](#)
 - opts, [150](#)
- output_file
 - configuration, [120](#)
- outputMask
 - DeviceInfo, [133](#)
- outputRange
 - AIOUSB::AO16_AnalogOutputSubsystem, [116](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- OutputVoltagePoint, [151](#)
 - AIOUSB::OutputVoltagePoint, [151](#)
- OutputVoltagePointArray, [151](#)
 - AIOUSB::OutputVoltagePointArray, [152](#)
- overSample
 - AIOUSB::AI16_DataSet, [69](#)
 - AIOUSB::AnalogInputSubsystem, [106](#)
- PICO_DIO16RO8
 - AIOTypes.h, [295](#)
- pBuf
 - BulkAcquireWorkerParams, [117](#)
- PID
 - AIODeviceInfo, [79](#)
 - AIOUSBDevice, [83](#)
- PORT_A
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [406](#)
- PORT_B
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [405](#)
- PORT_C
 - dio96_read_write.c, [402](#)
 - mytest.c, [404](#)
 - tmp.c, [405](#)
- PRIVATE
 - AIOUSB_Core.h, [333](#)
- PROD_NAME_SIZE
 - AIOUSB_Core.h, [333](#)
- PUBLIC_EXTERN
 - AIOUSB_Core.h, [333](#)
- parent
 - AIOUSB::Counter, [123](#)
 - AIOUSB::DeviceSubsystem, [136](#)
- pass_through
 - AIOCommandLineOptions, [76](#)
- PendingDACData
 - AIOUSBDevice, [84](#)
- perftest.c
 - main, [410](#)
- physical
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- points
 - AIOUSB::AI16_DataSet, [69](#)
- Pop
 - AIOFifo.c, [277](#)
 - new_aio_fifo, [148](#)
- PopN
 - AIOContinuousBuf, [78](#)
 - AIOFifo.c, [277](#)

- AIOFifo.h, [280](#)
 - new_ao_fifo, [148](#)
- pos
 - AIOChannelMask, [74](#)
- pow_of_minsize
 - AIOUSB_DIO.c, [345](#)
- prev
 - cJSON, [118](#)
- print
 - AIOUSB::AI16_DataSet, [69](#)
 - AIOUSB::AnalogInputSubsystem, [89](#)
 - AIOUSB::AnalogOutputSubsystem, [110](#)
 - AIOUSB::CounterSubsystem, [124](#)
 - AIOUSB::DeviceSubsystem, [136](#)
 - AIOUSB::DigitalIOSubsystem, [137](#)
 - AIOUSB::DIOSStreamSubsystem, [142](#)
 - AIOUSB::USB_AI16_Family, [158](#)
 - AIOUSB::USB_AIO16_Family, [161](#)
 - AIOUSB::USB_AO16_Family, [164](#)
 - AIOUSB::USB_CTR_15_Family, [166](#)
 - AIOUSB::USB_DA12_8A_Family, [168](#)
 - AIOUSB::USB_DA12_8E_Family, [171](#)
 - AIOUSB::USB_DIO_16_Family, [173](#)
 - AIOUSB::USB_DIO_32_Family, [175](#)
 - AIOUSB::USB_DIO_Family, [177](#)
 - AIOUSB::USBDeviceBase, [181](#)
 - AIOUSB::USBDeviceManager, [185](#)
- print_ao_usage
 - aiocommon.c, [380](#)
 - aiocommon.h, [381](#)
- print_usage
 - bulk_acquire_sample.c, [384](#)
 - slow_receiver_test.cpp, [390](#)
- printDevices
 - AIOUSB::USBDeviceManager, [185](#)
- process_ao_cmd_line
 - aiocommon.c, [380](#)
 - aiocommon.h, [381](#)
- process_cmd_line
 - bulk_acquire_sample.c, [384](#)
 - continuous_mode_callback.c, [386](#)
 - continuous_mode_from_json_config.c, [387](#)
 - dirctest.c, [389](#)
 - simple_continuous_with_json.c, [395](#)
 - slow_receiver_test.cpp, [390](#)
 - start_stop_continuous.c, [396](#)
 - test.c, [391](#)
- ProductIDS_begin
 - AIOTypes.h, [294](#)
- ProductIDS_end
 - AIOTypes.h, [296](#)
- ProductID
 - AIOUSBDevice, [83](#)
 - DeviceProperties, [134](#)
- productID
 - AIODeviceQuery, [79](#)
 - AIOUSB::USBDeviceBase, [183](#)
 - DeviceInfo, [133](#)
 - TestCaseSetup, [156](#)
- ProductIDName, [152](#)
 - id, [152](#)
 - name, [152](#)
- productIDNameTable
 - AIOUSB_Core.c, [330](#)
- ProductIDS
 - AIOTypes.h, [294](#)
- ProductIDToName
 - AIODeviceTable.c, [266](#)
 - AIODeviceTable.h, [271](#)
- productIDToName
 - AIOUSB::USBDeviceManager, [185](#), [186](#)
- ProductNameToID
 - AIODeviceTable.c, [266](#)
 - AIODeviceTable.h, [271](#)
- productNameToID
 - AIOUSB::USBDeviceManager, [186](#), [187](#)
- Push
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
 - new_ao_fifo, [148](#)
- PushN
 - AIOContinuousBuf, [78](#)
 - AIOFifo.c, [277](#)
 - AIOFifo.h, [280](#)
 - new_ao_fifo, [148](#)
- QueryDeviceInfo
 - AIODeviceTable.c, [266](#)
 - AIODeviceTable.h, [270](#)
- RUNNING
 - AIOTypes.h, [293](#)
- RUNNING_OR_WITH_DATA
 - AIOTypes.h, [293](#)
- RANGE_0_10V
 - AIOUSB::AnalogInputSubsystem, [104](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- RANGE_0_1V
 - AIOUSB::AnalogInputSubsystem, [104](#)
- RANGE_0_2_5V
 - AIOUSB::DA12_AnalogOutputSubsystem, [131](#)
- RANGE_0_2V
 - AIOUSB::AnalogInputSubsystem, [104](#)
- RANGE_0_5V
 - AIOUSB::AnalogInputSubsystem, [104](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- RANGE_10V
 - AIOUSB::AnalogInputSubsystem, [104](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- RANGE_1V
 - AIOUSB::AnalogInputSubsystem, [105](#)
- RANGE_2_5V
 - AIOUSB::DA12_AnalogOutputSubsystem, [131](#)
- RANGE_2V
 - AIOUSB::AnalogInputSubsystem, [104](#)
- RANGE_5V
 - AIOUSB::AnalogInputSubsystem, [104](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- RANGE_TEXT
 - AIOUSB::AnalogInputSubsystem, [105](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [132](#)
- RATE_LIMIT
 - idio_sample.c, [409](#)
 - idio_sample2.c, [409](#)
 - iio_sample.c, [411](#)
- RED
 - AIOUSB_Log.h, [349](#)
- REG_CAL_MODE
 - AIOUSB::AnalogInputSubsystem, [105](#)
- REG_GAIN_CODE
 - AIOUSB::AnalogInputSubsystem, [105](#)
- REG_OVERSAMPLE
 - AIOUSB::AnalogInputSubsystem, [105](#)
- REG_START_END
 - AIOUSB::AnalogInputSubsystem, [105](#)
- REG_TRIG_MODE
 - AIOUSB::AnalogInputSubsystem, [105](#)
- RELEASE_RESOURCE
 - AIOFifo.h, [278](#)
- REPEAT

- AIOCommandLine.h, [229](#)
 - aiocommon.c, [380](#)
- ROOTCLOCK
 - AIOContinuousBuffer.h, [249](#)
 - AIOTypes.h, [293](#)
- range
 - ADRange, [64](#)
 - AIOUSB::AI16_DataPoint, [66](#)
 - AIOUSB::AnalogIORange, [109](#)
- rangeCounts
 - AIOUSB::AnalogIORange, [109](#)
- RangeShift
 - AIOUSBDevice, [84](#)
- RangeValueLookup
 - AIOContinuousBuffer.c, [238](#)
- rangeVolts
 - AIOUSB::AnalogIORange, [109](#)
- rangelookup, [152](#)
 - maxvalue, [152](#)
 - minvalue, [152](#)
- ranges
 - AIOCommandLineOptions, [77](#)
 - opts, [150](#)
- rate_limit
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- RawCountsWorkFunction
 - AIOContinuousBuffer.c, [238](#)
- rdelta
 - AIOFifo.c, [276](#)
- read
 - AIOUSB::AnalogInputSubsystem, [99](#)
 - AIOUSB::DigitalIOSubsystem, [139](#)
 - AIOUSB::DIOSStreamSubsystem, [144](#)
- read_and_write.c
 - CHECK_RESULT, [402](#)
 - find_dio, [402](#)
 - main, [402](#)
- read_channels_test.c
 - find_ai_board, [393](#)
 - main, [393](#)
- read_channels_with_getchannelv_test.cpp
 - get_options, [393](#)
 - main, [393](#)
- readBuffer
 - DeviceInfo, [133](#)
- readBulkBuffer
 - AIOUSB::AnalogInputSubsystem, [106](#)
- readBulkNext
 - AIOUSB::AnalogInputSubsystem, [102](#)
- readBulkSamplesAvailable
 - AIOUSB::AnalogInputSubsystem, [101](#)
- readBulkSamplesRequested
 - AIOUSB::AnalogInputSubsystem, [106](#)
- readBulkSamplesRetrieved
 - AIOUSB::AnalogInputSubsystem, [106](#)
- readBulkStart
 - AIOUSB::AnalogInputSubsystem, [101](#)
- readConfig
 - AIOUSB::AnalogInputSubsystem, [91](#)
- ReadConfigBlock
 - AIOUSB_ADC.c, [305](#)
 - AIOUSB_ADC.h, [324](#)
- readCount
 - AIOUSB::Counter, [122](#)
- readCountAndSetModeAndCount
 - AIOUSB::Counter, [122](#)
- readCountAndStatus
 - AIOUSB::Counter, [122](#)
- readCounts
 - AIOUSB::AnalogInputSubsystem, [99](#)
 - AIOUSB::CounterSubsystem, [125](#)
- readVolts
 - AIOUSB::AnalogInputSubsystem, [101](#)
- receiver.cpp
 - main, [398](#)
- registers
 - ADConfigBlock, [64](#)
- repeat
 - opts, [150](#)
- repeat_number
 - AIOCommandLineOptions, [76](#)
- reset
 - AIOCommandLineOptions, [76](#)
 - AIOUSB::USBDeviceBase, [182](#)
 - opts, [150](#)
- resetCPU
 - TestCaseSetup, [156](#)
- ResetCounters
 - AIOContinuousBuffer.c, [243](#)
- ResolveDeviceIndex
 - AIOUSB_Core.c, [329](#)
- ResultCode_begin
 - AIOTypes.h, [297](#)
- ResultCode_end
 - AIOTypes.h, [298](#)
- ResultCode
 - AIOTypes.h, [297](#)
- retval
 - aioerror, [80](#)
- reverse_cal_table.cpp
 - goDolt, [390](#)
 - main, [390](#)
- right
 - aio_ret_value, [72](#)
- RootClock
 - AIOUSBDevice, [83](#)
 - DeviceProperties, [135](#)
- run
 - configuration, [120](#)
- run_acquisition
 - start_stop_continuous.c, [396](#)
- SAMPLE_OPT
 - AIOCommandLine.h, [229](#)
- SETCAL_OPT
 - AIOCommandLine.h, [229](#)
- START_CHANNEL
 - AIOChannelRange.c, [222](#)
- SUCCESS
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [408](#)
- STATE
 - AIOChannelRange.c, [222](#)
- STRING_ARRAY
 - CStringArray.h, [365](#)
- sample3.c
 - main, [401](#)
- sample_dio.c
 - find_ai_board, [390](#)
 - main, [390](#)
- SampleClass.cpp
 - main, [397](#)
- samples/TestLib/TestCaseSetup.cpp, [382](#)
- samples/TestLib/TestCaseSetup.h, [382](#)
- samples/TestLib/aiocommon.c, [380](#)
- samples/TestLib/aiocommon.h, [381](#)
- samples/USB-AI16-16/HOLD/dirktest.c, [388](#)
- samples/USB-AI16-16/HOLD/julian_test.c, [389](#)
- samples/USB-AI16-16/HOLD/reverse_cal_table.cpp, [389](#)
- samples/USB-AI16-16/HOLD/sample_dio.c, [390](#)
- samples/USB-AI16-16/HOLD/slow_receiver_test.cpp, [390](#)
- samples/USB-AI16-16/HOLD/test.c, [391](#)
- samples/USB-AI16-16/README.doc, [200](#)

- samples/USB-AI16-16/android/README.md, [411](#)
- samples/USB-AI16-16/android/read_channels_test/README.md, [411](#)
- samples/USB-AI16-16/bulk_acquire_sample.c, [384](#)
- samples/USB-AI16-16/burst_test.c, [384](#)
- samples/USB-AI16-16/continuous_mode.c, [385](#)
- samples/USB-AI16-16/continuous_mode_callback.c, [385](#)
- samples/USB-AI16-16/continuous_mode_from_json_config.c, [386](#)
- samples/USB-AI16-16/daitest.cpp, [387](#)
- samples/USB-AI16-16/dio_sample.c, [387](#)
- samples/USB-AI16-16/diotest.c, [388](#)
- samples/USB-AI16-16/diotest2.cpp, [388](#)
- samples/USB-AI16-16/java/README.md, [411](#)
- samples/USB-AI16-16/java/extcal/README.md, [411](#)
- samples/USB-AI16-16/java/extcal/src/main/java/com/acesio/extcal/AIOContinuousBuffer.c, [243](#)
- samples/USB-AI16-16/java/extcal/src/main/java/com/acesio/extcal/AIOContinuousBuffer.cpp, [391](#)
- samples/USB-AI16-16/java/read_channels_test/README.md, [411](#)
- samples/USB-AI16-16/java/read_channels_test/native-utils/README.md, [411](#)
- samples/USB-AI16-16/jni/read_channels_test.c, [392](#)
- samples/USB-AI16-16/read_channels_test.c, [393](#)
- samples/USB-AI16-16/read_channels_with_getchannelv_test.cpp, [393](#)
- samples/USB-AI16-16/reverse_cal_table.cpp, [389](#)
- samples/USB-AI16-16/sample.cpp, [394](#)
- samples/USB-AI16-16/simp_test.cpp, [395](#)
- samples/USB-AI16-16/simple_continuous_with_json.c, [395](#)
- samples/USB-AI16-16/start_stop_continuous.c, [396](#)
- samples/USB-AI16-16/test_fastscan.cpp, [396](#)
- samples/USB-AO16-16/README.doc, [200](#)
- samples/USB-AO16-16/sample.cpp, [394](#)
- samples/USB-ARB1/stream_test.c, [397](#)
- samples/USB-DA12-8A/README.doc, [200](#)
- samples/USB-DA12-8A/SampleClass.cpp, [397](#)
- samples/USB-DA12-8A/sample.cpp, [394](#)
- samples/USB-DIO-16/README.doc, [200](#)
- samples/USB-DIO-16/receiver.cpp, [397](#)
- samples/USB-DIO-16/sample.cpp, [394](#)
- samples/USB-DIO-16/standalone_receiver.c, [398](#)
- samples/USB-DIO-32/README.doc, [200](#)
- samples/USB-DIO-32/daisample.c, [398](#)
- samples/USB-DIO-32/read_and_write_sample.c, [398](#)
- samples/USB-DIO-32/sample3.c, [400](#)
- samples/USB-DIO-48/read_and_write.c, [401](#)
- samples/USB-DIO-96/dio96_read_write.c, [402](#)
- samples/USB-DIO-96/mytest.c, [403](#)
- samples/USB-DIO-96/read_and_write_sample.c, [399](#)
- samples/USB-DIO-96/tmp.c, [405](#)
- samples/USB-DIO-96/write_sample.c, [406](#)
- samples/USB-IDIO-16_8/README.doc, [200](#)
- samples/USB-IDIO-16_8/idio_sample.c, [408](#)
- samples/USB-IDIO-16_8/idio_sample2.c, [409](#)
- samples/USB-IDIO-16_8/perftest.c, [410](#)
- samples/USB-IIRO-16_8/README.doc, [200](#)
- samples/USB-IIRO-16_8/iirro_sample.c, [410](#)
- save_results
 - mock_dio.c, [375](#)
- scan_count
 - aio_counts_converter, [71](#)
- scan_type
 - configuration, [120](#)
- scanForDevices
 - AIOUSB::USBDeviceManager, [188](#)
- scans_read
 - AIOContinuousBuf, [78](#)
- selectGate
 - AIOUSB::CounterSubsystem, [125](#)
- SerialNumber
 - DeviceProperties, [134](#)
- serialNumber
 - AIOUSB::USBDeviceBase, [183](#)
 - DeviceInfo, [134](#)
- setAutoConfig
 - AIOUSB::AnalogInputSubsystem, [91](#)
- setCalMode
 - AIOUSB::AnalogInputSubsystem, [92](#)
- setCalibrationTable
 - AIOUSB::AnalogInputSubsystem, [97](#)
- setClock
 - AIOUSB::AnalogInputSubsystem, [98](#)
 - AIOUSB::DIOStreamSubsystem, [143](#)
- setCommTimeout
 - AIOUSB::USBDeviceBase, [182](#)
- SetConfig
 - AIOContinuousBuffer.c, [243](#)
- setCount
 - AIOUSB::Counter, [121](#)
- setCountRange
 - AIOUSB::AnalogIORange, [107](#)
- setCurrentDeviceIndex
 - TestCaseSetup, [154](#)
- setDifferentialMode
 - AIOUSB::AnalogInputSubsystem, [95](#)
- setDiscardFirstSample
 - AIOUSB::AnalogInputSubsystem, [92](#)
- setMaxCount
 - TestCaseSetup, [155](#)
- setMiscClock
 - AIOUSB::USBDeviceBase, [181](#)
- setMode
 - AIOUSB::Counter, [121](#)
- setModeAndCount
 - AIOUSB::Counter, [121](#)
- setOverSample
 - AIOUSB::AnalogInputSubsystem, [96](#)
- setRange
 - AIOUSB::AI16_InputRange, [70](#)
 - AIOUSB::AnalogInputSubsystem, [94](#)
 - AIOUSB::AnalogIORange, [107](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [113](#)
 - AIOUSB::AO16_OutputRange, [116](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [129](#)
 - AIOUSB::DA12_OutputRange, [133](#)
- setRangeAndDiffMode
 - AIOUSB::AnalogInputSubsystem, [96](#)
- setScanRange
 - AIOUSB::AnalogInputSubsystem, [89](#)
- setStreamingBlockSize
 - AIOUSB::AnalogInputSubsystem, [97](#)
 - AIOUSB::DIOStreamSubsystem, [143](#)
 - AIOUSB::USBDeviceBase, [181](#)
- setTriggerMode
 - AIOUSB::AnalogInputSubsystem, [93](#)
- setVoltRange
 - AIOUSB::AnalogIORange, [109](#)
- setupVoltageParameters
 - TestCaseSetup, [156](#)
- show_byte
 - dio96_read_write.c, [403](#)
 - mytest.c, [404](#)
 - tmp.c, [406](#)
- signal_index
 - AIOChannelMask, [74](#)
- signal_indices
 - AIOChannelMask, [74](#)
- signals
 - AIOChannelMask, [74](#)
- simp_test.cpp
 - main, [395](#)
- simple_continuous_with_json.c
 - fn, [396](#)

- fp, [396](#)
- get_channel_range, [395](#)
- main, [396](#)
- process_cmd_line, [395](#)
- size
 - ADCConfigBlock, [63](#)
 - aio_ret_value, [72](#)
 - AIOArgument, [72](#)
 - AIOBuf, [73](#)
 - AIOChannelMask, [74](#)
 - AIOContinuousBuf, [78](#)
 - DIOBuf, [141](#)
 - ushort_array, [190](#)
- slow_acquire
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- slow_receiver_test.cpp
 - main, [390](#)
 - print_usage, [390](#)
 - process_cmd_line, [390](#)
- standalone_receiver.c
 - main, [398](#)
- start
 - aio_channel_range, [70](#)
- start_channel
 - AIOChannelRangeTmp, [74](#)
 - AIOCommandLineOptions, [76](#)
 - channel_range, [118](#)
 - opts, [150](#)
- start_scanning
 - AIOContinuousBuf, [78](#)
- start_stop_continuous.c
 - fnd, [396](#)
 - fp, [396](#)
 - get_channel_range, [396](#)
 - main, [396](#)
 - process_cmd_line, [396](#)
 - run_acquisition, [396](#)
- startChannel
 - AIOUSB::AnalogInputSubsystem, [106](#)
- startClock
 - AIOUSB::CounterSubsystem, [125](#)
- StartStreaming
 - AIOContinuousBuffer.c, [243](#)
- status
 - AIOContinuousBuf, [78](#)
- std::vector< AI16_DataPoint >
 - AIOUSB::AI16_DataPoint, [66](#)
- stop_scan
 - AIOCmd, [75](#)
- stop_scan_arg
 - AIOCmd, [75](#)
- stopClock
 - AIOUSB::CounterSubsystem, [126](#)
 - AIOUSB::DIOStreamSubsystem, [144](#)
- str
 - lookup, [147](#)
- strbuf
 - DIOBuf, [141](#)
- strbuf_size
 - DIOBuf, [141](#)
- stream_test.c
 - find_ai_board, [397](#)
 - main, [397](#)
- StreamingBlockSize
 - AIOUSBDevice, [84](#)
- string
 - cJSON, [118](#)
- StringArray, [152](#)
 - AIOUSB::StringArray, [152](#)
- strrep
 - AIOChannelMask, [74](#)
- strrepsmall
 - AIOChannelMask, [74](#)
- strvalue
 - lookup, [147](#)
- subsystem
 - AIOUSB::AI16_DataSet, [69](#)
- sum
 - aio_counts_converter, [71](#)
- supportedProductIDs
 - AIOUSB::USB_AI16_Family, [159](#)
 - AIOUSB::USB_AIO16_Family, [162](#)
 - AIOUSB::USB_AO16_Family, [164](#)
 - AIOUSB::USB_CTR_15_Family, [167](#)
 - AIOUSB::USB_DA12_8A_Family, [169](#)
 - AIOUSB::USB_DA12_8E_Family, [171](#)
 - AIOUSB::USB_DIO_16_Family, [173](#)
 - AIOUSB::USB_DIO_32_Family, [176](#)
 - AIOUSB::USB_DIO_Family, [178](#)
- TERMINATED
 - AIOTypes.h, [293](#)
- TERMINATED_OVERRUN
 - AIOTypes.h, [293](#)
- TERMINATING
 - AIOTypes.h, [293](#)
- TERSE
 - AIOUSB_Properties.h, [352](#)
- THREAD_STATUS_begin
 - AIOTypes.h, [293](#)
- THREAD_STATUS_end
 - AIOTypes.h, [293](#)
- TIMEOUT_OPT
 - AIOCommandLine.h, [229](#)
- TAIL_Q_LIST
 - AIOList.h, [281](#), [282](#)
- TAIL_Q_LIST_TYPE
 - AIOList.h, [281](#), [282](#)
- TERSE_LOGGING
 - TestCaseSetup.h, [383](#)
- TEST_ADC_BulkPoll
 - TestCaseSetup, [156](#)
- THREAD_STATUS
 - AIOTypes.h, [293](#)
- THROW_ERROR
 - TestCaseSetup.h, [383](#)
- THROW_IF_ERROR
 - TestCaseSetup, [155](#)
- TRACE
 - TestCaseSetup.h, [383](#)
- TRACE_LEVEL
 - TestCaseSetup.h, [383](#)
- TRIG_MODE_SCAN
 - AIOUSB::AnalogInputSubsystem, [104](#)
- TRIG_MODE_TIMER
 - AIOUSB::AnalogInputSubsystem, [104](#)
- TYPE
 - AIOFifo.h, [279](#)
- targetSerialNumber
 - config_options, [119](#)
- tattr
 - AIOContinuousBuf, [78](#)
- test.c
 - capture_data, [391](#)
 - fnd, [391](#)
 - fp, [391](#)
 - get_channel_range, [391](#)
 - main, [391](#)
 - process_cmd_line, [391](#)
- test_fastscan.cpp
 - main, [397](#)
- TestCaseSetup, [153](#)
 - ~TestCaseSetup, [154](#)

- block_size, 156
- CAL_CHANNEL, 156
- calibration_enabled, 156
- clock_speed, 156
- configBlock, 156
- counts, 156
- dataBuf, 156
- deviceFound, 156
- DeviceIndex, 156
- doBulkAcquire, 155
- doBulkConfigBlock, 154
- doCSVReadVoltages, 155
- doCSVWithGetChannelV, 155
- doCleanupAfterBulk, 155
- doDACDirect, 155
- doDACDirectSetup, 155
- doDemonstrateReadVoltages, 154
- doDisplayBulkResults, 156
- doFastITScan, 155
- doFastITScanSetup, 155
- doGenericVendorWrite, 155
- doGetBuffer, 155
- doPreReadImmediateVoltages, 155
- doPreSetup, 154
- doScanSingleChannel, 155
- doSetAutoCalibration, 154
- doSomething, 154
- doTestSetAutoCalibration, 155
- doVerifyGroundCalibration, 154
- doVerifyReferenceCalibration, 154
- envGetDouble, 155
- envGetInteger, 155
- findDevice, 154
- gainCodes, 156
- getCounts, 156
- getGainCodes, 156
- getVolts, 156
- MAX_CHANNELS, 156
- maxcounts, 156
- NUM_CHANNELS, 156
- nameSize, 156
- numCounters, 156
- numDIOBytes, 156
- number_oversamples, 156
- productId, 156
- resetCPU, 156
- setCurrentDeviceIndex, 154
- setMaxCount, 155
- setupVoltageParameters, 156
- TEST_ADC_BulkPoll, 156
- THROW_IF_ERROR, 155
- TestCaseSetup, 154
- TestCaseSetup, 154
- ThrowError, 155
- volts, 156
- writeBuffer, 155
- TestCaseSetup.h
 - ALERT_LEVEL, 383
 - CHECK_RESULT, 383
 - DEBUG, 383
 - DEBUG_LEVEL, 383
 - DEF_CAL_CHANNEL, 384
 - DEF_MAX_CHANNELS, 383
 - DEF_NUM_CHANNELS, 384
 - ERROR, 383
 - ERROR_LEVEL, 383
 - FATAL, 383
 - FATAL_LEVEL, 383
 - INFO, 383
 - INFO_LEVEL, 383
 - LOG, 383
 - MAX_NAME_SIZE, 383
 - TERSE_LOGGING, 383
 - THROW_ERROR, 383
 - TRACE, 383
 - TRACE_LEVEL, 383
 - VERBOSE_LOGGING, 383
 - WARN_LEVEL, 383
- testing
 - ADCCfgBlock, 64
 - AIOContinuousBuf, 78
 - AIOUSBDevice, 85
- threaded
 - AIOArgument, 72
- ThrowError
 - TestCaseSetup, 155
- timeStamp
 - AIOUSB::AI16_DataSet, 69
- timeout
 - ADCCfgBlock, 64
 - AIOContinuousBuf, 78
 - AIOWDGConfig, 86
 - configuration, 119
 - USBDevice, 179
- tmp.c
 - BITS_PER_BYTE, 405
 - DEVICES_REQUIRED, 405
 - find_dio_96, 406
 - MAKE_MASK, 406
 - MASK_BYTES, 405
 - MAX_DIO_BYTES, 405
 - MAX_NAME_SIZE, 405
 - main, 406
 - PORT_A, 406
 - PORT_B, 405
 - PORT_C, 405
 - show_byte, 406
- toString
 - AIOUSB::AI16_DataPoint, 66
- triggerMode
 - AIOUSB::AI16_DataSet, 69
 - AIOUSB::AnalogInputSubsystem, 106
- Tristates
 - AIOUSBDevice, 83
 - DeviceProperties, 135
- TrueFalse
 - AIOContinuousBuffer.c, 246
- type
 - aio_ret_value, 72
 - AIOBuf, 73
 - AIOContinuousBuf, 78
 - cJSON, 118
 - configuration, 119
- USB-DIO-96/read_and_write_sample.c
 - NO_DEVICE_FOUND, 400
 - SUCCESS, 400
 - USB_ERROR, 400
- USB_AI12_128
 - AIOTypes.h, 295
- USB_AI12_128A
 - AIOTypes.h, 295
- USB_AI12_128E
 - AIOTypes.h, 295
- USB_AI12_16
 - AIOTypes.h, 295
- USB_AI12_16A
 - AIOTypes.h, 295
- USB_AI12_16E
 - AIOTypes.h, 295
- USB_AI12_32
 - AIOTypes.h, 295
- USB_AI12_32A
 - AIOTypes.h, 295

USB_AI12_32E
 AIOTypes.h, [295](#)
 USB_AI12_64
 AIOTypes.h, [295](#)
 USB_AI12_64A
 AIOTypes.h, [295](#)
 USB_AI12_64E
 AIOTypes.h, [295](#)
 USB_AI12_64M
 AIOTypes.h, [295](#)
 USB_AI12_64MA
 AIOTypes.h, [295](#)
 USB_AI12_64ME
 AIOTypes.h, [295](#)
 USB_AI12_96
 AIOTypes.h, [295](#)
 USB_AI12_96A
 AIOTypes.h, [295](#)
 USB_AI12_96E
 AIOTypes.h, [295](#)
 USB_AI16_128A
 AIOTypes.h, [295](#)
 USB_AI16_128E
 AIOTypes.h, [295](#)
 USB_AI16_16A
 AIOTypes.h, [295](#)
 USB_AI16_16E
 AIOTypes.h, [295](#)
 USB_AI16_32A
 AIOTypes.h, [295](#)
 USB_AI16_32E
 AIOTypes.h, [295](#)
 USB_AI16_64A
 AIOTypes.h, [295](#)
 USB_AI16_64E
 AIOTypes.h, [295](#)
 USB_AI16_64MA
 AIOTypes.h, [295](#)
 USB_AI16_64ME
 AIOTypes.h, [295](#)
 USB_AI16_96A
 AIOTypes.h, [295](#)
 USB_AI16_96E
 AIOTypes.h, [295](#)
 USB_AIO12_128
 AIOTypes.h, [296](#)
 USB_AIO12_128A
 AIOTypes.h, [296](#)
 USB_AIO12_128E
 AIOTypes.h, [296](#)
 USB_AIO12_16
 AIOTypes.h, [296](#)
 USB_AIO12_16A
 AIOTypes.h, [296](#)
 USB_AIO12_16E
 AIOTypes.h, [296](#)
 USB_AIO12_32
 AIOTypes.h, [296](#)
 USB_AIO12_32A
 AIOTypes.h, [296](#)
 USB_AIO12_32E
 AIOTypes.h, [296](#)
 USB_AIO12_64
 AIOTypes.h, [296](#)
 USB_AIO12_64A
 AIOTypes.h, [296](#)
 USB_AIO12_64E
 AIOTypes.h, [296](#)
 USB_AIO12_64M
 AIOTypes.h, [296](#)
 USB_AIO12_64MA
 AIOTypes.h, [296](#)
 USB_AIO12_64ME
 AIOTypes.h, [296](#)
 USB_AIO12_96
 AIOTypes.h, [296](#)
 USB_AIO12_96A
 AIOTypes.h, [296](#)
 USB_AIO12_96E
 AIOTypes.h, [296](#)
 USB_AIO16_128A
 AIOTypes.h, [296](#)
 USB_AIO16_128E
 AIOTypes.h, [296](#)
 USB_AIO16_16A
 AIOTypes.h, [296](#)
 USB_AIO16_16E
 AIOTypes.h, [296](#)
 USB_AIO16_32A
 AIOTypes.h, [296](#)
 USB_AIO16_32E
 AIOTypes.h, [296](#)
 USB_AIO16_64A
 AIOTypes.h, [296](#)
 USB_AIO16_64E
 AIOTypes.h, [296](#)
 USB_AIO16_64MA
 AIOTypes.h, [296](#)
 USB_AIO16_64ME
 AIOTypes.h, [296](#)
 USB_AIO16_96A
 AIOTypes.h, [296](#)
 USB_AIO16_96E
 AIOTypes.h, [296](#)
 USB_AO12_12
 AIOTypes.h, [295](#)
 USB_AO12_12A
 AIOTypes.h, [295](#)
 USB_AO12_16
 AIOTypes.h, [295](#)
 USB_AO12_16A
 AIOTypes.h, [295](#)
 USB_AO12_4
 AIOTypes.h, [295](#)
 USB_AO12_4A
 AIOTypes.h, [295](#)
 USB_AO12_8
 AIOTypes.h, [295](#)
 USB_AO12_8A
 AIOTypes.h, [295](#)
 USB_AO16_12
 AIOTypes.h, [295](#)
 USB_AO16_12A
 AIOTypes.h, [295](#)
 USB_AO16_16
 AIOTypes.h, [295](#)
 USB_AO16_16A
 AIOTypes.h, [295](#)
 USB_AO16_4
 AIOTypes.h, [295](#)
 USB_AO16_4A
 AIOTypes.h, [295](#)
 USB_AO16_8
 AIOTypes.h, [295](#)
 USB_AO16_8A
 AIOTypes.h, [295](#)
 USB_AO_ARB1
 AIOTypes.h, [295](#)
 USB_BULK_READ_ENDPOINT
 AIOTypes.h, [301](#)
 USB_BULK_WRITE_ENDPOINT
 AIOTypes.h, [301](#)
 USB_CTR_15
 AIOTypes.h, [294](#)

- USB_DA12_8A
 - AIOTypes.h, [294](#)
- USB_DA12_8A_REV_A
 - AIOTypes.h, [294](#)
- USB_DA12_8E
 - AIOTypes.h, [294](#)
- USB_DI16A
 - AIOTypes.h, [294](#)
- USB_DI16A_REV_A1
 - AIOTypes.h, [294](#)
- USB_DI16A_REV_A2
 - AIOTypes.h, [294](#)
- USB_DIO16RO8
 - AIOTypes.h, [295](#)
- USB_DIO24_CTR6
 - AIOTypes.h, [294](#)
- USB_DIO24DO12
 - AIOTypes.h, [295](#)
- USB_DIO48DO24
 - AIOTypes.h, [295](#)
- USB_DIO_16A
 - AIOTypes.h, [294](#)
- USB_DIO_16H
 - AIOTypes.h, [294](#)
- USB_DIO_32
 - AIOTypes.h, [294](#)
- USB_DIO_32I
 - AIOTypes.h, [294](#)
- USB_DIO_48
 - AIOTypes.h, [294](#)
- USB_DIO_96
 - AIOTypes.h, [294](#)
- USB_DO16A
 - AIOTypes.h, [294](#)
- USB_DO16A_REV_A1
 - AIOTypes.h, [294](#)
- USB_DO24
 - AIOTypes.h, [295](#)
- USB_ERROR
 - USB-DIO-96/read_and_write_sample.c, [400](#)
 - write_sample.c, [408](#)
- USB_IDIO_16
 - AIOTypes.h, [294](#)
- USB_IDIO_4
 - AIOTypes.h, [294](#)
- USB_IDIO_8
 - AIOTypes.h, [294](#)
- USB_IDO_16
 - AIOTypes.h, [294](#)
- USB_II_16
 - AIOTypes.h, [294](#)
- USB_II_16_OLD
 - AIOTypes.h, [294](#)
- USB_II_8
 - AIOTypes.h, [294](#)
- USB_II_8_OLD
 - AIOTypes.h, [294](#)
- USB_IIRO4_2SM
 - AIOTypes.h, [295](#)
- USB_IIRO4_COM
 - AIOTypes.h, [295](#)
- USB_IIRO_16
 - AIOTypes.h, [294](#)
- USB_IIRO_4
 - AIOTypes.h, [294](#)
- USB_IIRO_8
 - AIOTypes.h, [294](#)
- USB_READ_FROM_DEVICE
 - AIOTypes.h, [301](#)
- USB_RO_16
 - AIOTypes.h, [294](#)
- USB_WRITE_TO_DEVICE
 - AIOTypes.h, [301](#)
- USBP_II8IDO4A
 - AIOTypes.h, [295](#)
- UCharArray, [157](#)
 - AIOTypes.h, [157](#)
- USB-AI16-16/sample.cpp
 - main, [394](#)
- USB-AO16-16/sample.cpp
 - main, [394](#)
- USB-DA12-8A/sample.cpp
 - main, [394](#)
- USB-DIO-16/sample.cpp
 - main, [395](#)
- USB-DIO-32/read_and_write_sample.c
 - main, [398](#)
- USB-DIO-96/read_and_write_sample.c
 - find_dio_96, [400](#)
 - main, [400](#)
- USB_AI16_Family, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
- USB_AI16_Family, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
 - AIOTypes.h, [157](#)
- USB_AIO16_Family, [159](#)
 - AIOTypes.h, [159](#)
 - AIOTypes.h, [159](#)
 - AIOTypes.h, [159](#)
 - AIOTypes.h, [159](#)
 - AIOTypes.h, [159](#)
- USB_AO16_Family, [162](#)
 - AIOTypes.h, [162](#)
 - AIOTypes.h, [162](#)
 - AIOTypes.h, [162](#)
 - AIOTypes.h, [162](#)
 - AIOTypes.h, [162](#)
- USB_CTR_15_Family, [165](#)
 - AIOTypes.h, [165](#)
 - AIOTypes.h, [165](#)
 - AIOTypes.h, [165](#)
 - AIOTypes.h, [165](#)
 - AIOTypes.h, [165](#)
- USB_DA12_8A_Family, [167](#)
 - AIOTypes.h, [167](#)
 - AIOTypes.h, [167](#)
 - AIOTypes.h, [167](#)
 - AIOTypes.h, [167](#)
 - AIOTypes.h, [167](#)
- USB_DA12_8E_Family, [169](#)
 - AIOTypes.h, [169](#)
 - AIOTypes.h, [169](#)
 - AIOTypes.h, [169](#)
 - AIOTypes.h, [169](#)
 - AIOTypes.h, [169](#)
- USB_DIO_16_Family, [171](#)
 - AIOTypes.h, [171](#)
 - AIOTypes.h, [171](#)
 - AIOTypes.h, [171](#)
 - AIOTypes.h, [171](#)
 - AIOTypes.h, [171](#)
- USB_DIO_32_Family, [174](#)
 - AIOTypes.h, [174](#)
 - AIOTypes.h, [174](#)
 - AIOTypes.h, [174](#)
 - AIOTypes.h, [174](#)
 - AIOTypes.h, [174](#)
- USB_DIO_Family, [176](#)
 - AIOTypes.h, [176](#)
 - AIOTypes.h, [176](#)
 - AIOTypes.h, [176](#)
 - AIOTypes.h, [176](#)
 - AIOTypes.h, [176](#)
- USBDevice, [178](#)
 - altset, [179](#)
 - conf, [179](#)
 - debug, [179](#)
 - device, [179](#)
 - deviceDesc, [179](#)
 - deviceHandle, [179](#)
 - iface, [179](#)
 - origconf, [179](#)
 - timeout, [179](#)
 - usb_bulk_transfer, [179](#)
 - usb_control_transfer, [179](#)
 - usb_get_config, [179](#)
 - usb_put_config, [179](#)
 - usb_request, [179](#)
 - usb_reset_device, [179](#)
 - USBDevice.h, [379](#)
 - usbhp_attached, [179](#)
 - verbose, [179](#)
- USBDevice.c
 - AddAllACCESUSBDevices, [377](#)
 - AddDevice, [377](#)
 - CopyUSBDevice, [377](#)
 - DeleteUSBDevice, [377](#)
 - DeleteUSBDevices, [377](#)
 - get_usb_device, [377](#)
 - InitializeUSBDevice, [377](#)
 - NewUSBDevice, [377](#)

- USBDeviceClose, [377](#)
- USBDeviceFetchADCCConfigBlock, [377](#)
- USBDeviceGetIdProduct, [377](#)
- USBDeviceGetUSBDeviceHandle, [377](#)
- USBDevicePutADCCConfigBlock, [377](#)
- USBDeviceSetDebug, [377](#)
- usb_bulk_transfer, [377](#)
- usb_control_transfer, [377](#)
- usb_request, [378](#)
- usb_reset_device, [378](#)
- USBDevice.h
 - AddAllACCESUSBDevices, [379](#)
 - CopyUSBDevice, [379](#)
 - DeleteUSBDevice, [379](#)
 - DeleteUSBDevices, [379](#)
 - get_usb_device, [379](#)
 - INTERNAL_METHOD, [379](#)
 - InitializeUSBDevice, [379](#)
 - LIBUSBArgs, [379](#)
 - NewUSBDevice, [379](#)
 - USBDevice, [379](#)
 - USBDeviceClose, [379](#)
 - USBDeviceFetchADCCConfigBlock, [379](#)
 - USBDeviceGetIdProduct, [379](#)
 - USBDeviceGetUSBDeviceHandle, [379](#)
 - USBDevicePutADCCConfigBlock, [379](#)
 - usb_bulk_transfer, [379](#)
 - usb_control_transfer, [379](#)
 - usb_request, [379](#)
 - usb_reset_device, [379](#)
- USBDeviceArray, [179](#)
 - AIIOUSB::USBDeviceArray, [179](#)
- USBDeviceBase, [179](#)
 - AIIOUSB::USBDeviceBase, [181](#)
- USBDeviceClose
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- USBDeviceFetchADCCConfigBlock
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- USBDeviceGetIdProduct
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- USBDeviceGetUSBDeviceHandle
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- USBDeviceManager, [184](#)
 - AIIOUSB::USB_AI16_Family, [159](#)
 - AIIOUSB::USB_AIO16_Family, [162](#)
 - AIIOUSB::USB_AO16_Family, [164](#)
 - AIIOUSB::USB_CTR_15_Family, [167](#)
 - AIIOUSB::USB_DA12_8A_Family, [169](#)
 - AIIOUSB::USB_DA12_8E_Family, [171](#)
 - AIIOUSB::USB_DIO_16_Family, [173](#)
 - AIIOUSB::USB_DIO_32_Family, [176](#)
 - AIIOUSB::USB_DIO_Family, [178](#)
 - AIIOUSB::USBDeviceBase, [183](#)
 - AIIOUSB::USBDeviceManager, [185](#)
- USBDevicePutADCCConfigBlock
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- USBDeviceSetDebug
 - USBDevice.c, [377](#)
- UShortArray, [190](#)
 - AIIOUSB::UShortArray, [190](#)
- unit_size
 - aio_counts_converter, [71](#)
 - AIOContinuousBuf, [78](#)
- usb_bulk_transfer
 - USBDevice, [179](#)
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- usb_bulk_xfer
 - AIIOUSB_USB.h, [354](#)
- usb_close
 - AIIOUSB_USB.h, [354](#)
- usb_control_transfer
 - USBDevice, [179](#)
 - USBDevice.c, [377](#)
 - USBDevice.h, [379](#)
- usb_control_xfer
 - AIIOUSB_USB.h, [354](#)
- usb_device
 - AIIOUSBDevice, [83](#)
- usb_free_devices
 - AIIOUSB_USB.h, [354](#)
- usb_get_config
 - USBDevice, [179](#)
- usb_get_devices
 - AIIOUSB_USB.h, [354](#)
- usb_open
 - AIIOUSB_USB.h, [354](#)
- usb_put_config
 - USBDevice, [179](#)
- usb_request
 - USBDevice, [179](#)
 - USBDevice.c, [378](#)
 - USBDevice.h, [379](#)
- usb_reset_device
 - USBDevice, [179](#)
 - USBDevice.c, [378](#)
 - USBDevice.h, [379](#)
- usbplp_attached
 - USBDevice, [179](#)
- use_maxcount
 - options, [149](#)
- Ushort_Array
 - AIOTypes.h, [293](#)
- ushort_array, [190](#)
 - size, [190](#)
- VENDOR_REQUEST_begin
 - AIOTypes.h, [300](#)
- VENDOR_REQUEST_end
 - AIOTypes.h, [300](#)
- VALID_ENUM
 - AIOTypes.h, [291](#)
- VALID_PRODUCT
 - AIOTypes.h, [291](#)
- VENDOR_REQUEST
 - AIOTypes.h, [299](#)
- VERBOSE_LOGGING
 - TestCaseSetup.h, [383](#)
- VERSION_DATE
 - AIIOUSB::USBDeviceManager, [189](#)
- VERSION_NUMBER
 - AIIOUSB::USBDeviceManager, [189](#)
- valid
 - AIIOUSBDevice, [85](#)
- valid_config_block
 - AIIOUSB_ADC.c, [308](#)
- value
 - lookup, [147](#)
- valuedouble
 - cJSON, [118](#)
- valueint
 - cJSON, [118](#)
- valuestring
 - cJSON, [118](#)
- verbose
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
 - USBDevice, [179](#)
- volts

- AIOUSB::OutputVoltagePoint, [151](#)
 - TestCaseSetup, [156](#)
- voltsToCounts
 - AIOUSB::AnalogInputSubsystem, [103](#)
 - AIOUSB::AnalogIORange, [109](#)
 - AIOUSB::AO16_AnalogOutputSubsystem, [115](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [131](#)
- WDGVals_begin
 - AIOUSB_WDG.h, [356](#)
- WDGVals_end
 - AIOUSB_WDG.h, [356](#)
- WITH_DATA
 - AIOTypes.h, [293](#)
- WARN_LEVEL
 - TestCaseSetup.h, [383](#)
- WDG_GetStatus
 - AIOUSB_WDG.c, [355](#)
 - AIOUSB_WDG.h, [356](#)
- WDG_Pet
 - AIOUSB_WDG.c, [355](#)
 - AIOUSB_WDG.h, [356](#)
- WDG_SetConfig
 - AIOUSB_WDG.c, [355](#)
 - AIOUSB_WDG.h, [356](#)
- WDGBytes
 - AIOUSBDevice, [84](#)
- WDGVals
 - AIOUSB_WDG.h, [356](#)
- wdgbuf
 - AIOWDGConfig, [86](#)
- what
 - Error, [146](#)
- with_timing
 - AIOCommandLineOptions, [76](#)
 - opts, [150](#)
- work
 - AIOContinuousBuf, [78](#)
- worker
 - AIOContinuousBuf, [78](#)
- workerBusy
 - AIOUSBDevice, [85](#)
- workerResult
 - AIOUSBDevice, [85](#)
- workerStatus
 - AIOUSBDevice, [85](#)
- wrappers, [23](#)
- write
 - AIOUSB::DigitalIOSubsystem, [140](#)
 - AIOUSB::DIOStreamSubsystem, [144](#)
- write_sample.c
 - NO_DEVICE_FOUND, [408](#)
 - SUCCESS, [408](#)
 - USB_ERROR, [408](#)
- write_clock_rate
 - config_options, [119](#)
- write_sample.c
 - BITS_PER_BYTE, [407](#)
 - DEVICES_REQUIRED, [407](#)
 - EXIT_CODE, [407](#)
 - find_dio_96, [408](#)
 - fnd, [408](#)
 - MASK_BYTES, [407](#)
 - MAX_DIO_BYTES, [407](#)
 - MAX_NAME_SIZE, [407](#)
 - main, [408](#)
- writeBuffer
 - DeviceInfo, [133](#)
 - TestCaseSetup, [155](#)
- writeConfig
 - AIOUSB::AnalogInputSubsystem, [91](#)
- WriteConfigBlock
 - AIOUSB_ADC.c, [305](#)
 - AIOUSB_ADC.h, [324](#)
- writeCounts
 - AIOUSB::AnalogOutputSubsystem, [111](#)
- writeValues
 - AIOUSB::DigitalIOSubsystem, [140](#)
- writeVolts
 - AIOUSB::AO16_AnalogOutputSubsystem, [113](#)
 - AIOUSB::DA12_AnalogOutputSubsystem, [129](#), [131](#)
- YAML
 - AIOUSB_Properties.h, [352](#)